前　言

设计计算机或者说设计计算机软件时，最艰巨的事情就是确定整个任务与最终用户间的关系。处理器把越来越多的时间用于维护用户接口，而不是完成计算机认为工作量相当大的任务。数年前，当一台 Nebraska 档次的计算机经过几天的"深思熟虑"最后得出一个简单的数字时，身着白大褂的整个房间的人都会欢呼雀跃。然而今天，这种现象已经一去不复返了。

即便是配有鼠标和非常合适的监视器的个人计算机也并不能用人类使用的名词术语和人类进行通讯。如果你的计算机有足够的内存的话，Windows 与 DOS 相比可以比较容易地和人类共处，但是它依然坚持使用计算机独特的方式来表达自己。

在 Windows 应用中，多媒体技术为软件设计人员提供了一种与实际用户接口更接近一点的处理方式。比如，它使得你能够编写可以"唱歌"的软件。换句话说，你的应用软件能够说话、显示图形、播放音乐等等。尽管从前这一直是科幻电影里才出现的场面，但是现在却由计算机变成了现实。

除了提供实现这些功能的简单地方法，Windows 下的多媒体应用还提供了一系列驱动程序和数据格式的标准序列。你可以使用它们完成很多任务。这也意味着即便是运行在最小硬件配置的 Windows 下的软件也能够"唱歌"。和大多数 Windows 的设备无关特性一样，你能够在即使没有看到过设备的情况下编写支持多种选择的应用软件。

当多媒体一词出现在某本杂志的文章里或者"CNN 内容提要"里时，它很象一名绝望中的商人在向你推销一个普通的接口卡而不是微机功能的突破性的扩展。实际上，多媒体是一种你越来越喜欢的工具。当你第一次在计算机屏幕上观看电影或者第一次听到计算机驱动的音响中演奏出真正动听的管弦乐时，你就会感觉到到目前为止现有的计算机接口是多么的糟糕。

Windows 下多媒体最值得骄傲的方面就是它是研究你使用计算机所能实现的最有趣的事情。更确切的说，它既有创造力又处于技术前沿。你可以连续数天"玩"计算机而不感到丝毫内疚。你能够欣赏电影 Montry Python 中的音乐或看看电影，绝没有人指出你缺乏生活深度。

还有，当你终于完成并且发表你的关于多媒体应用的研究成果时，各种各样的人就能分享你的快乐，并且浪费和你研究时差不多的时间去玩多媒体。在最极端的情况下，多媒体能够为整个西方世界带来欢乐，甚至可停止全部有成效的工作。

尽管你可能是出于某种特定的原因而购买的本书，比如说你是想学习如何使用波形文件或者学习如何使用 Photo－CD，但是值得指出的是多媒体具有良好的开放特性。多媒体最好的定义恐怕应该是能够存放在贴有"Multimedia"标签的包装箱里的所有设备。可以把多媒体看做一张空白的画布，你能够让你的想象力在那里闪光。

本书将会为你提供 Windows 下一些最有价值的多媒体的基本函数。由于这些函数的模块化的本质，你可以把它们编写成你能想象得到的任何应用。多媒体的出现意味着很多人类尚未涉足的领域将要被研究。

编写本书时，我尽量选择 Windows 下多媒体的相对主流和实用的方面加以研究。我也尽量把解释的深度与大多数软件编程人员的要求保持一致。本书不计划讲述它所涉及到的专业

的全部内容，这是因为考虑到大多数编程人员也并无此要求，而且"全部内容"的概念也实在太大了。

最后需要说明的是任何书籍的写作的时间与被读者阅读的时间总有一定的时间间隔。具体到本书，我想告诉读者其中讨论过的软件开发包的文本说明中肯定存在错误。Microsoft 的《多媒体开发软件包》的使用手册是本书中应用软件的主要来源，其中也存在一些错误。如果你完全相信这本手册的话，每个错误都会浪费你不少的时间。当你发现本书中的说明与 Microsoft 中的说明相抵触时，信任本书将是明智的选择。

本书编写完毕到你阅读的时间间隔使得某些公司可能在这个间隔里已经声明并改正了文本中的错误。然而一般来说，在大公司里只有错误很严重时才得到重视。

目　录

第一章　为什么要使用多媒体

目前关于多媒体还存在很多不确定的东西。错误的定义和随意的扩展使得人们把任何不再过分依赖于键盘和不再运行在文本模式的计算机外设都称作多媒体。然而事实上，不管是否合适，这个词是作为一种标记贴在运输计算机的包装箱上的。

Windows 操作系统下的多媒体一词稍微容易定义一些。Windows 3.1 在开发时就使用了一些特定的多媒体扩展，目的是提高 Windows 3.1 的一些高端特性。

Windows 的多媒体扩展提供给程序编写人员一种标准化的和相对不太复杂的方法，以便他们在 Windows 操作系统下实现音响和图形功能。实际上，这些基本的功能可以划分为一些更具体的任务。比如音响是一种独立的工作，实现某一功能时程序编写人员的可选方案比外星人的胳膊都要多。因为人们从来不可能有机会去数一下外星人究竟有多少只胳膊，所以可能对这种比喻感到不太舒服。这种感觉正是 Windows 操作系统下编程人员使用多媒体时所面临的挑战。

实际上，Windows 下的多媒体程序可以分解为一系列相对来说比较容易理解的函数。本书将要讨论这些函数以及其它有帮助的功能，最终使得多媒体使用人员能够根据实际需要进行开发。

