程序员加油站系列图书

Windows下的C/C++高级编程

朱磊　周彬　编著

人民邮电出版社

前　言

Windows 操作系统是 Microsoft 公司开发的图形界面操作系统，因其友好的界面、简易的操作和强大的功能而深受用户的欢迎，为当今绝大多数的 PC 机所采用。

也正因此，Windows 下的开发工作也越来越受到人们的重视，许多程序员纷纷从传统的字符环境下的编程转入 Windows 图形界面环境下的编程，许多公司也为他们设计了多种相应的开发平台。

对于传统的 C/C++程序员而言，Microsoft 公司的 Visual C++集成开发环境以及相应的支持库 MFC（Microsoft Foundation Class——微软基础类库）是一个十分不错的选择。它使用方便并且功能强大，为许多 Windows C/C++程序员所喜爱。

但是，由于 MFC 对底层的 Windows 函数调用进行了封装，所以也隐藏了 Windows 系统内部的许多秘密，从而使得许多程序员虽然能利用 MFC 轻松的开发 Windows 下的应用程序，但是对其中的许多细节却不甚了解。

Windows API（Application Programming Interface），即 Windows 下的应用程序编程接口，由 Windows 下基本的系统函数调用组成，专门为 Windows 下的编程提供支持。就高级编程语言而言，Windows API 是 Windows 环境下底层的函数调用。

既然位于底层，完全使用 API 编程便显得十分繁琐，事倍功半。但是使用底层的函数可以使程序员了解 Windows 内部的秘密，知道 Windows 程序的来龙去脉，这样编程的水平才能有进一步的提高，才能实现“知其然更知其所以然”。

本书讲述 Windows 环境下的 C 语言 API 编程。其中，第 1 章介绍 Windows 下 C 语言编程的基本知识，第 2 章到第 8 章是基础的部分，分别介绍窗口和消息、文本显示、图形显示、键盘操作、鼠标操作、子窗口控制与常用控件和对话框，第 9 到第 13 章是相对提高的部分，分别讲述内存管理、文件系统、多文档界面、多任务和多线程以及 Windows Socket 编程。书中的每一章都有一些短小的实例程序，均使用 Windows API 编写而成，并在 Microsoft Visual C++ 6.0 环境下调试并编译通过，读者可以在 http://www.ucbook.com/download/vc.zip 下载。

虽然我们尽了最大的努力，但由于水平有限，本书仍然难免存在一些不足之处，希望大家批评指正。

最后，真诚的希望本书能对 Windows 下的 C/C++程序员有所帮助。

编者

2002 年 4 月

目　录

第 1 章　Windows 基础 ····· 1
1.1　Windows 的简单历史 ····· 1
1.2　第一个 Windows 程序 ····· 2
1.2.1　关于 Windows API 编程 ····· 2
1.2.2　编写第一个 Windows 程序 ····· 2
1.2.3　对 FirstProgram.c 的分析 ····· 3
1.3　Windows 与 Unicode ····· 5
1.3.1　字符集 ····· 5
1.3.2　C 运行库对 Unicode 的支持 ····· 6
1.3.3　Windows 与 Unicode ····· 7
第 2 章　窗口和消息 ····· 11
2.1　HELLOWIN 程序 ····· 11
2.2　一些新概念 ····· 14
2.2.1　Windows API 函数调用说明 ····· 14
2.2.2　新的数据类型 ····· 15
2.2.3　匈牙利表示法 ····· 16
2.3　注册窗口类 ····· 17
2.4　创建窗口 ····· 21
2.5　显示窗口 ····· 22
2.6　消息循环 ····· 23
2.7　窗口函数与消息处理 ····· 25
2.8　WM_PAINT 消息 ····· 26
2.9　WM_DESTROY 消息 ····· 27
2.10　Windows 编程的关键 ····· 27
第 3 章　文本显示 ····· 29
3.1　显示的时机 ····· 29
3.1.1　发送 WM_PAINT 消息 ····· 29
3.1.2　重绘的范围 ····· 30
3.2　简单的文本显示 ····· 33
3.2.1　窗口坐标 ····· 33
3.2.2　文本和背景的颜色 ····· 34
3.2.3　设置背景显示模式 ····· 35
3.2.4　选择字体 ····· 35
3.2.5　字符大小 ····· 40

