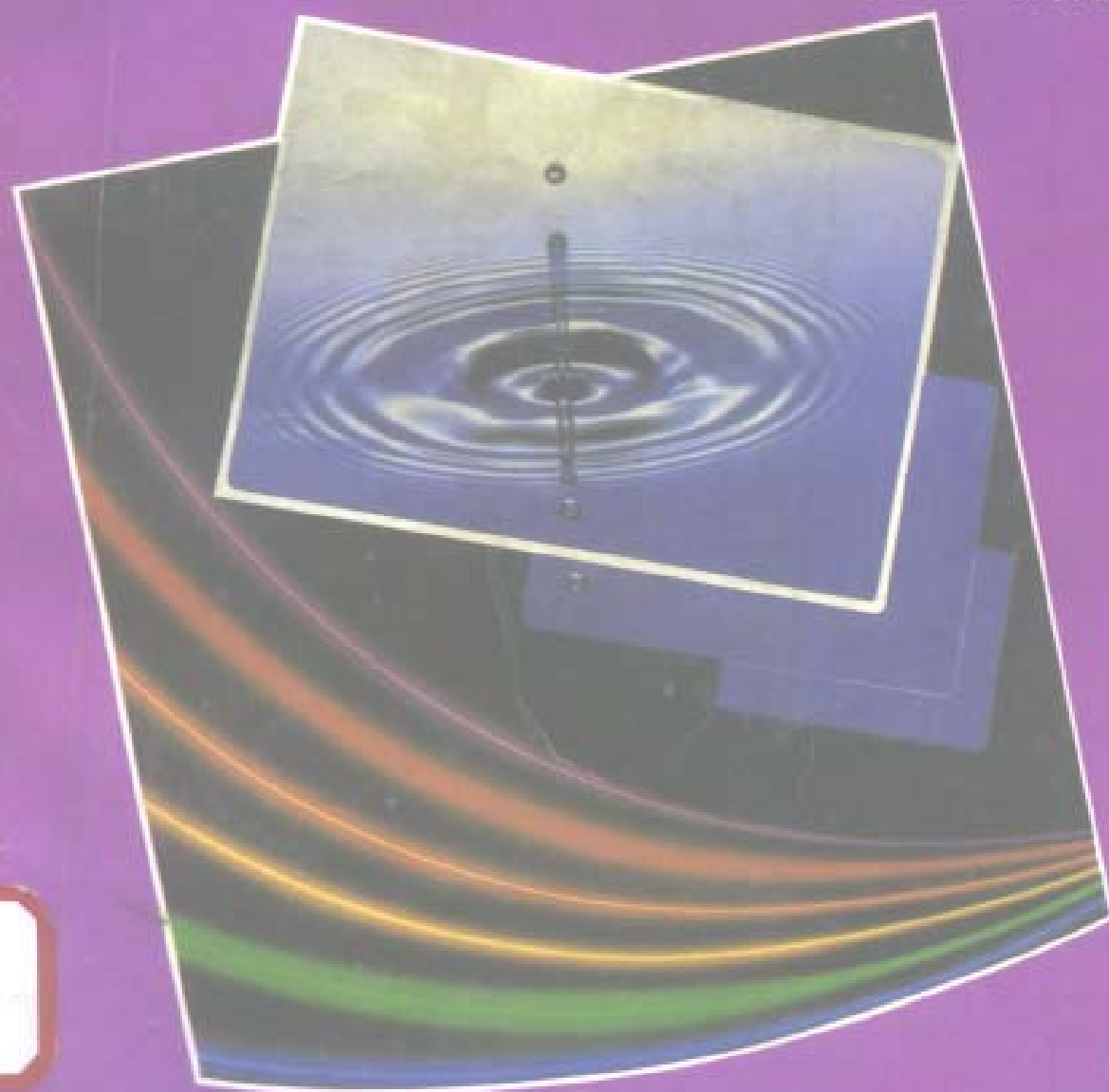


Turbo C/Borland C++

# 用户界面 程序设计

周升锋 李立新 孙传俊 编著



西安交通大学出版社

7P312  
2,840

382753

Turbo C/Borland C++  
**用户界面程序设计**

周升锋 李立新 孙传俊 编著



西安交通大学出版社

## 内容提要

本书从实例入手,详细介绍了如何用 Turbo C/Borland C 进行用户界面程序设计的一些技术和技巧。主要包括:屏幕颜色的程序设计、点阵汉字及矢量汉字的多种显示技术、弹出式窗口设计、各种菜单的设计技术、鼠标器编程、数据的输入/输出程序设计、图符及图符菜单设计、屏幕特技设计、声音及音乐程序设计、以及动画程序设计共十个部分。

