多媒体 高级编程技术

胡建宝　陈世杰　编

电子科技大学出版社

多媒体高级编程技术

胡建宝　陈世杰　编

电子科技大学出版社

• 1996 •

[川]新登字 016 号

内 容 简 介

本书详细介绍了如何在 Windows 下开发多媒体应用程序，内容包括：多媒体 PC 机的软硬件环境、WAVE 文件演播器的实现、MIDI 演播器的实现、影片演播器的实现，CD 唱机的模拟以及 mmreg.h 文件等。

本书是开发 Windows 多媒体应用程序的必备参考书，也是学习 Borland C++ OWL 编程的宝贵资料，可供广大的多媒体计算机用户、多媒体应用技术人员和大专院校师生阅读参考。

多媒体高级编程技术

胡建宝 陈世杰 编

*

电子科技大学出版社出版

（成都建设北路二段四号） 邮编 610054

四川省自然资源研究所印刷厂印刷

新华书店经销

*

开本 787×1092 1/16 印张 12 字数 289 千字

版次 1996 年 1 月第一版 印次 1996 年 1 月第一次印刷

印数 1—6000 册

ISBN 7-81043-286-9/TP・1

定价：12.80 元

前　言

多媒体技术是目前计算机领域内最活跃部分。多媒体计算机把图像处理技术、声音处理技术、视频处理技术集成到计算机中，交互地处理多种媒体信息，如文字、声音、图形、图像等。多媒体计算机技术的引入，完全改善了人机界面，使人与计算机倾向于更自然的方式进行信息交流，拓宽了计算机的应用领域。

在多媒体 PC 机上开发 Windows 多媒体应用程序令每个软件技术人员着迷。但是，至今国内还没有一本全面介绍如何在 Windows 下开发多媒体应用程序的技术书籍。本书从多媒体应用程序开发的角度出发，全面、详尽地介绍了 Windows 下多媒体应用程序的开发技术。

全书共分为六章，第一章概述多媒体计算机的概念，多媒体计算机的系统构成，多媒体软件开发平台，以及多媒体文件格式标准——RIFF；第二章全面介绍了 WAVE 文件的结构，播放 WAVE 文件的四种方式，WAVE 文件演播器的实现；第三章介绍了如何将多媒体计算机变为一台 CD 唱机；第四章着重介绍了 MIDI 文件的结构，MIDI 演播器的实现，最后给出一个完整的 MIDI 播放程序实例；第五章围绕 AVI 演播器的实现，详细描述了 AVI 文件的结构以及 AVI 文件的播放方法，最后给出一个完整的实例；第六章介绍了 mmreg.h 文件。书中所有的示例程序都是在 Borland C++ 3.1 下面用 OWL 开发的，全都编译调试通过。

全书由胡建宝、陈世杰共同审阅，陈世杰完成了第二章、第三章、第六章，胡建宝完成了第一章、第四章、第五章。在编写过程中，作者参考了国内外有关多媒体技术的书刊和资料，希望本书能为从事多媒体应用程序开发的技术人员和大学生提供有益的参考。

由于作者水平有限，加上时间仓促，书中难免有错误和缺点，欢迎广大读者提出宝贵意见，给予批评指正。

在本书编写过程中，得到了孙国良老师、吴濯才老师、门羽川同志以及电子科技大学出版社的张琴老师、成都科技大学的刘炜、徐刚、王培强等人的大力支持和帮助，在此对他们表示衷心的感谢。

陈世杰在此要特别感谢余兰小姐。

我们对支持和关心本书出版的所有朋友都表示由衷的感谢！

编　者

1995 年 5 月

目　录

第一章　多媒体计算机

进入 90 年代，多媒体(Multimedia)同 ATM、OOP 一样成为计算机领域出现的最频繁的词语。什么是多媒体呢？通常我们把媒体看作存储信息的物理介质，而多媒体中媒体的确切含义是指信息的载体，如文字、声音、图像等。

从计算机发展初期到现在，人们一直主要以文字作为计算机输入和输出载体。人们使用字符编写源程序，输入到计算机，计算机处理的结果也主要以字符的形式输出。然而，单一的输入输出媒体形式妨碍了人与计算机之间的信息交流。80 年代后期，随着微电子技术和光盘技术的发展，人们开始致力于研究将声音、图像、文字也作为输入输出载体的计算机，这就是多媒体计算机。