3.2.6 显示文本的例子……41
3.3 滚动条的应用……45
3.3.1 滚动条基础……45
3.3.2 滚动条的范围和位置……46
3.3.3 滚动条消息……47
3.3.4 添加滚动条……49
3.3.5 改善滚动条……55
3.4 关于文本显示的更多内容……65
3.4.1 更多的文本输出函数……66
3.4.2 Windows 中的字体……70
3.4.3 逻辑字体……71
第 4 章 图形显示……79
4.1 图形设备接口……79
4.2 设备描述表……80
4.2.1 获取设备描述表句柄……80
4.2.2 设备描述表属性……82
4.2.3 保存设备描述表……83
4.3 设备坐标系和逻辑坐标系……85
4.4 映射模式……90
4.4.1 窗口和视图……90
4.4.2 Windows 的映射模式……91
4.4.3 设备描述表信息……102
4.5 关于色彩……106
4.6 画笔、刷子和空隙的填充……109
4.6.1 画笔的使用……109
4.6.2 刷子的使用……111
4.6.3 填充空隙……113
4.7 画点和线条……114
4.7.1 绘制像素点……114
4.7.2 绘制线条……114
4.7.3 绘制线条的示例程序……117
4.8 绘制填充图形……121
4.9 矩形函数、区域和裁剪……132
4.9.1 矩形函数……132
4.9.2 区域……133
4.9.3 区域和裁剪……136
4.9.4 区域和裁剪的示例程序……137
4.10 绘图模式……140
第 5 章 键盘……143
5.1 键盘基础……143

5.2 键盘消息……145
5.2.1 击键消息……145
5.2.2 击键消息示例程序……148
5.2.3 字符消息……156
5.2.4 字符消息示例程序……158
5.3 活动窗口和输入焦点……164
5.4 插入标记和输入焦点……164
5.4.1 插入标记的概念……164
5.4.2 插入标记示例程序……166
第6章 鼠标……173
6.1 鼠标基础……173
6.2 鼠标消息……173
6.2.1 鼠标的客户区消息……173
6.2.2 非客户区消息……177
6.3 来自鼠标消息的附加消息……178
6.3.1 用户区的信息细节……178
6.3.2 非用户区的信息细节……182
6.4 鼠标光标形状……183
6.5 一个鼠标的例子程序……183
第7章 子窗口与常用控件……187
7.1 子窗口……187
7.1.1 创建子窗口……188
7.1.2 一般子窗口的示例程序……192
7.2 常用控件……204
7.2.1 常用控件的一般操作……204
7.2.2 几种常用控件……206
7.2.3 常用控件的示例程序……216
第8章 对话框……227
8.1 模态对话框和非模态对话框……227
8.2 通用对话框库……228
8.3 通用对话框……231
8.3.1 颜色对话框……231
8.3.2 字体对话框……234
8.3.3 查找和替换对话框……237
8.3.4 打开文件和另存为对话框……241
8.3.5 页面设置对话框……245
第9章 内存管理……253
9.1 Windows 内存的结构……253
9.2 虚拟内存……255
9.2.1 虚拟内存的检测……255

9.2.2 虚拟内存操作……259
9.3 应用程序堆……260
9.3.1 堆的生命周期……261
9.3.2 堆的内部情况……265
9.4 内存扩展……268
9.4.1 实用函数……268
9.4.2 文件映射……268
第 10 章 文件系统……271
10.1 创建和打开文件……272
10.2 读出和写入文件……273
10.3 创建和使用临时文件……275
10.4 搜索文件……276
10.5 监控目录中的变化……278
10.6 异步读写文件……282
第 11 章 多文档界面……285
11.1 MDI 概述……285
11.2 MDI 应用程序结构……286
11.3 MDI 支持……287
11.3.1 MDI 函数……287
11.3.2 MDI 消息……289
11.4 MDI 示例程序……290
11.4.1 程序……290
11.4.2 资源……304
11.4.3 程序初始化……305
11.4.4 框架窗口消息处理……305
11.4.5 文档窗口消息处理……307
第 12 章 多任务和多线程……309
12.1 线程与进程……309
12.1.1 进程的描述……309
12.1.2 线程的基本概念……310
12.1.3 线程和进程的比较……310
12.2 多任务的各种模式……311
12.2.1 DOS 下的多任务……311
12.2.2 单用户多任务……311
12.2.3 多用户多任务……312
12.3 Windows 多线程……312
12.3.1 多线程的基本知识……312
12.3.2 线程对象……314
12.4 线程局部存储……328