书中所提供的所有源程序均在 Turbo C 2.0 和 Borland C++ 2.0 下调试通过,可直接为软件开发者所采用,对软件开发者有启迪和帮助作用。

本书适合从事计算机软件开发的工程技术人员及大中专院校师生阅读参考。

JS206/02

(陕)新登字 007 号



Turbo C/Borland C++

用户界面程序设计

周升锋 李立新 孙传俊 编著

责任编辑 叶 涛

\*

西安交通大学出版社出版

(西安市咸宁西路 28 号 邮政编码 710049)

西安理工大学印刷厂印装

陕西省新华书店经销

\*

开本 787×1092 1/16 印张:16.625 字数:401 千字

1994 年 12 月第 1 版 1994 年 12 月第 1 次印刷

印数:1—5000

ISBN7-5605-0648-8/TP·91 定 价:15.00 元  
实例软盘:25.00 元

# 前 言

用户界面(UI)是连接计算机和操作者的纽带,操作者与计算机的信息交换就是通过一个个界面来完成的。随着计算机软件技术的发展,用户界面的设计越来越成为软件设计的重点。一个成功的软件必定有良好的用户界面。在整个软件的开发过程中,用户界面的工作量是很大的,大约要占整个软件工作量的40%—60%。从某种程度上可以说:没有用户界面也就没有了软件,用户界面的好坏,不仅关系到软件的质量,而且对一个软件的被认可程度与成功起着巨大的作用。当今的计算机软件界面越来越漂亮,令计算机软件开发人员耳目一新,非常希望了解这些界面的设计技术。目前,系统地介绍用户界面设计方面的书籍还不多见。

C语言是目前颇为流行的一种计算机语言,其代码效率高、与系统十分接近,Turbo C以其小巧、快速、高效的集成开发环境及丰富的图形功能赢得了广大计算机用户的喜爱;Borland公司1991年推出的Borland C++ 2.0比Turbo C功能更强大,提供了更加丰富的函数资源。所以用Turbo C和Borland C++开发软件的用户越来越多。

为此,作者将近几年来从事C语言开发工作积累的经验 and 知识,用Turbo C/Borland C作为编程工具,从实际出发,系统介绍用户界面程序设计的一些技术和技巧,期望对广大读者的编程能够收到事半功倍的效果。

本书内容包括:

- 屏幕颜色的程序设计:包括屏幕16种颜色的使用、改变VGA设置的16种颜色及256种颜色,半色调明暗处理技术、立体块及按钮设计;
- 汉字显示技术:讲述中西文DOS下显示彩色汉字的方法,点阵汉字与矢量汉字的显示、汉字的放大旋转与倾斜显示、小字库的建立技术、立体字与空心字、256色汉字显示技术;
- 文本窗口、图形窗口及弹出式窗口程序设计;
- 关于鼠标的程序设计,鼠标的功能及函数设计,鼠标的编程应用;
- 下拉式菜单设计:中西文DOS下的中西文下拉式菜单及弹出式菜单的设计、用鼠标选择立体按钮菜单、鼠标与键盘兼容的下拉式菜单设计、交替运行其它语言程序时的菜单设计;
- 数据的输入/输出程序设计;

- 图符的创建及图符菜单的程序设计;
- 屏幕特技设计: 包括特技清屏、整屏弹出特技、连续颜色变换技术、屏幕的淡入淡出技术以及屏幕填充图案设计;
- 声音及音乐程序设计;
- 动画程序设计

共十个部分。书中列举大量的实例,全部用汉字注释,每个例程均是作者编制并且全部在 Turbo C 2.0 和 Borland C++ 2.0 下调试通过,可直接为广大读者所采用。

所有程序收集在一张高密软盘中。

本书不是 C 语言的入门教材,但也没有采用很复杂的数据结构进行描述,只要系统地学习过 C 语言并了解最基本的 DOS, BIOS 功能调用,就可以顺利阅读本书。

在本书的整个编著过程中,得到了张建教授的大力支持和帮助,李舜酩老师、孟宪皆老师以及赵庆志老师也给予了很多的帮助。在此,特向他们表示衷心地感谢!