多媒体计算机把图像处理技术、声音处理技术、视频处理技术集成到计算机中，交互地处理多种媒体信息，如文字、声音、图形、图像等。多媒体计算机技术的引入，完全改善了人机界面，使人与计算机倾向于更自然的方式进行信息交流。多媒体计算机技术拓宽了计算机应用的领域，扩展了计算机处理的对象，必将对计算机的体系结构和人类的生活方式产生深远的影响。

第一节　多媒体计算机的应用

多媒体技术为计算机的应用注入了新的活力，使计算机除了应用在文字处理、科学计算、信息管理等领域外，还进一步应用到家庭娱乐、教育和培训、商业和艺术等前所未有的领域。

1. 家庭娱乐

一台多媒体 PC 机将家庭中的电视机、录像机、音响设备连结在一起，除了充当文字处理、家政管理等传统角色外，还可以充当 CD 唱机，使你欣赏到高保真的音乐。它也可以充当一台交互式电视机，使你欣赏到精彩的节目。多媒体计算机游戏由于具有人物逼真、音响效果好、立体感强等优点，已越来越多的进入家庭。

2. 教育和培训

除了采用面对面的教学方式外，人们还采用电化教学、电视教学等手段进行教育和培训。在现代社会中起重要作用的电视教育，由于缺乏交互性而受到局限。多媒体辅助教学系统因为其形象性和交互性，使接受教育和培训的人乐意选择感兴趣的专题，集中注意力，完成学习。多媒体的教育和培训系统提高了培训效率，节约了培训时间。

3. 动画和广告制作

多媒体计算机相当于一个台式的电影制片厂，制作者利用著作工具，就可以高效率的制作动画节目和广告节目，从而节约大量的人力和物力。

多媒体的应用范围还将越来越广，必将深入到社会生活的各个方面。

第二节　专用的多媒体计算机系统

目前，国外一些大公司都在积极开发多媒体技术，并相继推出了一些高性能的专用多媒体系统，其中典型的系统有：

1. Commodore 公司的 Amiga 系统

Commodore 公司在 1985 年率先推出了世界上第一个多媒体计算机系统 Amiga。Amiga 系统是在 MC 68000 系列微机上，加上三个专用芯片 Agnus(8370)、Paula(8364)和 Denise (8362)构成。

Agnus 是专用的动画制作芯片，它的主要功能是：用硬件显示移动数据，允许高速的动画制作；控制多个 DMA，使 CPU 以最小的开销处理声音和视频信息；为视频 RAM 和扩展 RAM 提供所需要的控制信号。

Paula 是专用的音响处理及外设接口芯片，它含有音响处理器、盘控制器等。音响处理器通过 DMA 方式和 Amiga 系统的存储器或其它音响设备交换音响信息，在 Paula 的音响处理器中处理音响信息，最后经过 D/A 变换，输出到音响设备中。盘控制器也通过 DMA 方式将 Amiga 系统中存储器的数据输出到盘上或把盘上数据读入到 Amiga 的存储器中。