本书将要讨论的主要内容包括：

- 播放波形文件
- 显示位映射的图形图像
- 获取 Kodak 光盘图像
- 播放 MIDI 音乐
- 播放 Windows 视频

下面给出以上内容的简要说明，如果你对某个题目感兴趣，请详细阅读有关章节。

- 波形文件：指对声音进行采样的采样数据文件，在 Windows 下的扩展名为.WAV。在合适的硬件（将在以后讨论）支持下和计算机控制下，一个波形文件能够重现出各种声音。从电话音质的声音到 CD 盘音质的声音，无论单声道还是立体声，都可以重现。在硬件支持和计算机控制下，多媒体能够说话、唱歌、产生音响效果甚至以哀怨的声音恳求你不要在文件菜单中选择退出。

- 位映射图形：这些大概是 Windows 下人们更为熟悉的现象——比如图符以及其它东西。我们将会探讨怎样产生和显示位图以使它们成为我们应用的一部分。

- Kodak 光盘：从某种意义上说，Kodak 光盘是位图的一种复杂的形式。光盘是 CD－ROM 的一种，存放着扫描进去的 35mm 的图像。这些图像可以从光盘上读出并由软件使用，你可以提供底片让 Kodak 公司及其代理商为你制作光盘。

- CD 音响：如果你的计算机包含有一台 CD－ROM 驱动器，你也就拥有了一台 CD 播放机。除了能够从开始到结束不停顿地播放一首乐曲外，你还能够用软件控制以 1/75 秒的精度选择播放 CD 盘的一部分。

• MIDI 音乐：MIDI 音乐标准定义了器乐作为数据的标准。它不采样音乐(这与波形文件不同)，但是把每个音符定义为一个数字。MIDI 标准同样定义了不同的音调怎样发声以及有多个音调的音乐怎样发声等等。在 Windows 下，由于 MIDI 有一个标准的乐器音序列，MIDI 越发显得有用。你可以用 MIDI 播放巴赫创作的乐曲并在每次打开文件时，加入听众的声音。

• Video for Windows：这是微软公司关于在 Windows 下的数字电视的一个标准。你可以在你的软件中播放符合 Video for Windows 标准的视频信息，这样做的结果是更加吸引人。

1.1 最大的 CD－ROM

如果你翻到本书封底的内页，你就会发现一般计算机类书籍所不提供的一张随书赠送的 CD－ROM。和一般的 CD－ROM 一样，这张 CD－ROM 也存有大量数据。在它的目录中你会发现：

1. 本书中所有样例程序的源代码。
2. 本书中所有样例程序的可执行文件。
3. Windows 的图形工作站。
4. 大约 20M 字节的公用波形文件。
5. 大约 20M 字节的公用 MIDI 文件。
6. 大约 20M 字节的公用图形文件。
7. 一些 Kodak Photo－CD 图像。
8. 一些 Video forWindows 的剪裁。
9. 半小时的 Loftus。

1.2 软件要求

多媒体的应用有很多种形式。至今大多数开发者经常使用的是一种称为“授权语言”(authoring language)的形式。所谓授权语言是一种高级开发环境。在这种环境下，开发人员可以把应用和命令结构处理成类似于 BASIC 语言与英语的关系。就象其它语言一样，这些语言使用起来不难，写出来的软件也不令人敬畏。但是，规范语言往往使得使用人员局限在语言创始人本人认识范围内。功能的增加(例如读入光盘数据)往往不易实现甚至于不可能。在有些应用场合，规范语言不啻为一种结合多媒体各种设备的好方法。然而当你想用一些新的资源做一些事情时，规范语言不得不做一些妥协、折衷。

最好的方法是用 C 语言来写多媒体的应用程序，在硬件支持下 C 语言的强大功能和可扩展性使得你想完成的事情成为可能。这就是本书要讨论的：在 Windows 环境下多媒体的功能将会大放异彩。

Windows 3.1 平台上多媒体的开发还需要微软公司多媒体开发软件包。在你使用本书的代码前，你应该拥有该软件包。唯一的例外是光盘读入器，它需要 Kodak 公司的开发工具包但不使用微软公司的多媒体开发软件包，详情见第 4 章。在任何 Windows C 语言开发环境下，你可以很好地使用本书中提供的源代码。我曾经在 Windows 使用过 Borland C++ 3.1，但是 Microsoft C++同样适用。Microsoft C 的用户有可能要对某些源代码进行改动以适应于 Mi-

crosoft C 编译器，我将在本章稍后给出说明。

建议读者在 Windows 3.1 下或更高的操作系统下(不包括 Windows 3.0)使用本书中的代码。

1.3　硬件要求

各章需要你提供的硬件稍有差异。本节给出总的情况。

1.3.1　音响(或语音)卡

Windows 多媒体一个最基本的部件就是音响卡。让我们把提高 Windows 的音响功能作为开发多媒体的开端。图 1－1 就是一种典型的音响卡，名字叫做 Omni Labs Audiomaster。然而，还有许多种其它类型的音响卡同样适合于本书中涉及到的应用，例如 Sound Blaster Adlib 卡，Microsoft 公司的音响卡等等。