由于水平的关系,加之时间仓促,错误之处当请批评指正。

编著者

1994 年 8 月

# 目 录

## 第 1 章 有关的基础知识及说明

1.1 软件及硬件要求 .....	(2)
1.2 目录结构及说明 .....	(2)
1.3 图形方式设置及图形独立运行程序的建立 .....	(2)
1.4 在 Turbo C/Borland C 中使用 DOS, BIOS 中断 .....	(5)
1.5 关于位操作 .....	(8)
1.6 关于端口操作函数 .....	(8)

## 第 2 章 关于颜色的程序设计

2.1 颜色的基本知识与使用原则 .....	(11)
2.1.1 颜色的类型与混合 .....	(11)
2.1.2 颜色方程式 .....	(11)
2.1.3 颜色的功能 .....	(11)
2.1.4 颜色使用的基本原则 .....	(12)
2.2 VGA 图形显示适配器简介 .....	(12)
2.2.1 VGA 的显示模式 .....	(12)
2.2.2 VGA 的结构 .....	(14)
2.2.3 VGA 的寄存器 .....	(15)
2.2.4 VGA 的数模转换器 .....	(16)
2.3 屏幕 16 种颜色的使用 .....	(17)
2.3.1 文本窗口颜色的设置 .....	(17)
2.3.2 图形方式下颜色的设置 .....	(19)
2.4 改变 VGA 设置的 16 种颜色 .....	(21)
2.5 半色调明暗处理技术 .....	(25)
2.6 VGA 256 色编程技术 .....	(27)
2.7 改变 VGA 设置的 256 种颜色 .....	(31)

2.8 颜色与视觉效果——立体块及按钮设计.....	(33)
<b>第3章 汉字显示技术</b>	
3.1 中文 DOS 下的汉字显示 .....	(40)
3.1.1 利用 C 语言的库函数显示彩色汉字 .....	(40)
3.1.2 利用 BIOS 显示彩色汉字 .....	(41)
3.2 西文 DOS 下的汉字显示 .....	(45)
3.2.1 点阵汉字.....	(46)
3.2.1.1 16 点阵汉字的显示 .....	(46)
3.2.1.2 24 点阵汉字的显示 .....	(48)
3.2.1.3 点阵汉字的无级放大、旋转与倾斜显示 .....	(51)
3.2.1.4 小字库的建立.....	(57)
3.2.2 矢量汉字.....	(65)
3.2.2.1 矢量汉字的显示.....	(65)
3.2.2.2 矢量汉字的无级放大、旋转与倾斜显示 .....	(70)
3.2.2.3 矢量汉字小字库的建立.....	(73)
3.3 立体汉字与空心汉字.....	(78)
3.4 256 色汉字显示 .....	(79)
<b>第4章 窗口程序设计</b>	
4.1 文本窗口.....	(83)
4.2 图形窗口.....	(85)
4.3 弹出式窗口系统.....	(86)
4.3.1 弹出式窗口.....	(86)
4.3.2 弹出式窗口的改进.....	(92)
<b>第5章 关于鼠标的程序设计</b>	
5.1 鼠标的功能介绍及功能函数设计 .....	(104)
5.1.1 鼠标的常用功能介绍 .....	(104)
5.1.2 鼠标的基本函数设计 .....	(105)
5.2 鼠标的应用编程 .....	(107)
<b>第6章 菜单程序设计</b>	
6.1 西文 DOS 下的菜单设计.....	(113)
6.1.1 西文 DOS 下的西文下拉式菜单的设计.....	(113)
6.1.2 西文 DOS 下的汉字下拉式菜单设计.....	(124)
6.1.3 西文 DOS 下的汉字弹出式下拉菜单设计.....	(135)
6.2 中文 DOS 下的汉字下拉式菜单设计.....	(143)
6.3 用鼠标选择立体按钮菜单 .....	(151)
6.4 鼠标和键盘兼容的下拉式菜单设计 .....	(157)
6.5 交替运行其它语言程序时的菜单设计 .....	(168)
<b>第7章 数据输入/输出程序设计</b>	
7.1 标准输入/输出函数.....	(175)