第 13 章　Windows Sockets 编程……333
13.1　TCP/IP 与 Windows Sockets ……333
13.1.1　TCP/IP……333
13.1.2　Sockets……334
13.2　WinSock 示例程序 ……339
13.2.1　一个简单的聊天程序……339
13.2.2　对 Windows Sockets 的更多了解 ……349

第 1 章　Windows 基础

1.1　Windows 的简单历史

Windows 的起源可以追溯到 Xerox 公司所进行的工作，1970 年，美国的 Xerox 公司成立了著名的研究机构 Palo Alto Research Center（PARC），从事局域网、激光打印机、图形用户接口和面向对象技术的研究，并于 1981 年宣布推出世界上第一个商用的 GUI（图形用户接口）系统——Star 8010 工作站。但如后来许多公司一样，由于种种原因，技术上的先进性并没有给它带来它所期望的商业上的成功。

当时，Apple Computer 公司的创始人之一 Steve Jobs，在参观 Xerox 公司的 PARC 研究中心后，认识到了图形用户接口的重要性以及广阔的市场前景，开始着手进行自己的 GUI 系统的研究开发工作，并于 1983 年研制成功了 Apple Computer 公司的第一个 GUI 系统——Apple Lisa。随后不久，它又推出第二个 GUI 系统 Apple Macintosh。当时，Apple 公司在开发 Macintosh 时，出于市场战略上的考虑，只开发了 Apple 公司自己的微机上的 GUI 系统，而此时，基于 Intel x86 微处理器芯片的 IBM 兼容微机已渐露峥嵘。这样，就给 Microsoft 公司开发 Windows 提供了发展空间和市场。

Microsoft 公司早就意识到建立行业标准的重要性，在 1983 年春季就宣布开始研究开发 Windows，希望它能够成为基于 Intel x86 微处理芯片计算机上的标准 GUI 操作系统。它在 1985 年和 1987 年分别推出 Windows 1.03 版和 Windows 2.0 版。但是，由于当时硬件和 DOS 操作系统的限制，这两个版本并没有取得很大的成功。此后，Microsoft 公司对 Windows 的内存管理、图形界面做了重大改进，使图形界面更加美观并支持虚拟内存。Microsoft 公司于 1990 年 5 月份推出 Windows 3.0 并一炮打红。这个“千呼万唤始出来”的操作系统一经面世便在商业上取得了惊人的成功，不到 6 周时间，Microsoft 公司销出 50 万份 Windows 3.0 拷贝，打破了任何软件产品的 6 周销售记录，从而一举奠定了 Microsoft 公司在操作系统上的垄断地位。

一年之后推出的 Windows 3.1 对 Windows 3.0 作了一些改进，引入了 TrueType 字体技术，这是一种可缩放的字体技术；还引入了一种新设计的文件管理程序，改进了系统的可靠性。更重要的是增加了对象链接嵌入技术（OLE）和多媒体技术的支持。Windows 3.0 和 Windows 3.1 都必须运行于 MS-DOS 操作系统之上。

随后，Microsoft 公司借 Windows 东风，于 1995 年推出了新一代操作系统 Windows 95，它可以独立运行而无需 DOS 支持。Windows 95 是操作系统发展史上一个里程碑式的作品，它对 Windows 3.1 版作了许多重大改进，包括更加优秀的、面向对象的图形用户界面，从而减轻了用户的学习负担；全 32 位的高性能的抢先式多任务和多线程；内置的对 Internet 的支持；更加高级的多媒体支持（声音、图形和影像等），可以直接写屏并很好的支持游戏；即插即用，简化用户配置硬件操作，并避免了硬件上的冲突；32 位线性寻址的内存管理和良好的向下兼容性等。

Windows 98 于 1998 年 6 月发布，它具有许多扩展功能，其中包括性能的提高、更好的硬件支持以及与 Internet 和 World Wild Web 更紧密的结合等。但是，为了提供与之前版本 Windows 的兼容性，Windows 98 保留了部分 16 位的属性。Windows NT 则是完全基于 32 位的系统，它具有更高的安全性以及对 RISC 机器的可移植性等特性，是服务器版本的 Windows 操作系统。