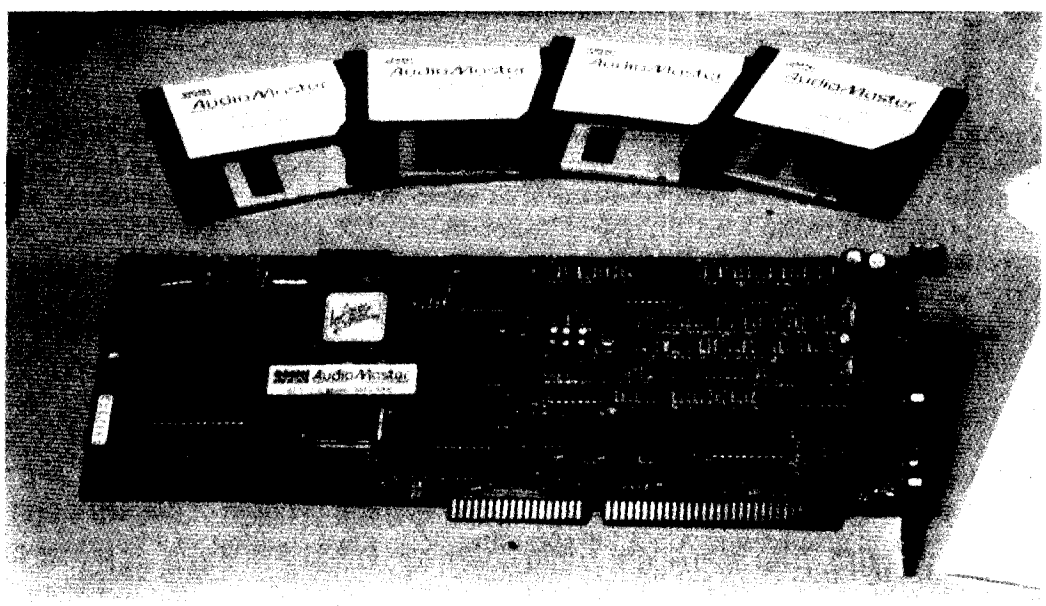

图 1－1　Omni Labs Audio Master 音响卡

总之，由于多种很明显的原因，你应该选择一种与 Windows 兼容的音响卡来实现本书中描述的功能。

在微机内部有一只很小的扬声器，即使是全功率发声时，功率也只有 1/10W。正像把计算机内存设计成 640K 的人应该受到指责一样，设计这个扬声器的人也该受指责。

很多年来一直有不少人想在扬声器上做文章，以便让它不只是在微机发生错误时鸣叫。一个简单的道理就是高质量的音质需要高质量的音响系统。使用 PC 机内部的扬声器或着把音响系统安装在 PC 机内部都是行不通的。

在 PC 机获取高质量音质的最好方法就是彻底抛弃微机上小扬声器并且 安装一个音响接

口，这就是音响卡的由来。该卡可与立体声音响系统或其它合适的放大器相连，并且用和激光唱盘同样的道理产生音响效果。所以毫无疑问，音响卡所产生的声音的质量比 CD 盘的音质毫不逊色。

像图 1－1 所示的音响卡能够对声音实现 PCM 调制(即脉冲编码调制)。这种调制技术虽然需要较长时间去理解，却能实时地产生满足声学定义的声音。实际上，这种技术能够使你以数字方式录制并播放声音，并且获得类似 CD 盘的音质。

1.3.2 CD－ROM 驱动器

多媒体应用中还有一个很重要的部件就是 CD－ROM 驱动器，在第五章中我们将详细讨论 CD－ROM 驱动器的各种能力。图 1－2 是 Sony CDU 31A CD－ROM 驱动器的照片，本书将在以后描述。

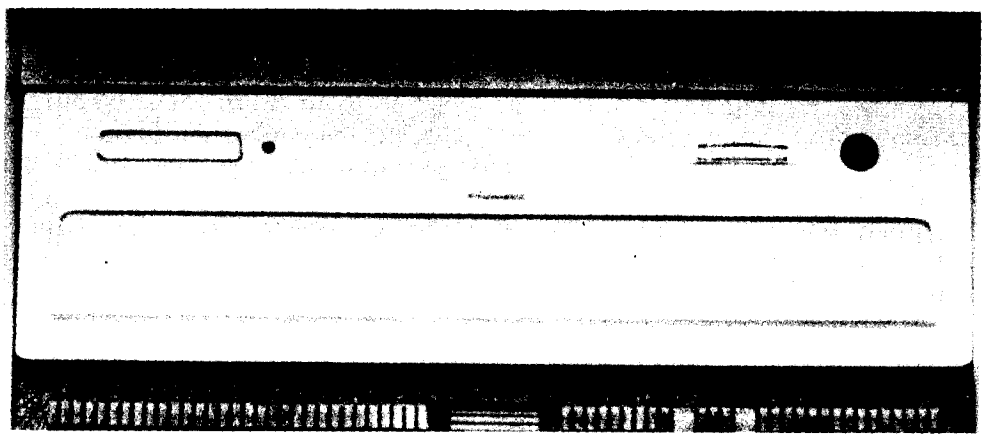

图 1－2 Sony CDU 31A CD－ROM 驱动器

1.3.3 其它硬件

除了上述两个必需的基本硬件外，用户可以根据自己的需要进行配置。比如 MIDI 的进一步应用需要一些 MIDI 工具。你可能需要为你的系统添加一台高级终端显示卡，这样你就可以通过系统欣赏真彩色图像的影碟了。更进一步的应用假定你拥有一台 Camcorder 或其它的视频源及软件 Video forWindows 。你还需要一块合适的视频捕获板(video－capture board)，有关该板在 Video for Windows 软件包中有说明。