Denise 是一个多功能的彩色图形控制器，它可以控制不同分辨率的输出以及在电视机和 RGB 彩色监视器的屏幕上输出 4096 种颜色等。

为了适应不同用户对多媒体技术的需要，Commodore 公司提供了一个多任务 Amiga 操作系统以及大量的应用软件，如动画制作软件等。

2. Philips/Sony 公司的 CD-I 系统

Philips/Sony 于 1986 年公布了基本的 CD-I 系统，其结构如图 1-1 所示。

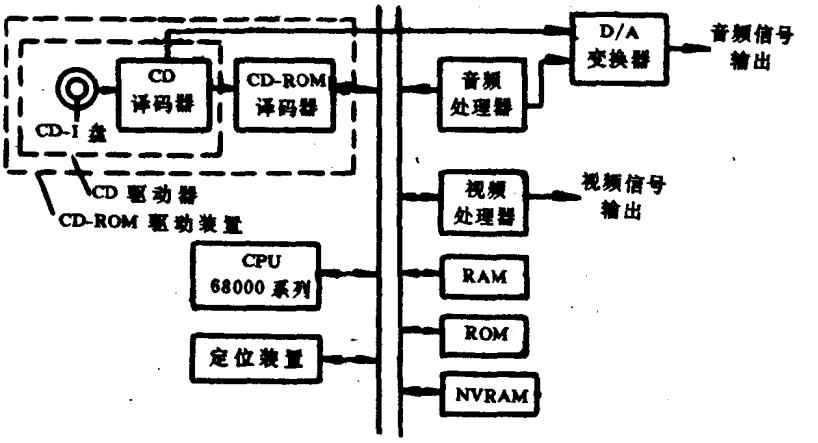

图 1-1

整个系统分为两个部分：CD-ROM 驱动装置和多媒体控制器 MMC。CD-ROM 驱动装置包括 CD-ROM 驱动器和 CD-ROM 译码器，通过 CD-ROM 译码器连接到系统总线中。MMC 是

由音频信号处理器、视频信号处理器、68000 微处理器、ROM、RAM 以及定位装置组成。

从图上可以看出，CD-I 有两种工作方式。一种是不需要计算机，CD-I 系统与电视机、录像机、音响设备连接到一起，在光盘实时操作系统的管理控制下，组成一个简单的多媒体系统。另一种工作方式是把 CD-I 系统作为多媒体控制器连接到计算机上。

为了提高 CD-I 基本系统的性能，Motorola 公司为 CD-I 设计了专用芯片，Sony 公司扩充了 CD-I 的硬件，增强功能后的 CD-I 系统如图 1-2。从图 1-2 可以看出，由于采用 DSP 处理音频信息和采用专用芯片（视频系统控制器、视频合成器、全运动视频信号控制器等）处理视频信号，使得增强型 CD-I 系统在全屏幕运动视频及音响处理方面比 CD-I 基本系统有较大的改进。

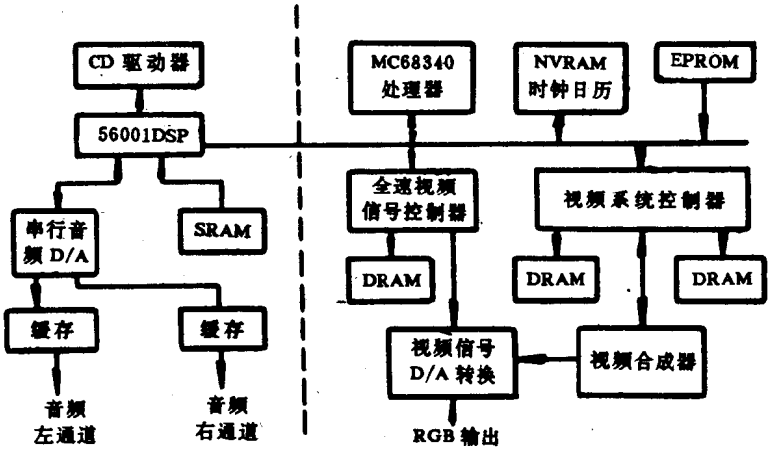

图 1-2

3. Intel 和 IBM 的 DVI 系统

1989 年，Intel 公司和 IBM 公司推出了 DVI（Digital Video Interactive，数字视频交互式系统），其结构如图 1-3 所示，系统的核心是三块专用的 DVI 接口板：DVI 视频板、DVI 音频板和 DVI 多功能板。

以 386/486 或其兼容机作为工作平台，加上视频板、音频板、CD-ROM 驱动器以及带放大器和音响效果的 RGB 彩色监视器，就组成了 DVI 用户系统。在此基础上，再配置与多媒体有关的外设如摄像机、扫描仪等，就可以组成 DVI 的开发系统。

DVI 的软件核心是 AVSS（Audio Video Subsystem，声音视频系统）和 AVK（Audio Video Kernel，音像核心）。它们的主要功能为音频和视频数据流提供需要的实时任务调度、实时数据的压缩和还原以及管理音频数据和视频数据等。

DVI 系统在技术上受到 Intel 公司和众多独立软件开发商的支持，很可能成为未来多媒体系统的主流。

典型的专用多媒体系统还有 Apple 公司的 Hypercard 系统等。这些多媒体系统的主要特点是以多媒体为目标进行系统的硬件和软件设计，在音响、动画和图像等方面具有很强的处理能力，但价格昂贵。