1.2 第一个 Windows 程序

1.2.1 关于 Windows API 编程

本书与一般的 Windows 编程书籍不同，并不使用 C++语言和 MFC，而是使用 C 语言和“原始”的 Windows 应用程序编程接口（API——Application Programming Interface），这些都是比较底层的函数调用，其地位大致相当于字符界面环境下传统 C 语言的标准库函数，只不过其运行环境为 Windows 的图形界面环境。

当然，编写 Windows 应用程序并不是必须使用这种方法，甚至很多程序员会觉得使用 MFC 来编写 Windows 应用程序会方便得多。但是，使用底层的 API 函数编写程序可以使程序员看到许多使用 MFC 编程时看不到的东西，从而对 Windows 的内部有更深入的了解。如果您不甘于仅仅在 MFC 的环境下指指点点，而希望了解一些 Windows 系统更底层的知识，那么，笔者强烈推荐使用 Windows API。

需要注意的是，使用 Windows API 进行编程，程序员需要具备两个条件：

- 熟悉 Windows 环境。
- 了解 C 语言。

否则，贸然进行 Windows 的 API 编程将是一件十分困难的事情，只会事倍功半。另外，还需要一个 Windows 环境下 C 语言的编译器和开发环境。本书中所有的示例程序都在 Microsoft Visual C++ 6.0 环境下编写并编译通过。

当使用 API 函数遇到问题时，可以随时查阅 Windows 的 API 文档。本书所使用的 API 函数全部位于 MSDN 中 Platform SDK 部分。

1.2.2 编写第一个 Windows 程序

几乎所有的高级语言中都有一个经典入门示例程序，那就是“Hello World!”，即在屏幕上显示“Hello World!”语句。该程序在字符环境下的 C 语言版本如下：

```
#include <stdio.h>

main()
{
    printf("Hello World!");
}
```

下面的程序将在 Windows 下显示“Hello World!”。这是一个极其简单的 Windows 程序，它并没有显示传统意义上的窗口，而是简单的通过一个对话框来显示。程序如下：

程序 1-1 FirstProgram.c

```
#include <windows.h>

int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
                                        LPSTR lpcmdLine, int nCmdShow )
{
    MessageBox( NULL, TEXT("Hello World!"), TEXT("FirstProgram"), MB_OK );
    return 0;
}
```

在运行程序后，会弹出如图 1-1 所示的对话框。

图 1-1 Hello World 程序运行效果图

在 Microsoft Visual C++ Developer Studio 中创建该程序的步骤如下：

■ 选择 File 菜单中的 New 选项，在弹出的 New 对话框中单击 Projects 标签，并选择 Win32 Application，表示创建一个 Windows 应用程序工程项目。在 Project Name 域中输入该工程项目的名称，这里为“FirstProgram”。Location 域为工程项目所在的目录，如果不存在，系统会自动创建。然后单击 OK 按钮。

■ 系统会弹出一个对话框，标题为“Win32 Application - Setp 1 Of 1”，选择“An empty project.”，表示创建一个空的工程文件，程序代码由程序员随后输入。然后单击 Finish 按钮。系统会弹出一个总结信息对话框，单击 OK 按钮。

■ 再次选择 File 菜单中的 New 选项，在 New 对话框中单击 Files 标签，并选择 C++ Source File，表示创建一个 C/C++源文件。选中 Add to project:复选框，并指定工程名为“FirstProgram”，表示将该文件加入到 FirstProgram 工程中。在 File 域中输入文件名，这里为“FristProgram.c”，单击 OK 按钮。

■ 手动输入文件代码，或从本书所附带的光盘中复制 FirstProgram.c 的内容到 FirstProgram.c 文件中。

1.2.3 对 FirstProgram.c 的分析

在程序的总体结构上，FirstProgram.c 与传统的 Hello World 程序是一致的。头文件 windows.h 包括了 stdio.h 的作用，WinMain 代替 main 成为程序的入口点，Windows API 函数 MessageBox 代替 C 运行时库函数 printf 显示字符串“Hello World!”。下面逐一对它们进行分析。