需要说明的是，尽管上述硬件中的大多数能够在以 80286 为 CPU 的微机上使用，但是实际上你在这类微机上无法完成你想做的事情。软件 Video for Windows 适合于本书中提到的许多硬件，但是该软件需要相当大的内存和处理器的相当强的处理能力。尽管以 80386 为 CPU 的微机勉强能用，但是我建议最好使用以 80486 为 CPU 的微机。顺便开句玩笑，你可以把旧微机用做鞋架子、咖啡桌子等等。

1.4　Windows 平台上的编程特点

如果你曾经用 C 语言做过 Windows 的编程，那么你可以跳过这节，但是如果你在 Windows 应用方面是个新手，阅读一下本节对你大有裨益。Windows 复杂的外部表现源于它极为复杂的内在本质。

本书的目的既不是教会读者一般的使用 C 语言的编程技巧，也不是教会读者 Windows 应用的基本技巧，有关这些技巧已有专门的论著。如果你尚不熟悉 C 语言，你最好首先去用 C 语言编写一些 DOS 下应用程序。

与 DOS 相比，Windows 能够成为一个良好的工作平台的主要原因是它的内存管理方式。在 DOS 下，有常规内存、扩充内存、扩展内存、虚拟内存和其它方法增加的内存。但在 Windows 下，所有的内存都一样，在用户编写应用程序时无需顾及它们的差异。

毫无疑问，运行 Windows 的计算机应该具有使用很特殊的内存管理方法的处理器，理解 PC 的低水平的内存管理模式如何影响 Windows 的应用是很重要的。

早些时候，微处理器都采用线性内存寻址方式。这意味着内存的低地址从 0 开始直到最高地址，用以寻址的地址值在该空间内变化。与现在的内存管理方式相比，这种方式在内存不大时依然奏效。七十年代后期的在当时说来功能强大的计算机和九十年代初期的烤箱拥有差不多大的内存空间，只是烤箱的微处理器速度要快一些。

8086 系列芯片的问世最终导致了第一代 IBM PC 机的诞生。芯片的设计者们意识到原来的线性寻址方式对 8086 系列不太适用，因为 8086 芯片的寻址范围要大得多。所以如果采用线性寻址就需要一个很大的数值。具体地说，如果 8086 采用线性寻址，它就需要一个 20 位的地址寄存器方能寻址 1M 字节的内存。1M 字节就是 1048576 个字节。8086 的体系结构只允许寄存器为 16 位，而达不到 20 位。实际上任何事情都是一分为二的，寄存器的位数越多，那么运算速度就会越慢。

后来设计者们发明了分段方式存储器。在一个分段处理器内，任何绝对地址都由两个 16 位数据构成，分别称为段地址和偏移量。段地址决定了地址在内存的哪一段，每段以 16 字节为增量。偏移量确定了在某一段内 65535 个字节中的哪些是我们感兴趣的。65535 是 16 位寄存器所能达到的最大值。采用段地址与偏移量相结合起来表达绝对地址的方法与线性存储器寻址相比有一些显著优势。例如，尽管大的复杂的程序必须使用 32 位地址去为内存定位，但是需要内存小于 65535 字节的程序依然能用 16 位地址去定位。这意味着小程序可以运行更快并占用较小空间，在 PC 机的综合应用(或综合应用环境，比如 Windows)时，PC 机内存的分段本质变成工作的不可缺少的部分。在 Windows 下，把小的目标模块与大的目标模块区分开来——标准是看目标模块所占空间是否大于 64K——是 Windows 提高性能的方法之一。当然，这一点也经常令编程人员头疼。

PC 机应用时的内存分段的结果首先是产生了术语“内存模式”。内存模式取决于程序代码和数据的大小，也就是寻址这些代码及数据所用的地址大小。所用地址大小用指针表示，指针指向内存中的代码和数据。

在 Windows 下最简单的模式是“小模式”。小模式程序的代码和它使用的数据都限制在一个内存段内。如果某项 Windows 的应用满足上述条件，那么所有代码和数据都可以用 16 位

指针寻址，我们称之为近(near)指针，这是 Windows 的各种应用模式中最有效率的模式，然而它限制了你所能写的代码的范围。

Windows 操作系统下最常使用的还是中模式(medium model)。在中模式下，用户使用的数据空间限制在一个段内，但是代码空间可以在多个段内。这样，地址指针就是 32 位，或称之为长(far)指针。

在 DOS 操作系统下应用程序编写过程中还有两种其它的内存模式，但是在 Windows 下不是这样直接称呼的。首先谈第一种：大模式(large model)，大模式程序使用长数据指针和长地址指针，指针都是 32 位。