7.1.1	格式化输入输出函数 .....	(175)
7.1.2	非格式化输入输出函数 .....	(179)
7.2	文件的输入/输出函数 .....	(182)
7.2.1	标准文件函数 .....	(182)
7.2.2	非标准文件函数 .....	(187)
7.3	可编辑的数据输入程序设计 .....	(189)
7.4	图形方式下的数据输入 .....	(193)
7.5	图形方式下的文本输出 .....	(196)
7.5.1	文本输出函数 .....	(196)
7.5.2	有关文本字体、字型和输出方式的设置 .....	(197)
7.5.3	用户对文本字符大小的设置 .....	(199)
7.6	格式化数据输出 .....	(203)
7.6.1	格式化文本输出 .....	(204)
7.6.2	文本覆盖 .....	(204)
7.6.3	突出显示文本 .....	(204)
7.7	西文字符的放大、旋转与倾斜显示 .....	(207)
7.7.1	矢量字库*.CHR 的结构分析 .....	(207)
7.7.2	放大、旋转与倾斜 .....	(208)
<b>第8章 图符及图符菜单</b>		
8.1	创建图符 .....	(213)
8.2	图符菜单设计 .....	(218)
<b>第9章 屏幕特技设计</b>		
9.1	特技清屏 .....	(225)
9.2	整屏弹出特技 .....	(228)
9.3	连续颜色变换技术 .....	(230)
9.4	屏幕的淡入淡出技术 .....	(233)
9.5	屏幕图案设计 .....	(236)
<b>第10章 声音及音乐程序设计</b>		
10.1	发声函数及音乐程序设计 .....	(241)
10.2	振铃的声响技术 .....	(243)
<b>第11章 动画程序设计 .....</b>		
<b>附录一 书中程序出现章节对照表 .....</b>		<b>(255)</b>
<b>附录二 本书常用的 ASCII 码表 .....</b>		<b>(256)</b>
<b>参考文献 .....</b>		<b>(257)</b>



# 有关的基础知识及说明

## 第 **1** 章

本章内容:

- 1.1 软件及硬件要求
- 1.2 目录结构及说明
- 1.3 图形方式设置及图形独立运行程序的建立
- 1.4 在 Turbo C/Borland C 中使用 DOS, BIOS 中断
- 1.5 关于位操作
- 1.6 关于端口操作函数

## 1.1 软件及硬件要求

本书以 Turbo C 2.0 和 Borland C++ 2.0 为蓝本,因此,必须有一套完整的 Turbo C 2.0 或 Borland C++ 2.0。除此之外,要顺利使用本书例程,应具有以下软硬件配置:

硬件 80286 以上主机或兼容机

VGA(TVGA)彩色高分辨率(640×480)图形显示器

标准键盘

[鼠标]

软件 DOS 3.0 以上版本操作系统

UCDOS 3.0 汉字操作系统

[华光排版系统的矢量汉字库 SLP 文件]

另外,本书介绍的函数和许多语句是 C 语言的标准,即其它 C 语言也有,但作者未对它们进行区分,而叙述成 Turbo C/Borland C 的函数和语句,特此说明。

## 1.2 目录结构及说明

本书将 Turbo C 或 Borland C 按下面的目录结构进行叙述:

ROOT—TC(BC)	—	INCLUDE	头文件目录
	—	LIB	库文件目录
	—	BGI	BGI 和 CHR 文件目录
	—	[USER]	用户文件目录,也可直接放在 TC(BC)子目录下。

Turbo C 2.0 的扩展名为: .C; Borland C++ 2.0 的扩展名为: .CPP 或 .C(默认为 .CPP);

Turbo C 2.0 较 Borland C++ 2.0 具有更快的编译速度,但语法检查不够严格。作者建议 Borland C++ 2.0 的用户编程时先在 Turbo C 2.0 下调试程序,然后在 Borland C 2.0 下编译。