■ 头文件

FirstProgram.c 中包含了一个头文件，即 windows.h，实际上，每一个用 C 编写的 Windows 程序都包含该文件。

windows.h 是最主要的头文件，它包含了许多其他的 Windows 头文件，其中比较重要的头文件如下：

WINDEF.H	用于定义基本数据类型
WINBASE.H	用于包含内核函数
WINUSER.H	用于包含用户接口函数
WINGDI.H	用于包含图形设备接口函数
WINNT.H	用于定义支持 Unicode 编码的基本数据类型

这些头文件定义了 Windows 所支持的基本数据类型、数据结构、常用标识符和所提供的系统函数调用。

■ 程序入口点

在传统的 C 语言中，程序的入口点总是 main 函数，而 Windows 程序的入口点总是 WinMain 函数，该函数的定义如下：

```
int WINAPI WinMain(
  HINSTANCE hInstance,      // 当前实例句柄
  HINSTANCE hPrevInstance,  // 总是置为 NULL
  LPSTR lpCmdLine,          // 命令行
  int nCmdShow              // 程序最初的显示模式
);
```

这里出现了许多奇怪的数据类型，我们会在后面陆续介绍。在 MSDN / Platform SDK / Win32 API / Reference / Data Types 中有 Windows 中各种数据类型的定义说明。

WinMain 函数的第一个参数 hInstance 是当前实例句柄。句柄是 Windows 系统中一个极其重要的概念，它是一个整数，用于标识程序、窗口和资源等。在这里，参数 hInstance 用于标识应用程序本身。句柄的概念将在第 2 章中详细阐述。

WinMain 函数的第二个参数 hPrevInstance 总是被置为 NULL（空）。在 Windows 的早期版本中，当多次同时运行同一程序时，系统会创建该程序的多个“实例”，同一程序的所有实例共享代码和只读内存。程序需要通过检查 hPrevInstance 参数来判断自身的其他实例是否正在运行，以便从已经存在的实例中获取数据。但在 32 位版本的 Windows 中，运行同一程序的方式改变了，所以不再需要 hPrevInstance 参数，而总是将其置为 NULL。

WinMain 的第三个参数 lpCmdLine 是运行程序的命令行。用户可以单击“开始”菜单中的“运行”选项，然后输入命令来运行一个程序，并且可以指定运行程序所需的参数，这里的命令就由参数 lpCmdLine 传入。

WinMain 的第四个参数 nCmdShow 用于指定程序窗口最初的显示模式，可以正常显示，也可以在初始时就最大化或最小化显示。

这里变量的名称使用了一种称为“匈牙利命名法”的命名规则，即在变量名前加一个标识变量数据类型的短前缀。例如，前缀 i 表示一个整型（int）变量，前缀 h 表示变量是一个句柄（handle）。关于“匈牙利命名法”我们将在第 2 章中详细阐述。

WinMain 函数的返回值是一个整数（int）。但是这里还有一个 WINAPI 标志，该标志用于指定一种调用约定，在 WINDEF.H 加以定义：

```
#define WINAPI __stdcall
```

它定义了一种生成机器代码和放置函数参数的方式。

■ MessageBox 函数

MessageBox 是一个 Windows API 函数，它通过弹出一个对话框来显示一些简短的信息。

该函数的定义如下：

```
int MessageBox(
    HWND hWnd,              // 所属窗口句柄
    LPCTSTR lpText,         // 需要显示的信息字符串
    LPCTSTR lpCaption,      // 对话框窗口标题字符串
    UINT uType              // 对话框中的按钮和提示图标类型
);
```

MessageBox 函数的第 1 个参数通常是一个窗口句柄，用以标明对话框所属的窗口。我们将在第 2 章中详细阐述。

MessageBox 函数的第 2 个参数是需要显示的信息，这是一个字符串，是对话框显示的主要部分。

MessageBox 函数的第 3 个参数是对话框的标题字符串，将作为对话框的标题显示。FristProgram.c 中这两个字符串都经过 TEXT 宏封装处理，主要是为了提供对 Unicode 编码的支持，这将在下一节加以解释。

MessageBox 函数的第 4 个参数用于指定对话框中显示的按钮和提示图标的类型。常用的按钮类型有 MB_OK（“确定”按钮）、MB_OKCANCEL（“确定”和“取消”按钮）和 MB_YESNO（“是”和“否”按钮）等。常用的提示图标类型有 MB_INFORMATION（“提示”图标）、MB_WARNING（“警告”图标）和 MB_ERROR（“出错”图标）。各种按钮和提示图标类型可以用“|”连接，表示同时使用。