在 DOS 下，大模式理论上能够处理 1M 字节的代码和 1M 字节的数据。但是在总共只能支持 640K 字节寻址内存的环境下，这种说法听起来有些滑稽。

在大模式下，指针由两个整型值构成，一个是段地址，一个是偏移量。在 Windows 下这有两个严重后果。首先在这种安排下，内存的每一个字节都有各个指针值。你可以改变两个整型值，尽管整型值不相同，却指向内存的同一个物理地址。例如，一个段地址为 0 而偏移量为 16 的指针指向内存的第 16 个字节。然而一个段地址为 16 而偏移量为 0 的指针同样指向这个绝对地址，尽管这两这种组合表面上看来不同。实际上却指向了内存的同一位置。第二：在 C 语言中你只能将短整型值与长指针相加而不能将长整型值与长指针相加。原因是指针算术的实际动作涉及到把整型值与长指针的偏移量相加。由于无法进位或借位，当偏移量溢出时便复零，段地址仍保持不变。这意味着大模式指针不能寻址大于所占 65535 字节的单个模块。

最后一种模式就是针对上述情况建立的，我们称之为巨模式 (hugemodel)。在这种模式下，指针也都是 32 位的；但是当你使用指针时，它们能够自动地归一化，这意味着指针变成了一个线性内存指针，一个简单的长整型值，当你使用完毕该指针后，它就自动转换成相应的偏移量和段值。在 C 语言中，所有指针的归一化操作都是透明的。

巨型模式指针的意文在于你可以将指针与其它整型值相加，这样就可以寻址大于 64K 字节的模块了。

除非你有很特殊的理由，原则上在 Windows 的应用中很少会使用大模式和巨模式。隐藏在这两种模式之后的速度和空间限制都会使你望而生畏。

在 Windows 下，混合模式(mixed model)是一种容易让人接受的方案。事实上，用户应该而且不得不这样做。混合模式下，程序使用中模式去写，但是在必要时可以使用长指针或巨指针。这种模式允许用户既享受到速度和紧凑代码的好处，又能在需要时访问占用内存空间大的模块。

使用混合模式的方法不难理解。但是如果你不仔细，也会遇到麻烦。下面简要举例说明 Windows 下的指针。在中模式下，下述语句将会产生一个短指针：

```
char *p;
```

而下述语句产生一个长指针：

```
char far *p;
```

下述语句则产生一个巨指针：

```
char huge *p;
```

Windows 为了保护用户以避免错误，为长指针专门定义了一个数据类型叫做 LPSTR，为巨指针定义了数据类型 HPSTR。

Windows 下的内存由局部内存和全局内存组成。所有的 Windows 应用都有一块大小为 64K 局部内存空间，用于存放程序堆栈、程序需要的静态数据和局部堆阵。所谓局部堆阵是指分配小程序模块的地方。局部堆阵中的模块可以通过近指针访问。

局部堆阵的长度不可能比局部数据段内存减去堆栈段和静态数据所剩余 的空间大。堆栈和堆阵空间的大小在 DEF 文件中进行定义。

需要注意的是函数中的局部变量实际上分配在堆栈上而堆栈又分配在局 部数据段上，所以对局部变量也可以用近指针进行寻址。

即使在最有利的情况下，本地数据段也只能给你提供几 10K 字节的内存。大的内存模块必须从全局内存中进行分配，在 Windows 环境下，全局内存是除了局部内存外的全部剩余空间。Windows 能够管理所有常驻的、扩展的、扩充的内存，使得系统中全部内存对用户来说都可以作为全局内存使用。

全局内存块依赖于它们的大小必须通过远指针或者近指针来寻址。当你在中模式下定义一个近指针时，该指针不具有段地址。通常来说，该指针将会用于参考局部数据段地址。这样，在 Windows 下给一个指针赋值时你一定要十分小心。例如下述语句是正确的：

```
char *nearpointer;
char far *farpointer;
farpointer=nearpointer;
```

在上例情况下，farpointer 将会获得 nearpointer 的偏移量以及局部数据段的段值。下述语句则不太正确：

```
char *nearpointer;
char far *farpointer;
nearponinter;
```

在这种情况下，指针 nearpointer 会获得 farpointer 的偏移量。但它的段值仍取公共数据段段值却不会获得 farpointer 的段值。对指针 nearpointer 的修改有可能影响到应用程序局部数据段。比如可能在修改过程中，错误地改变堆栈的内容。

你可以在远指针和巨指针间互相赋值而不出什么问题，但是你在确信你正在寻址一个大的寻址模块时才有必要使用巨指针。

在你使用 C 语言中的类型强制转换符时一定要格外注意指针模式，由于类型强制转换符就是告诉 C 语言不必检查指针的类型，所以你必须对这样做的结果负责。下述例子可以说明你很容易在不小心时制造麻烦：

```
GLOBALHANDLE objecthandle;
OBJECT far *objectpointer;
objecthandle=GlobalAlloc(GMEM_MOVEABLE,100*(long)sizeof(BOJECT));
objectpointer=(OBJECT *) GlobalLock(objecthandle);
```

在本例中，OBJECT 是一种数据类型，objecthandle 用做为 100 个数据分配的缓冲区的参考量。缓冲区分配完后，你必须锁定它以便访问内存。根据 C 语言的要求，Globalhandlef 返回的指针要强制转换成类型 OBJECTY。问题是指针本身定义成了远程指针，但强制转换却是近程的。解决问题的方法是改成这样：

```
objectpointer =(object far * )Globallock(objecthandle);
```

还有一个例子更加能够说明混合模式中潜在的问题，这个问题一直困扰 Windows 的软件编程人员。下面是函数 Wsprintf 的一个调用，大模式下标准调用是这样实现的：

```
char b[64];
Istrcpy(s,"dogs of war * );
wsprintf(b, * Cry 'Havoc!'and let slip the %s.",s);
```