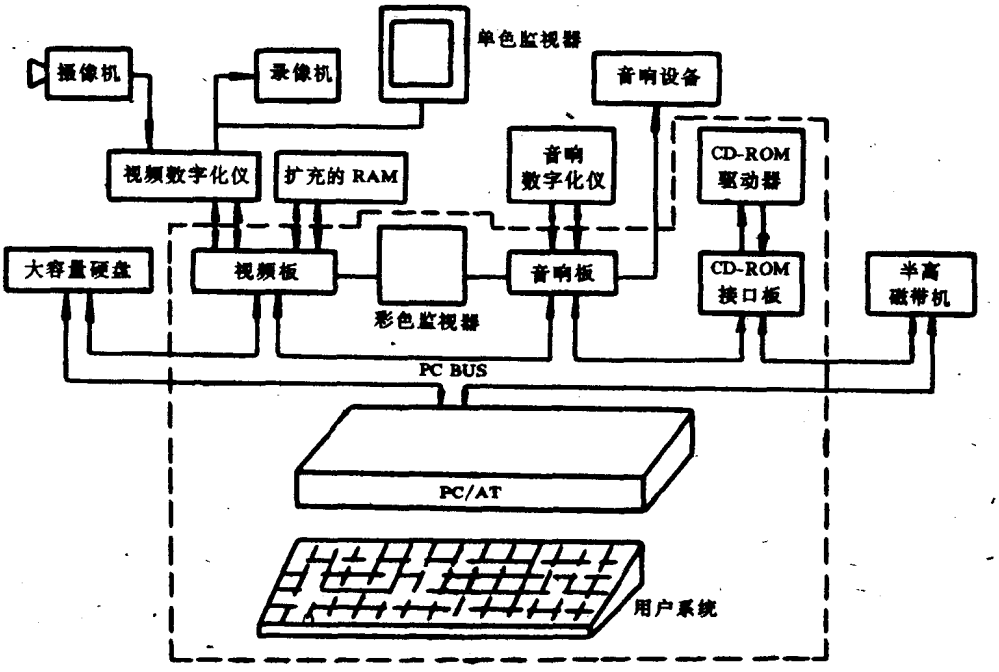

图 1-3

第三节　多媒体 PC 机

专用的多媒体系统如 DVI 系统和 CD I 系统毕竟离我们远了点，组成多媒体系统最简单的方式莫过于将 PC 机升级为多媒体 PC 机(MPC)。声卡、视卡和 CD-ROM 与 PC 机就组成了基本的多媒体 PC 机系统。

一、MPC 的硬件环境

MPC 需要声卡和视卡的支持去处理声音和图像。MPC 利用声卡完成单声道/立体声声音的录制、播放以及实时声音数据的压缩和还原、声音的合成，利用视卡接受摄像机以及电视信号的输入、实时图像数据的压缩和还原、图像播放和图像处理等。伴随图像和声音而来的是大批量数据，CD-ROM 是目前能找到的最理想的多媒体系统的外存储设备(参看第三章)。

Creative 公司提供了我们需要的声卡和视卡。

Sound Blaster 是 Creative 公司开发的多功能音频接口卡，其主要功能有：

1. 16 位单声道/立体声采样和播放。

2. 具有乐器数字接口 MIDI(Musical Instrument Digital Interface)，可用于驱动外部 MIDI 乐器。

3. 具有 CD-ROM 接口。

4. 允许多路输入的可编程立体声响混合器。

5. 含有卡上音量调节按钮。

6. 每声道 4W 的立体声放大器。

Video Blaster 也是 Creative 公司出品的高性能视频卡，其主要功能如下：

1. 在一个可移动、可改变大小的窗口内，显示活动图像。

2. 可播放、存储和处理来自录像机、摄像机和广播电视上的图像信号。

3. 可调节色调、亮度和对比度。

4. 可进行图像或文字的迭加。

MPC 需要综合处理文字、图像、声音等多种媒体信息，这就要求 PC 机本身具有高速的 CPU、大容量的外存和高分辨率的显示器。一般说来，推荐使用 486 以上的机型、内存大于 4M、硬盘容量大于 200M、640×480 或 1021×768 的高分辨率显示器的 PC 机。