1.3 Windows 与 Unicode

Windows 下的 C 编程与传统的 C 编程还有一个十分重要的不同之处，那就是对字符的处理问题。

1.3.1 字符集

从计算机发展的初期开始，对字符的定义与处理问题就一直存在。经过一段时间的探索与争论，最终由美国国家标准局颁布了一种通行的 7 位编码标准，这就是著名的 ASCII（美国国家标准信息交换码，American National Standard Code for Information Interchange）。ASCII 码采用 7 位编码，所以总共有 128 个代码。该编码具有许多优点，因此得到了极其广泛的应用。

ASCII 是一个地道的美国标准的编码，虽然对美国的本土应用提供了很好的支持，但是并不能很好的支持其他使用英语的国家。例如，某些国家的语言中会使用到读音符号，而类似£和¥等英语和拉丁语所用的符号在 ASCII 中也没有。为了满足这种需求，人们试着增加了一些符号。另一方面，在 PC 机中，一个字节的大小是 8 位，而 ASCII 码只使用了其中的低 7 位，于是人们把最高位也用上，并最终由美国国家标准局（ANSI）和国际标准化组织（ISO）共同制订了一个 8 位编码标准，为 Windows1.0 所采用，通常称为 Windows ANSI 字符集。由于 ANSI 采用 8 位编码，所以字符比 ASCII 多一倍，有 256 个。

但是，这仍然不能满足需求。随后，人们又提出了代码页的概念，允许不同的语言使用

不完全一样的编码，但是效果并不理想。

一个很自然的想法，既然一个字节不够用，那就多用几个字节。的确，随后人们陆续提出了几种多字节的编码方案（通常不多于两个字节），这些方案统称为“宽字符集”编码方案，其中比较成功的一种就是 Unicode 编码。

Unicode 是 Apple 和 Xerox 公司于 1988 年建立的一个技术标准，并由一批业界顶尖的企业共同成立了一个集团机构来负责 Unicode 的开发和推广应用，并维护 Unicode 标准。

Unicode 提供了一种简单而又一致的表示字符串的方法。Unicode 字符串中的所有字符都是 16 位的（两个字节）。它没有像一般的宽字符集编码方案那样，使用专门的字节来指明下一个字节是属于同一个字符的组成部分，还是一个新字符。这意味着可以对所有的字符一视同仁，而不再需要对其中的部分字符进行特殊处理。由于 Unicode 用一个 16 位的值来表示每个字符，因此总共可以得到 65000 个字符，这样，它就能够对世界各国的书面文字中的所有字符进行编码，远远超过了单字节字符集的 256 个字符的数目。目前，已经为阿拉伯文、中文拼音、西里尔字母（俄文）、希腊文、西伯莱文、日文、韩文和拉丁文（英文）字母定义了 Unicode 代码点。这些字符集中还包含了大量的标点符号、数学符号、技术符号、箭头、装饰标志、区分标志和其他许多字符。如果将所有这些字母和符号加在一起，总计约达 35000 个不同的代码点，这样，总计 65000 多个代码点中，还有大约 29000 个可供将来扩充时使用。另外，大约有 6000 个代码点是保留供个人使用的。

1.3.2 C 运行库对 Unicode 的支持

■ 数据类型

Unicode 并没有改变 C 中字符（char）数据类型的定义，char 仍然表示一个字节（8 位）。C 通过定义一种新的数据类型来提供对 Unicode 的支持，这种新的数据类型为 wchar_t，其定义如下：

```
typedef unsigned short      wchar_t;
```

即将 wchar_t 定义为与无符号短整型相同，都是两个字节（16 位）。

定义 Unicode 字符、指向 Unicode 字符串的指针和 Unicode 字符数组方法如下：

```
wchar_t    wChar = 'W';
wchar_t    *wPoint = L"This is a Unicode string.";
static wchar_t     wChars[ ] = L"This is a Unicode string.";
```

其中，字符串前面的 L 指示编译器将字符串按 Unicode 保存，即每个字符占 2 个字节，于是字符串将占用 52 个字节的存储空间，即如果调用 sizeof（wChars），将返回 52。