上述语句表面上看来应该成为一句话，但实际上行不通。正确的方法应是：

```
wsprintf(b,"Cry 'Havoc!' and let slip the %s.",(LPSTR)s);
```

注意上述语句中的变量 s 被强制转换成 LPSTR 类型。这似乎看起来无关紧要，因为是从一个指针变成另外一个指针。但实际上却是非常必要的。

wsprintf 函数的特别之处在于它的前两个自变量被定义为 prototype。但是其它的自变量可以是你所需要的任何类型，不存在 prototype 定义问题。例如，第一个自变量被定义成 LPSTR 类型，即一个远指针，编译器知道这样对待它。自变量 b 在这种情况下作为近指针处理，因为它存在于公共数据段中。函数 Wsprintf 中由 prototype 类型产生的强制转换在上例中能够进行正确处理。

第三个和以后的自变量不能是 prototype，既然它们可以是你想打印的任何变量类型。当然，你应该确信由格式限制串传递给 wsprintf 函数的自变量的类型与实际自变量的类型应该相符。你同样要记住 wsprintf 中要用远指针指向变量。局部变量在堆栈中分配，在小模式和中模式的 Windows 应用中，它们总是驻留在公共数据段中。这样，在小模式或中模式下指向这些模块的指针应该是近指针，除非你显式地强制说明。这些规定使得函数 wsprintf 显得有些混乱。如果你在使用这个函数时遇到问题，请按上述说明进行检查。

在你的应用程序采用混合模式指针时还需考虑一个问题。这个问题如果考虑不到，你仿佛走进了沼泽地而找不到走出去的办法。

对 windows 编程人员来说，有两种库函数(intrinsic function)的来源。现在我们假定，用户要在 Windows 操作系统下使用 Borland 编译器进行程序开发。在其它系统下库函数的名字可能因为环境关系而有所不同。这两种来源是 Windows 和 C 编译器。

Windows 为其环境下所做的应用开发提供了各种各样的调用。这些调用用于管理窗口、打开对话盒、制造声音、删除微粒(delete atoms)等等人们从前不敢想象的事情。它们都具有文件类型 Windows 以及其它 Windows 头文件。当你向这些函数传递指针时，指针的原型规定指针应是长或巨型的，这也正是 Windows 函数所希望的。你能毫无问题地向 Windows 调用传递指针。但是要注意 wsprintf 中的特殊情况。

另外一个函数资源是你的编译器提供的库函数。Borland C 语言的库函数为你提供了很多 windows 不具备的功能。它具有很多串和内存操作函数、一些好的分类函数和数学运算函数等等。它们中的大多数能够在 Windows 下直接使用，但是有些函数在使用时如果你不小心可能会遇上指针的问题。

下面举例说明这类问题。例如函数 memset 将会将一块缓冲区填写一个固定值，下面是一个具体语句：

```
char b[128];
```

```
memset[b,0,128];
```

上述语句将把 128 个字节的数组 b 全部填充成 0。而下述语句有可能破坏你的堆栈或产生其它致命错误：

```
GLOBALHANDLE h;
LPSTR p;
h=GlobalAlloc(GMEM_MOVEABLE,128L);
p=GlobalLock(h);
memset(p,0,128);
```

理论上上述语句是正确的。但实际上如果你的程序在小模式或中模式下编译，上述语句却是不能正确执行的。当你编译你的程序而你的程序又需要调用编译器库函数（在本例中为 memset）时，连接器会选择适合于程序模式的库。比如对一个中模式程序来说，编译器会连接中模式库。

memset 在中模式下应用时要求第一个自变量（指针）必须是近指针。然而在上例中，传递给它的指针类型为 far。memset 函数将会取传递来的指针的偏移量作为 near 指针并且把局部数据段值作为参考段值。事实上这是行不通的。因为它已经被分配为全局变量。上述语句执行的结果显然是错误地把某处 128 个字节置为 0。

这个问题是无法解决的。在 Windows 的小模式和中模式程序应用中，库函数在与被长或巨指针寻址的模块一起使用时是会出现错误的。在本例中，你只能自己编写一个 memset 函数来处理 LPSTR 指针。

上述例子并不意味着你的编译器库中的所有函数都毫无用处，只是意味着不能和 far 或 huge 指针一起使用。大多数情况下这并非一个难题，只是需要加倍小心。

还有需要注意的一点是：与 DOS 环境下不同，除非你特别指定 Windows 运行在实模式或者主机 CPU 为 80286 并且内存不大于 1M 字节，Windows 软件都会运行在“保护模式”。

“保护模式”一词的由来是它能保护驻留内存的程序以免受到其它程序的入侵。也就是说，当程序运行在保护模式时，它拥有属于自己的内存。如果程序试图访问不属于自己的那部分内存，微处理器通过发出一个保护模式错误提示来防止此类动作的发生。在 Windows 下，这类动作将会调用一个保护模式错误对话框并且终止入侵程序。

保护模式下的内存地址有特殊之处。保护模式下指针的段值是对处理器地址选择表的索引。那么你的应用程序不拥有的内存指针代表了无效的对选择表的索引。无论是读内存还是写内存，在这点上都是一样的。

如果你正在一台运行在 Windows 实模式下的微机上开发程序，你要始终记住：实模式下的 Windows 很象是 DOS，你的程序不会被禁止访问程序不拥有的内存。除非你写入不应该写的内存区并破坏了其它应用程序或者 Windows 系统本身，这种做法没有害处。然而，大多数 Windows 用户运行在保护模式，目的是抛掉实模式下不细致的指针管理可能带来的麻烦。