## 1.3 图形方式设置及图形独立运行程序的建立

本书中大部分程序是在图形方式下编制的。Turbo C/Borland C 提供了非常丰富的图形函数,所有图形函数的原形均在 graphics.h 中,使用图形函数时要确保有显示器图形驱动程序 \*.BGI,同时应将集成开发环境 Options/Linker 中的 Graphics Lib 选为 on,只有这样才能保证正确使用图形函数。

使用图形函数之前,应首先将屏幕设置为图形模式,可用下面的函数:

```
void far initgraph(int far * GraphDriver,int far * GraphMode,char * Path);
```

其中,GraphDriver 和 GraphMode 分别表示图形驱动器和模式,Path 是指图形驱动程序所在的目录路径。对 VGA 显示适配器,其图形模式的符号常数及对应的分辨率如表 1-1。

表 1-1 VGA 图形驱动器、模式的符号常数及数值

图形驱动器		图形模式		色 调	分辨率
符号常数	数 值	符号常数	数 值		
VGA	9	VGALO	0	16 色	640×200
VGA	9	VGAMED	1	16 色	640×350
VGA	9	VGAHI	2	16 色	640×480
DETECT	0	用于硬件测试			

当 GraphDriver = DETECT 时,表示用于硬件测试,对 VGA 取屏幕最高分辨率 640×480。

图形驱动程序由 Borland 公司提供,文件扩展名为 BGI。EGA,VGA 图形适配器的驱动程序为 EGAVGA.BGI。

另外,Turbo C/Borland C 提供了退出图形状态的函数 closegraph(),其调用格式如下:

```
void far closegraph(void);
```

调用该函数后可退出图形状态而进入文本方式(默认方式),并释放用于保存图形驱动程序和字体的系统内存。

Turbo C/Borland C 的图形库是由 graphics.h,GRAPHICS.LIB,\*.BGI,\*.CHR 四部分组成的,其中 VGA 的驱动程序名为 EGAVGA.BGI。\*.CHR 文件有四个,分别为:TRIP.CHR(三倍笔划字体),LITT.CHR(小号笔划字体),SANS.CHR(无衬线笔划字体)和 GOTH.CHR(黑体笔划字体),对于用 initgraph() 函数直接进行图形初始化程序,在编译和连接时并没有将相应的驱动程序(\*.BGI)装入到可执行程序,当程序运行到 initgraph() 语句时,再从该函数中的第三个形式参数 char \* Path 中所规定的路径中寻找相应的驱动程序。若没有驱动程序,则在 TC(BC)子目录中去找,如果 TC(BC)子目录中仍没有或 TC(BC)子目录不存在,将会出现下列错误:

BGI Error: Graphics not initialized (use 'initgraph').

而对于图形方式下的文字输出,若当前目录下无相应的 \*.CHR 文件,则输出的字体及文本大小的定义将无效。

因此,为了使用方便,应该建立一个不需要驱动程序或字体文件就能独立运行的可执行图形程序,我们可用下列步骤(以 EGAVGA 为例):

(1) 将驱动程序 EGAVGA.BGI 转换成 EGAVGA.OBJ 的目标文件:

确保 BGI0BJ.EXE 在 TC(BC)子目录下,在 TC(BC)子目录下,键入:

```
BGI0BJ BGI\EGAVGA
```

将字体文件 \*.CHR 转换成 OBJ 文件(以 TRIP 为例):

BGIOBJ BGI\TRIP (另外三个 CHR 文件方法相同)

(2) 将以上建立的 OBJ 文件加入到 GRAPHICS.LIB 库文件中:

在 TC(BC)子目录下,键入:

TLIB LIB\GRAPHICS.LIB+EGAVGA (也可用 PRJ,TLINK)

TLIB LIB\GRAPHICS.LIB+TRIP

说明:

如果不希望 EGAVGA.OBJ 文件和 TRIP.OBJ 等文件加在 GRAPHICS.LIB 中时,可以采用下面的方法将其从 GRAPHICS.LIB 中除去掉:

TLIB LIB\GRAPHICS.LIB-EGAVGA.OBJ

TLIB LIB\GRAPHICS.LIB-TRIP.OBJ

(3) 在程序中 initgraph()函数调用之前,加上一句:

registerbgidriver(EGAVGA \_\_driver);