为了将单字节字符扩充为双字节字符，系统会自动在单字节字符后差入 0。例如，单字节字符 W 在 Intel 处理器内存储为 57，而双字节字符 W 则存储为 57 00。

■ 字符串处理函数

对双字节字符串的处理也与对单字节字符串的处理不同。以计算字符串长度为例，计算单字节字符串长度使用函数 strlen。

```
iStringLength = strlen( "This is a Unicode string." );
```

iStringLength 的值将为 26。而计算双字节字符串长度需要使用新定义的 wcslen 函数。

```
iwStringLength = wcslen( L"This is a Unicode string." );
```

iwStringLength 的值也将为 26。不能使用 strlen 函数计算双字节字符串的长度，否则会出现编

译警告（类型不匹配）。

类似地，所有字符串处理函数都需要重新定义一个新的版本来支持 Unicode。为了维护代码的统一性，Microsoft 公司提供了 TCHAR.H 头文件来实现对单字节和双字节字符串的统一处理，使用该文件中定义的字符串处理函数，可以根据是否定义了 UNICODE 宏来分别处理 Unicode 字符集或单字节字符集。

仍然以计算字符串长度为例，处理单字节和双字节字符串的统一函数为_tcslen，在 TCHAR.H 中的定义大致如下（因为该头文件不是 ANSI C 标准的一部分，所以这里定义的宏和函数名之前都有一条下划线）：

```
#ifdef _UNICODE
    #define _tcslen   wcslen
#else
    #define _tcslen   strlen
#endif
```

也就是说，如果定义了 UNICODE，就定义_tcslen 为 wsclen，否则（未定义 UNICODE），就定义_tcslen 为 strlen。

另外，TCHAR.H 中还定义了一个新的数据类型 TCHAT 来解决两种字符数据类型的统一问题，其定义大致如下：

```
#ifdef _UNICODE
    typedef wchar_t  TCHAR;
#else
    typedef char     TCHAR;
#endif
```

也就是说，如果定义了 UNICODE，就定义 TCHAR 为 wchar_t，否则就定义 TCHAR 为 char。

另外，为了解决字符串前 L（指定为双字节字符串）的问题，TCHAR.H 中还定义了几个宏。

```
#ifdef _UNICODE
    #define __T(x)        L ## x
#endif
```

大致的意思就是如果定义了 UNICODE，就以 L 开头的字符串替代以__T()宏所包围字符串（即__T()宏同样具有将单字节字符串转换为双字节字符串的功能）。另外还有两个宏与__T 宏相同：

```
#define _T(x)          __T(x)
#define _TEXT(x)       __T(x)
```

这 3 个宏都具有将单字节字符串转换为双字节字符串的功能，可任意选用。

1.3.3 Windows 与 Unicode

■ Windows 98 与 Unicode

Windows 98 不是全新的操作系统。为了与之前的 Windows 系统保持兼容，它保留了一些 16 位 Windows 操作系统的特性，但它不是用来处理 Unicode 的。如果要增加对 Unicode 的支持，工作量非常大，因此在该产品的特性列表中没有包括这个支持项目。由于这个原因，

Windows 98 像它的前任产品一样，几乎都是使用 ANSI 字符串来进行所有的内部操作的。虽然也可以编写用于处理 Unicode 字符和字符串的 Windows 98 应用程序，但这十分困难。可以说，Windows 98 只在很少的地方对 Unicode 提供了一定程度的支持，仅有少数 Unicode 版本的 Windows API 函数可以在 Windows 98 下正常运行，这其中就包括 MessageBox 函数。

■ Windows 2000 与 Unicode

在 Windows 2000 的开发过程中，始终都注意了对 Unicode 的支持，用于创建窗口、显示文本、进行字符串操作等的所有核心函数都需要 Unicode 字符串。如果调用任何一个 Windows 函数并给它传递一个 ANSI 字符串，那么系统首先要将字符串转换成 Unicode，然后将 Unicode 字符串传递给操作系统。如果希望函数返回 ANSI 字符串，系统就会首先将 Unicode 字符串转换成 ANSI 字符串，然后将结果返回给用户的应用程序。所有这些转换操作都是在用户看不见的情况下发生的。当然，进行这些字符串的转换需要占用系统的时间和内存。