1.5　编译器和头文件

C 语言的一个很大的优点就是它的可移植性。在一种 C 编译器下编写的 C 程序在另一个编译器上可以成功地编译。然而在 DOS 下的编译器的标准令人害怕。

在 Windows 下情况就好得多。Windows 的固有函数的调用方法在很大程序上确定了 Windows 语言的行为。这样在 Windows 下的开发程序用 Borland C 来实现并且还在 Microsoft 语言下进行编译是完全现实的。

本书中列举的程序都可以在编译器之间移植。然而也要注意几点。第一是 Borland 的控制库(也就是 BWCC. DLL 文件) 的使用。BWCC. DLL 库提供给程序一种特殊的三维控制形式，如图 1－3 所示。

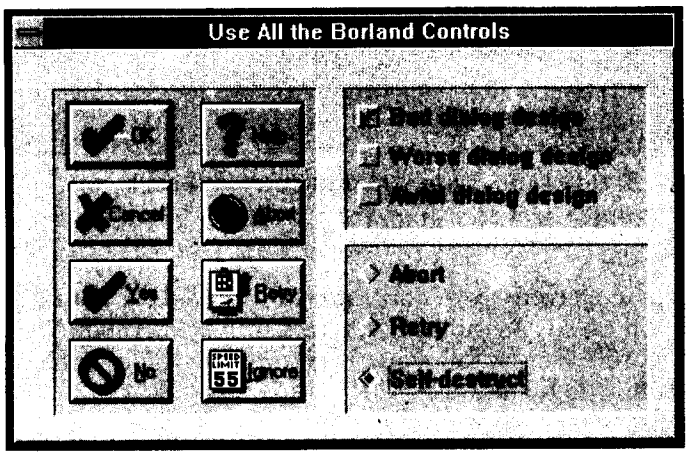

图 1－3 BWCC DLL 产生的三维控制

实际上，Microsoft 公司最近发布了自己的定制控制库。除了 Borland 控制中有些有点过于花哨外——当然这取决于你个人的看法——Borland 库很方便地与 Borland Resource Workshop 结合在一起，结果是复杂的大的会话生成变得容易。

如果你要使用 Borland 语言编写本书的代码，你要做的事就是确认 BWCC. LIB 文件存放在你的硬盘中的 WINDOWS\SYSTEM 目录下。它应该存在，因为在你建立 Borland Windows 开发环境时就生成了这个文件。

Borland 定制控制库只有在本书应用举例时作为点缀出现。例如图 1－4 中的窗口是下章要讲述的使用波形文件。围着清单框的落影(dropshadow)就是由 Borland 定制控制来管理的。

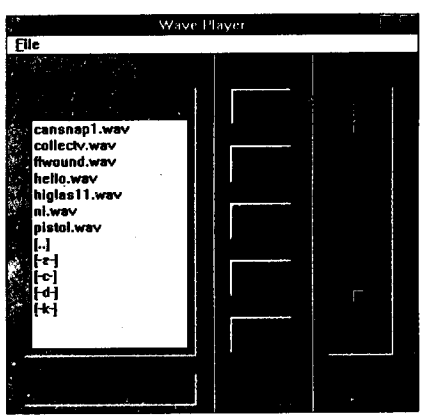

图 1－4 波形文件的使用

显然如果把影子去掉，显示的内容将更清楚。如果你在使用一个 Microsoft C 编译器，你要

做的全部事情就是从本书中把所有举例用程序中的第一行(下面这一行)删去：

```
#include<bwcc.h>
```

你同样需要从举例用程序的 WINMAIN 函数中删除下面这一行：

```
BWCCGet Version();
```

对 BWCCGetVersion()的调用用于在使用该函数的应用程序第一次引导时，确认 BWCC.DLL 文件的存在。最后，你可以在本书应用举例中从不同 RC 资源原本中删除 Borland 控制。

本书中的程序都是用标准 ANSI C 完成的，这也是 Windows 操作系统下较为便利的语言。它使用目前流行的 PASCAL 说明类型的方式。尽管 Windows 不太关心类型说明的方式，但这种方式使得原型的生成比较容易。大多数 Windows C 语言编译器对使用原型很感兴趣，所以在一般简单情况下，函数用以下格式进行说明：

```
void AbortOnTuesdays(p,n)
  char *p;
  int n;
{
  /* some code goes here */
}
```

现在比较时髦的写法是：

```
void AbortOnTuesdays(char *p,int n)
{
  /* some code goes here */
}
```

上述两种写法在编译器编译结束后产生的代码没有什么不同，你可以任意选择一种你感觉舒服的写法。

本书中的程序都提供函数包含的原型，尽管这会让很长时期以来一直从事 DOS 编程的人员觉得可笑。原型是一种保护你自己的方法。

原型强调 C 语言的一种基本假设：任何变量或函数一般来说都应该是整型的。C 还假定如果前面的假设不能成立，那么整型与其它类型的内部转换即使不允许也不会对其它类型产生严重危害。然而在 Windows 下，这两种假定都会产生严重后果。下面是一条合法但不严格的 C 说明语句：

```
AbortOnTuesdays(n)
{
  /* some code goes here */
}
```