该函数告诉连接程序在连接时把 EGAVGA 的驱动程序装入到用户的执行程序当中;

对于文字字体,在调用之前,要加:

registerbgifont(符号名);

表示将程序连接时,把字体文件装入到用户程序。

有关驱动程序和字体名字如表 1-2。

表 1-2 驱动程序和字体名字表

驱动程序文件 (*.BGI)	registerbgidriver 符号名	字体文件 (*.CHR)	registerbgifont 符号名
CGA	CGA __driver	TRIP	triplex __font
EGAVGA	EGAVGA __driver	LITT	small __font
HERC	Here __driver	SANS	sansserif __font
ATT	ATT __driver	GOTH	gothic __font

经过以上处理,编译连接后的执行程序在任何目录或其它兼容机上都可运行。

作过上述几步之后,初始化屏幕为图形模式的函数可以写成:

```
void InitGra(void){
    int GraphDrive = DETECT,GraphMode;          /* 定义为 DETECT 方式 */
    registerbgidriver(EGAVGA __driver);          /* 登记的图形驱动名 */
    registerbgifont(triplex __font);              /* 使用三倍笔划字体 */
    registerbgifont(small __font);                /* 使用小号笔划字体 */
    registerbgifont(sansserif __font);            /* 使用无衬线笔划字体 */
    registerbgifont(gothic __font);               /* 使用黑体笔划字体 */
    initgraph(&GraphDrive,&GraphMode,"");
}
```

## 1.4 在 Turbo C/Borland C 中使用 DOS, BIOS 中断

Turbo C/Borland C 提供了一些与操作系统有关的函数,这些函数使用起来比较灵活,可实现许多高级功能。

### 1. DOS 软中断功能调用

DOS 的软中断(21H 中断)具有强大的功能, Turbo C/Borland C 可用函数 `intdos()` 直接访问这些系统调用,其调用格式为:

```
int intdos(union REGS * Inregs, union REGS * Outregs);
```

该函数表示用 `Inregs` 指向的联合中的内容所确定的 DOS 系统调用,执行一次 DOS 21H 中断,并将结果存入指定的联合中。该函数的头文件为 `dos.h`,联合 `REGS` 的定义如下:

```
union REGS {  
    struct WORDREGS x;  
    struct BYTEREGS h;  
}
```

其中, `WORDREGS` 和 `BYTEREGS` 的结构分别为:

```
struct WORDREGS {  
    unsigned int ax, bx;  
    unsigned int cx, dx;  
    unsigned int si, di;  
    unsigned int cflag, flags;  
}  
  
struct BYTEREGS {  
    unsigned char al, ah;  
    unsigned char bl, bh;  
    unsigned char cl, ch;  
    unsigned char dl, dh;  
}
```

下面举例说明 `intdos()` 函数的用法: 读取系统日期并显示。

在 DOS 功能调用中, 读取系统日期的子功能是 2AH:

入口参数:

AH = 2AH; 得到系统日、星期、月、年。

出口参数:

CX = 年(1980-2099)

DH = 月(1-12)

DL = 日(1-31)

AL = 星期(0=星期天; 1=星期一等)

下面是利用 Turbo C 的 `intdos()` 函数写成的源程序:

```

/*
RDSYSDT1.C —— 读取系统日期并输出,用 intdos()函数
*/
#include "dos.h"
#include "conio.h"
int main()
{
union REGS In,Out;      /* 定义结构变量 */
In.h.ah = 0x2a;         /* 设置 DOS 21H 类中断功能调用 2AH */
intdos(&In,&Out);        /* 执行中断 */
printf("%4d 年%2d 月%2d 日\n",Out.x.cx,Out.h.dh,Out.h.dl);
printf("星期%1d\n",Out.h.al);
getch();
return 0;
}

```

## 2. 关于 BIOS, DOS 软中断调用的函数 int86()

BIOS 除了打印机(17H 中断)和键盘(16H 中断)之外,还有显示器驱动(10H 中断)、软盘服务中断(13H)等。Turbo C/Borland C 提供了关于这些中断的函数 int86(),其调用格式为:

```
int int86(int intnum, union REGS * In, union REGS * Out);
```