二、MPC 的软件环境

多媒体系统要连续地同步播放声音信号和视频信号，就需要实时多任务的操作系统。MPC 主要采用 Microsoft Windows 3.1 作为软件支撑平台。Windows 支持多媒体，提供了如下服务：

1. 提供了控制多媒体设备的多媒体控制接口 MCI(Multimedia Controler Interface)。

MCI 设备控制程序，用于演奏和录制 WAVE(波形)文件，播放 MIDI 音乐等。可使用两种方式(命令串方式或消息方式)与 MCI 设备驱动程序通信。

2. 提供了多媒体服务的低层 API(Application Programming Interface，应用程序编程接口)支持。

提供了低层影片演播函数，使用波形和 MIDI 设备播放和录制音频数据的低层 API 支持，精确计时器的低层支持，操纵杆的低层支持。

3. 提供了多媒体文件的 I/O 函数，支持标准的 RIFF(Resource Interchange File Format，资源交换文件格式)格式文件。

支持缓冲型文件 I/O，支持非缓冲型文件 I/O。

4. 为多媒体应用程序提供了设备驱动程序。

增强了高分辨率的显示驱动程序的显示性能，增加了低分辨率的 VGA 视频显示驱动程序的 256 色、320×200 的显示方式。

5. 提供了标准的 MIDI 映射器，支持标准 MIDI 合成音色服务。

6. 支持创建屏幕保护程序。

Windows 提供了完善地管理和控制多媒体设备的机制，软件人员乐意选用 C/C++或 VB(Visual BASIC)在 Windows 下开发多媒体应用程序。目前，最流行的 C/C++编译环境是 MSC 7.0 和 Borland C++ 3.1。相对来说，我更乐意使用 Borland C++ 3.1，因为它有更完美的集成环境，使用 Borland C++的 OWL(Object Windows Lib)更容易编写 Windows 应用程序。

OWL 是一个面向对象的库，它使用了 C++面向对象的技术。Object Windows 具有三个新的特点：(1)窗口消息的封装；(2)消息的自动响应；(3)许多 Windows API 函数的抽象。

对于 Windows 的窗口、对话框等界面元素，Object Windows 都提供了已定义好的对象(统称界面对象)。Object Windows 封装了界面对象的行为、属性和数据。当使用 OWL 编程时，Windows 自动地调用界面对象预定义的函数去创建界面元素。

用 C 编写传统的 Windows 应用程序，要由应用程序自己组织消息循环，同时使用 Switch 语句处理各种消息，因而代码冗长。使用 OWL 的应用程序，将自动地进行消息循环和消息响应，代码简洁，便于重构。

Windows API 是由几百个函数组成的，Windows 应用程序需要调用这些函数。Object Windows 抽象了许多 API 函数，把这些函数连同部分参数都封装在界面对象中，这样就提供了一组简化的成员函数，便于控制界面元素。除此之外，Object Windows 还把相关的函数调用组合到一个成员函数中，完成更复杂的功能。而且，基于 OWL 的应用程序也可以直接使用 Windows API 函数。

本书中所有的程序都是使用 OWL 开发的，下面通过一个简单的例子 Sample.cpp，来介绍 Object Windows 的应用程序设计的过程，以便于使用 C 的软件人员读懂本书中的源代码。如果你已使用过 Object Windows，可以跳过此一部分，阅读第四节。

每一个 Object Windows 应用程序都必须定义一个派生于 TApplication 类的应用程序类。这一派生类封装了应用程序的行为，包括主窗口的创建和显示，应用程序的消息处理，应用程序的关闭等。

在本例中，TApplication 的派生类如下：

```
class TMyApp : public TApplication
{
  public:
  TMyApp(LPSTR AName, HINSTANCE hInstance,
                    HINSTANCE hPrevInstance,LPSTR lpCmdLine, int nCmdShow)
  : TApplication(AName, hInstance, hPrevInstance, lpCmdLine, nCmdShow) {};
  virtual void InitMainWindow();
};
```