■ 数据类型

前面我们提到，几乎所有的 Windows 程序都包含了 WINDOWS.H 头文件，其中包含的 WINNT.H 文件负责提供对 Unicode 的支持。在 WINNT.H 中，分别为单字节和双字节编码定义了两套数据结构，单字节编码数据类型如下：

```
typedef char        CHAR;                    // 字符
// 以下用作字符指针
typedef CHAR        *PCHAR;
typedef CHAR        *LPCH, *PCH;
typedef CONST       CHAR *LPCCH, *PCCH;
// 以下用作字符串指针
typedef CHAR        *NPSTR;
typedef CHAR        *LPSTR, *PSTR;
typedef CONST       CHAR *LPCSTR, *PCSTR;
```

双字节编码数据类型如下：

```
typedef wchar_t     WCHAR;                   // 字符
// 以下用作字符指针
typedef WCHAR       *PWCHAR;
typedef WCHAR       *LPWCH, *PWCH;
typedef CONST       WCHAR *LPCWCH, *PCWCH;
// 以下用作字符串指针
typedef WCHAR       *NWPSTR;
typedef WCHAR       *LPWSTR, *PWSTR;
typedef CONST       WCHAR *LPCWSTR, *PCWSTR;
```

并在此基础上定义了一套统一的数据结构，如下所示：

```
#ifdef  UNICODE
    typedef WCHAR TCHAR, *PTCHAR;            // 字符及字符指针
    // 以下为字符串指针
    typedef LPWSTR LPTCH, PTCH;
    typedef LPWSTR PTSTR, LPTSTR;
```

```
    typedef LPCWSTR LPCTSTR;
#else
    typedef char TCHAR, *PTCHAR;                    // 字符及字符指针
    // 以下为字符串指针
    typedef LPSTR LPTCH, PTCH;
    typedef LPSTR PTSTR, LPTSTR;
    typedef LPCSTR LPCTSTR;
#endif
```

为了处理字符串前的L（指定为双字节字符串），还定义了__TEXT宏。

```
#ifdef  UNICODE
    #define __TEXT(quote)     L##quote
#else
    #define __TEXT(quote)     quote
#endif
```

另外还定义了一个与__TEXT一样的宏TEXT。

```
#define TEXT(quote) __TEXT(quote)
```

二者作用相同，都是将单字节编码转换为双字节编码，可任选其一。

如果明确需要编写ASCII编码的程序，就使用单字节编码的数据类型体系；如果明确需要编写Unicode编码的程序，就使用双字节编码的数据类型体系；如果不确定使用何种编码，则应当使用统一的数据类型体系，而由将来是否定义了UNICODE来决定解释为何种编码。

■ 涉及字符串的函数

类似地，Windows也针对单字节和双字节字符串，分别为涉及字符串的函数定义了两种不同的版本。以MessageBox函数为例，显示单字节字符串信息的消息框函数为MessageBoxA，其定义如下：

```
WINUSERAPI
int
WINAPI
MessageBoxA(
    HWND hWnd,
    LPCSTR lpText,          // 单字节字符串
    LPCSTR lpCaption,       // 单字节字符串
    UINT uType);
```

而显示双字节字符串信息的消息框函数为MessageBoxW，其定义如下：

```
WINUSERAPI
int
WINAPI
MessageBoxW(
    HWND hWnd,
    LPCWSTR lpText,         // 双字节字符串
    LPCWSTR lpCaption,      // 双字节字符串
```

```
    UINT uType);
```

不难看出，前者的两个字符串参数均为单字节字符串，而后者的两个字符串参数均为双字节字符串。

在此基础上，还定义了一个统一的消息框函数，即 MessageBox 函数，其定义如下：

```
#ifdef UNICODE
#define MessageBox    MessageBoxW
#else
#define MessageBox    MessageBoxA
#endif
```

其他涉及字符串的函数与此类似。但是，在 Windows 98 中不能运行 Unicode 版本的 Windows 函数，MessageBox 函数是极少数例外的函数之一。

本书中的实例程序均使用统一的数据类型和 API 函数，均可在 ASCII 版本和 Unicode 版本下编译通过。所有版本的程序均可在 Windows NT / Windows 2000 系统下正常执行，但基本上所有的 Unicode 版本的程序都不能在 Windows 98 下正常运行。