其中,intnum 为 BIOS 的软中断,联合 REGS 与上节中所述相同。该函数用 In 指向的联合变量中的内容所确定的系统调用,执行一次 intnum 规定的软中断,并将结果放入 Out 指向的联合中,该函数的头文件为 dos.h。

只要将 intnum 定义为 0x21,则该函数与 intdos()函数功能相同。

将上例中的函数 intdos()改用 int86(),即变成:

```

/*
RDSYSDT2.C —— 读取系统日期并输出,用 int86()函数
*/
#include "dos.h"
#include "conio.h"
int main()
{
union REGS In,Out;      /* 定义结构变量 */
In.h.ah = 0x2a;         /* 设置 DOS 21H 类中断功能调用 2AH */
int86(0x21,&In,&Out);    /* 执行中断 */
printf("%4d 年%2d 月%2d 日\n",Out.x.cx,Out.h.dh,Out.h.dl);
printf("星期%1d\n",Out.h.al);
getch();
return 0;
}

```

### 3. 本书中用到的 DOS 及 BIOS 中断

#### (1) 输出字符功能 09H(INT 10H)

入口参数:

AH = 09H

BH = 页号

CX = 字符数

AL = 输出的字符

BL = 输出字符的属性

出口参数: 无。

#### (2) 写图形象素功能 0CH(INT 10H)

提供一种在图形模式中与设备独立的,但很慢的处理象素的方法。

入口参数:

AH = 0CH;

BH = 页面号

AL = 颜色

CX = 横坐标 X

DX = 纵坐标 Y

执行: INT 10H

出口参数: 无。

#### (3) 读屏幕上点(x,y)图形象素的功能

入口参数:

AH = 0DH

CX = x

DX = y

出口参数: AL = 象素值(范围取决于图形方式)

#### (4) 读 VGA 设置的颜色功能 1015H(INT 10H)

入口参数:

AH = 10H

AL = 15H

BX = 颜色号

执行: INT 10H

出口参数:

DH = 红色分量

CH = 绿色分量

CL = 蓝色分量

#### (5) 设置 VGA 初始设置的颜色 1010H(INT 10H)

入口参数:

AH = 10H

AL = 10H

BX = 颜色号  
 DH = 红色分量  
 CH = 绿色分量  
 CL = 蓝色分量

执行: INT 10H

出口参数: 无。

## 1.5 关于位操作

C 语言与其它高级语言的不同是它完全支持按位运算符。这与汇编语言的位操作有些相似。

Turbo C/Borland C 中按位运算符如表 1-3。

表 1-3 按位操作运算符

操作符	作用
&	位逻辑与
	位逻辑或
^	位逻辑异或
~	位逻辑反
>>	右 移
<<	左 移

按位运算是对于字节或字中的实际位进行检测、设置或移位,它只适用于字符型和整型变量以及它们的变体,对其它数据类型不适用。

移位运算符“>>”和“<<”是指将变量中的每一位向右或向左移动,其通常形式为:

右移: 变量名 >> 位的位数

左移: 变量名 << 移位的位数

经过移位以后,一端的位被“挤掉”,另一端空出的位以 0 填补。所以, Turbo C/Borland C 中的移位不是循环移动的。

## 1.6 关于端口操作函数

Turbo C/Borland C 可以对端口操作,其中包括: 从端口读入数据,写数据到端口。

```
int inport(int portid);
```

该函数从参数 portid 指定的输入硬件端口读入一个字的低字节,从 portid+2 中读入高



字节。

```
unsigned char inportb(int portid);
```

该函数从参数 portid 指定的输入硬件端口读入一个字节,函数的原型在 dos.h。

```
void outport(int portid,int value);
```

该函数将给定字 value 的低字节写在 portid 所指的输出端口中,高位字节写在 portid+1 所指定的输出端口中。

```
void outportb(int portid,unsigned char value);
```