对于 TApplication 的派生类，必须重定义其成员函数 InitMainWindow，这个成员函数完成应用程序的主窗口的构造，并把指向主窗口对象的指针存放在应用程序类的数据成员 MainWindow 中。本例中，InitMainWindow 成员函数的定义如下：

```
void TMyApp::InitMainWindow()
{
    MainWindow = new TMyWindow(NULL, Name);
}
```

Sample.cpp 把 TWindow 的派生类 TMyWindow 作为应用程序的主窗口，TMyWindow 类的定义如下：

```
class TMyWindow : public TWindow
{
  public:
  TMyWindow(PTWindowsObject AParent, LPSTR ATitle)
```

```
        : TWindow(AParent, ATitle) {};
    virtual BOOL CanClose();
    virtual void WButtonDown(RTMessage Msg)
            = [WM_FIRST + WM_LBUTTONDOWN];
    virtual void WMRButtonDown(RTMessage Msg)
            = [WM_FIRST + WM_RBUTTONDOWN];
};
```

Object Windows 应用程序如果想响应 Windows 的消息，就必须在相应的界面对象中定义消息响应成员函数。这种函数的定义采用 Object Windows 的约定，形式如下：

(1) 响应标准的 Windows 消息，如窗口管理消息、鼠标器等。

```
virtual void MsgProc(RTMessage)=[WM_FIRST+消息索引号];
```

(2) 响应菜单和加速键产生的消息。

```
virtual void MsgProc(RTMessage)=[CM_FIRST+消息索引号];
```

(3) 响应 OWL 中定义的控制对象(按钮、列表框、滚动条等)发出的消息。

```
virtual void MsgProc(RTMessage)=[NF_FIRST+消息索引号];
```

(4) 响应子窗口消息。

```
virtual void MsgProc(RTMessage)=[ID_FIRST+子窗口的 ID 值];
```

TMyWindow 重载了响应鼠标器左键和右键按下的消息响应成员函数以及成员函数 CanClose。当用户按下鼠标器左键或右键时，显示一个消息框，指示鼠标器左键或右键已被按下。当用户试图退出应用程序时，CanClose 将显示一个带 YES 和 NO 按钮的消息框，要求用户确认是否退出应用程序。

同 Windows 应用程序一样，所有的 Object Windows 应用程序都包含一个 WinMain 函数作为应用程序的入口点。本例中 WinMain 的定义如下：

```
int PASCAL WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                   LPSTR lpCmdLine, int nCmdShow)
{
    //生成应用程序对象。
    TMyApp MyApp("Sample ObjectWindows Program", hInstance, hPrevInstance,
                 lpCmdLine, nCmdShow);
    //处理消息循环，直到应用程序结束。
    MyApp.Run();
    return MyApp.Status;
}
```

大多数 Object Window 应用程序的 WinMain 同本例中的 WinMain 一样，包括类似的三条语句。

Sample.cpp 虽然简单，但确代表了大多数 Object Windows 应用程序的基本结构。要想进一步了解 OWL，可以参看 Borland C++技术丛书——《Windows 环境下面向对象程序设计》。最后，给出了 Sample.cpp 完整代码。

```
//模块定义文件 Sample.def。
```

```
    EXETYPE WINDOWS
    CODE PRELOAD MOVEABLE DISCARDABLE
    DATA PRELOAD MOVEABLE MULTIPLE
    HEAPSIZE 4096
    STACKSIZE 5120

//工程文件 Sample.prj。
    Sample.cpp
    Sample.def

//源程序 Sample.cpp。
// Sample.cpp - (C) Copyright 1994。
#include <owl.h>

class TMyApp : public TApplication
{
public:
  TMyApp(LPSTR AName, HINSTANCE hInstance, HINSTANCE hPrevInstance,
         LPSTR lpCmdLine, int nCmdShow)
         : TApplication(AName, hInstance, hPrevInstance, lpCmdLine, nCmdShow) {};
  virtual void InitMainWindow();
};

_CLASSDEF(TMyWindow)
class TMyWindow : public TWindow
{
public:
  TMyWindow(PTWindowsObject AParent, LPSTR ATitle)
    : TWindow(AParent, ATitle) {};
  virtual BOOL CanClose();
  virtual void WMLButtonDown(RTMessage Msg)
             = [WM_FIRST + WM_LBUTTONDOWN];
  virtual void WMRButtonDown(RTMessage Msg)
             = [WM_FIRST + WM_RBUTTONDOWN];
};

BOOL TMyWindow::CanClose()
{
  return MessageBox(HWindow, "Do you want to exit?",
```

```
            "Exit Window", MB_YESNO | MB_ICONQUESTION) == IDYES;
}

void TMyWindow::WButtonDown(RTMessage)
{
  MessageBox(HWindow, "You have pressed the left mouse button",
                      "Message Dispatched", MB_OK);
}

void TMyWindow::WMRButtonDown(RTMessage)
{
  MessageBox(HWindow, "You have pressed the right mouse button",
                "Message Dispatched", MB_OK);
}

void TMyApp::InitMainWindow()
{
  MainWindow = new TMyWindow(NULL, Name);
}

int PASCAL WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                   LPSTR lpCmdLine, int nCmdShow)
{
  TMyApp MyApp("Sample ObjectWindows Program", hInstance, hPrevInstance,
                lpCmdLine, nCmdShow);
  MyApp.Run();
  return MyApp.Status;
}
```