你会注意到这个函数与以前两个函数的编程风格不一致。这条语句合法是因为除非特殊说明，C 总是假定函数或变量都是整型的。在上例中，C 假定 n 是整型并且函数将返回一个整型值。如果你用下述语句调用函数 AbortOnTuesday，结果将毫无用处：

```
AbortOntuesday("It's Wednesday");
```

在此例中，函数 AbortOnTuesday 在应该获得整型值的位置被传递了一个指向字符串的

指针。在传统 C 语言中，编译器能够接受这种做法，因为只有 AbortOnTuesday 知道它的参数应该是什么。

原型是指函数中数据和变量类型的说明，以及对函数返回值类型的说明。函数原型的说明一般放在应用程序文件的顶部或者头文件里。上述函数的原型应写成这样：

```
Void AbortOnTuesday(int n);
```

原型告诉 C 编译器，函数 AbortOnTuesday 的参量 n 应该为整型值并且调用该函数时不能使用返回值(因为该函数已经定义成了 void)。

在写 WINDOWS 应用程序时，函数原型同样能够消除前面章节中讨论的指针类的问题。例如，考虑函数 AbortOnTuesday 的下述变形：

```
void AbortOnTuesdays(char far *p)
{
  /* some goes here */
}
```

这便是 Windows 中函数接收串指针作为自变量的一种正常做法，你可以用 char far * 代替 LPSTR。如果函数没有认定为原型而你用下述方法调用的话，你就会遇到麻烦：

```
AbortOnTuesdays("It's Wednesday");
```

这是因为，在上例中字符串"It's wendsday"将在局部数据段中作为静态数据存储起来。这样就由近指针寻址，传送给 AbortOnTuesday 的也是近指针。而上述语句执行的结果就会导致错误的数据段：AbortOnTuesday 函数将根据栈上数据随意地生成一个段值——可怕的事情就会发生。

你需要清晰地强制转换指针，就像下面一样：

```
AbortOnTuesdays(char far *)"It's Wednesday");
```

或者：

```
AbortOnTuesdays((LPSTR)"It's Wednesday");
```

如果在适当的位置上加上原形，Windows 将会执行类型转换。如果你实在不喜欢原型，你可以选择把编译器的原型检查关闭，那样你的编译器就不会因为函数不具有原型而报错。然而在 Windows 的复杂环境下，这样做无异于与狼共舞。

前面我曾经提到过本书中的程序都用 ANSI C 编写，这和 C++不同。正如名字本身，C++是 C 的高级系列(superset)。有些你能够编译本书中程序的 C 编译器实际上是 C++编译器。你只需忽略 C++的扩展名就可以在 C++编译器下使用 C 语言代码。

如果你喜欢在 C++下工作，你可以在这里举出的应用例子中增加 C++捕获(trapping)，或者把你从以后章节中得到的 C 语言函数与你的 C++程序混合使用。这种做法没有障碍，因为 C++实际上不允许你做 C 语言不能做的事情。它只是允许你用不同的方式编写程序，这种方式使得程序员感到更方便。

最后需要对本书中程序进行说明的是，你会发现从前没有使用过的两个"宏"。举例如下：

```
#define say(s) MessageBox(NULL,s."Yo...",MB_OK | MB_ICONSTOP);
#define saynumber(f,s) {char b[128]; \
  sprintf((LPSTR)b,(LPSTR)f,s); \
  MessageBox(NULL,b,"Debug Message",MB_OK | MB_ICONSTOP),}
```

调试工具在不使用时不会扩展到任何其它程序，在等待时也不会带来任何麻烦。宏 say 将会产生一个信息块，包含它所接到的作为变量的任何文本。变量 saynumber 将会显示一个 printf 格式串和一个数字——实际上，只要格式串一致第二个变量可以是你喜欢的任何东西。你可以显示如下值：

```
saynumber("The value is %d",value);
```

宏 say 和 saynumber 使用起来很方便。当你试图查找程序中问题所在时，你可以用它们来检查程序中某个地方的值或变量，当然你也可以使用 Windows 的 debugger，但是，相比之下上述宏却是既快又简单。

1.6　多媒体的使用

多媒体最吸引人的地方可能就在于很多人不知道多媒体究竟是什么东西。不像应用程序的大多数领域，多媒体就像砌墙的石头而并非墙本身。正如你可以按照自己的想法去砌墙一样，你也可以按照自己的想法去配置多媒体。

本书中将为你提供一些多媒体简单应用的源程序。这些程序包括在 Windows 下使用波形文件，显示影碟图象等等。事实上，你不可能自己写这类程序。除了疯狂的软件收藏者或者抱着一定目的使用计算机的人们，一般没有必要调用制造声音或显示照片的程序。

多媒体的真正优势在于把这些因素结合起来以实现复杂的应用。例如作为既能讲解内容又能显示画面的书，具有立体声音响和图形的交互式计算机游戏，具有图形和 CD 唱片音质的大的 CD—ROM 信息库。而你的灵感将会是多媒体应用发展的源泉。

Windows 作为操作环境，已被证明比 DOS 要流行的多。原因之一就是操作者感到是在和人而不是和机器相互作用，起码它是用人类很熟悉的比喻来表达事物的。Windows 的扩展功能能使你开发的应用程序在和用户对话时使用户感到舒服和容易接受。

在个人计算机历史上，你可能第一次用人类语言来替换如下的 C 语言程序：

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