第四节　多媒体文件的格式

声音和图像以文件的形式存放在多媒体系统中。多媒体系统存储声音和图像推荐和使用的文件格式是 IBM/Microsoft 的资源交换文件格式 RIFF(Resource Interchange File Format)。RIFF 格式是一种带标记的文件结构。RIFF 本身并不是实际的文件格式，它是一种文件结构的标准。

RIFF 文件的基本构造元素是块(chunk)，块的定义如下：

```
typedef unsigned long DWORD;
typedef unsigned char    BYTE;
typedef DWORD            FOURCC;
```

```
typedef struct{
    FOURCC          ckID;
    DWORD           ckSize;
    BYTE            ckData[ckSize];
} ck;
```

ckID 是块的标识，我们可以把它看作一个32位的量——块的 ID 值，也可以把它看作由1～4个字母或数字组成的字符序列，这一字符序列说明块的类型。如 ckID 为“INAM”，说明该块用于描述文件主题的题目。ckID 为“ICOP”，说明该块用于描述文件的版权信息。4字符中间不能有空格，但最右边可用空白字符填补。同时，它区别对待大小写字符。

chSize 是一个32位的数，它定义了数据域 ckData 的大小。这个值不包括 ckID 和 ckSize 字段的大小，也不包括 ckData 字段的末尾填充字节。

ckData 为数据块。它是由固定长度或可变长度的二进制数组成。为了改善访问速度，以及与 EA IFF 格式(一种用于电影艺术的文件格式)兼容，该块采用字对齐方式。如果块的大小为奇数个字节，ckData 的末尾填写一个字节(值为零)。

两种最基本类型的块是 RIFF 块和 LIST 块。只有这两种类型的块才能作为父块包含子块，其它类型的块只能作 RIFF 或 LIST 的子块，在 RIFF 或 LIST 的数据段ckData中存储单个表示特定信息的二进制数。

所有符合 RIFF 格式的文件的第一个块必须是 RIFF 块。在 RIFF 块中，数据段ckData的前四个字节规定了数据的格式类型。类型名称由4个字符组成，如 WAVE、AVI 等。同样，LIST 块的 ckData 的前四个字节为列表类型，由四字符编码组成。

多媒体常用的文件类型是 WAVE 文件、AVI 文件和 MIDI 文件，它们都是 RIFF 格式文件。下面我们以 WAVE 文件为例，讨论它的具体格式。

WAVE 文件是多媒体存放数字化声音信息的标准格式文件。WAVE 文件的结构如下所示。

```
RIFF{"WAVE",
        SubChunk{"fmt ",ckSize,ckData},
        SubChunk{"data",ckSize,ckData},
};
```

它有两个子块.fmt 块和 data 块。fmt 说明声音的格式信息，data 块说明声音数据本身。fmt 块的数据域 ckData 对应一个 PCMWAVEFORMAT 对象，PCMWAVEFORMAT的定义如下：

```
typedef   struct waveformat_tag {
  WORD    wFormatTag;
  WORD    nChannels;
  DWORD   nSamplesPerSec;
  DWORD   nAvgBytesPerSec;
  DWORD   nBlockAlign;
} WAVEFORMAT;
```

```
typedef   struct pcmwaveformat_tag {
  WAVEFORMAT wf;
  WORD   wBitPerSample;
} PCMWAVEFORMAT;
```

PCMWAVEFORMAT 对象就包含了 WAVE 文件的信息，如通道个数nChannels、采样频率 nSamplesPerSec 等。

chord.wav 是 Windows 下的一个 WAVE 文件，下面列出了它最前面的64个字节：

```
52 49 46 46 8E 61 00 00－57 41 56 45 66 6D 74 20   RIFF.a..WAVEfmt
10 00 00 00 01 00 01 00－22 56 00 00 22 56 00 00   ........"V.."V..
01 00 08 00 64 61 74 61－6A 61 00 00 80 80 80 80   ....dataja......
80 80 80 80 80 80 80 80－80 80 80 80 80 80 80 80   ................
```

前四个字节定义了块标识 RIFF，5～8个字节为 RIFF 块的数据段 ckData 的长度是618EH(24974)个字节，数据段的格式类型为 WAVE。fmt 块的数据段 ckData 的长度为0010H(16)个字节，对照 PCMWAVEFORMAT 的定义，可以得出：

wFormatTag＝0001H，表示波形编码方式为 PCM(目前只有这种方式)。

nChannels ＝0001H，表示为单声道。

nSamplesPerSec ＝5622H(22050)，表示采样频率为22.05kHz。

nAvgBytesPerSec＝5622H(22050)，表示播放时每秒平均输出22050个字节。

nBlockAlign＝0001H，表示采样一次所占字节数，nBlockAlign＝nChannels×wBitPerSample/8。

wBitPerSample＝0008H，表示8位采样。

知道了 WAVE 文件的格式后，就可以从 WAVE 文件中读出我们需要的信息。

Windows 支持 RIFF 文件的 I/O 操作，提供了下列文件 I/O 函数：

mmioOpen

打开文件，返回文件句柄。

mmioClose

关闭打开的文件。

mmioRead

从文件中读指定的字节数。

mmioWrite

向文件写指定的字节数。

mmioSeek

改变文件读写操作的当前位置。

mmioAscend

从一个 RIFF 文件块上升到文件的下一个块。

mmioDescend

搜寻指定的数据块。

mmioCreatChunk

在一个 RIFF 文件中创建一个新的数据块。

mmioFOURCC

将四个字符变为一个四字符编码。

mmioStringToFOURCC

将一个以 NULL 结尾的字符串变为一个四字符编码。

利用这些文件 I/O 函数，可以方便地实现 RIFF 格式文件的 I/O 操作。下面的例子说明如何使用这些文件 I/O 函数。

函数 GetWaveInfo 用于获取 WAVE 文件的信息，它将 fmt 块的 ckData 域读到一个 PCMWAVEFORMAT 对象中，读取声音数据，把声音数据变成波形图显示出来。

```
int GetWaveInfo(LPSTR path) //path 为打开的 WAVE 文件名。
{
  unsigned int n;
  HMMIO h;
  PCMWAVEFORMAT aWaveFormat;
  MMCKINFO    mmParent,mmSub;
  //打开一个 WAVE 文件。
  if((h=mmioOpen(path,NULL,MMIO_READ))==NULL) return 0;
  mmParent.fccType=mmioFOURCC("W","A","V","E");
  if(mmioDescend(h,(LPMMCINFO)&mmParent,NULL,MMIO_FINDRIFF)){
    mmioClose(h,0);
    return 0;
   }
  //寻找 fmt 数据块。
  mmSub.ckid=mmioFOURCC("f","m","t"," ");
  if(mmioDescend(h,(LPMMCINFO)&mmSub,(LPMMCINFO)&mmParent
                ,MMIO_FOUNDCHUNK)){
    mmioClose(h,0);
    return 0;
  }
  //将 fmt 块的 ckData 域读到一个 PCMWAVEFORMAT 对象中。
  n=min((unsigned int)mmSub.ckSize,sizeof(PCMWAVEFORMAT));
  if(mmioRead(h,(LPSTR)&aWaveFormat,n)!=n){
    mmioClose(h,0);
       return 0;
  }
  if(aWaveFormat.wf.wFormatTag!=WAVE_FORMAT_PCM){
    mmioClose(h,0);
```