该函数将给定的字节值 value 输出到 portid 所指定的输出端口中。

# 关于颜色的程序设计

## 第 2 章

本章内容：

### 2.1 颜色的基本知识与使用原则

#### 2.1.1 颜色的类型与混合

#### 2.1.2 颜色方程式

#### 2.1.3 颜色的功能

#### 2.1.4 颜色使用的基本原则

### 2.2 VGA 图形显示适配器简介

#### 2.2.1 VGA 的显示模式

#### 2.2.2 VGA 的结构

#### 2.2.3 VGA 的寄存器

#### 2.2.4 VGA 的数模转换器

### 2.3 屏幕 16 种颜色的使用

#### 2.3.1 文本窗口颜色的设置

#### 2.3.2 图形方式下颜色的设置

### 2.4 改变 VGA 设置的 16 种颜色

### 2.5 半色调明暗处理技术

### 2.6 VGA 256 色编程技术

### 2.7 改变 VGA 设置的 256 种颜色

### 2.8 颜色与视觉效果——立体块及按钮设计

我们生活在一个充满色彩的世界里,颜色能吸引我们的注意,影响我们的情绪,向我们传递特定的含义,并且在美学上颜色可以让我们感觉愉悦或不悦。目前的计算机软件几乎都使用了非常丰富的颜色,颜色在用户界面设计中是一个非常重要的内容。

## 2.1 颜色的基本知识与使用原则

### 2.1.1 颜色的类型与混合

在自然界里,红、橙、黄、绿、青、蓝、紫七种颜色我们称其为标准色,或称“彩虹”七色。色光是具有一定波长和频率的电磁波,波长在(380~760)nm的电磁辐射能引起人们的视觉,并把它们感知为各种颜色。

颜色的三原色是红、黄、蓝,红+黄=紫,黄+蓝=绿,颜色的混合是愈混合愈暗,颜色的三原色等量相加得到近乎黑色;而光的三原色是红、绿、蓝,红+蓝=洋红,绿+蓝=青,红+绿=黄,色光的三原色等量相加为白光。

色彩系列的颜色由色相、明度、纯度三个基本要素构成:

色相:即某一颜色品质所独具的相貌特征,如红、橙、黄、绿、青、蓝、紫是不同色相的颜色;

明度(光亮度):是指色彩的明暗程度;

纯度(饱和度):是指一个颜色的纯洁程度。

### 2.1.2 颜色方程式

任何色光都可由红、绿、蓝三原色混和而成,改变三者的不同强度比例,便能产生各种色光。因此,对于任何一种确定的颜色,都可用三原色相加的形式来表示,这种表示法可写出如下颜色方程式:

$$(C)=r(R)+g(G)+b(B)$$

其中,(C)为某特定的颜色,(R),(G),(B)为红、绿、蓝三原色, $r,g,b$ 为三原色的比例系数,“=”是指匹配,即视觉上颜色相同。

### 2.1.3 颜色的功能

(1) 颜色的冷暖感:如红、橙、黄系列的颜色感觉上是温暖的,称为暖色;蓝、绿、紫系列的颜色称为冷色。

(2) 重量感:主要取觉于颜色的明度,明度高显得轻,明度低显得重。

(3) 尺寸感:明度高的颜色和暖色具有扩散的感觉,给人以膨胀的感觉。明度低的颜色和冷色具有内聚的作用,给人以缩小的感觉。

(4) 距离感:不同的颜色在不同背景的对比作用之下,可使对色彩在感觉上产生距离上

的变化,造成有进、退、凸、凹、远、近不同的感觉。一般来说,高亮度和暖色系列的颜色具有进、凸,接近的感觉;而明度较低和冷色系列的颜色具有退、凹,远离的感觉。

(5) 软硬感:取觉于色彩的明度和纯度,明色软,暗色硬;中纯度色感软,高纯度或低纯度硬,黑与白是坚固色,灰色是柔软色。

(6) 情绪感:明度较低,柔和的冷色,给人有稳重和宁静的感觉,暖色系列颜色给人以兴奋感,激发人的情绪,冷色系列颜色给人以沉静感,可抑制人的情绪。

#### 2.1.4 颜色使用的基本原则

(1) 一次显示中使用的颜色数目不要太多,一般以不超过 7 种为宜。

(2) 由于眼睛的生理机能,对红色和绿色不敏感。因此,红色和绿色宜布置在显示器中央,而不宜布置在边缘。

(3) 由于眼睛对蓝色最不敏感,所以在小区域中最不易感知蓝色,蓝色不适用于正文、细线或小形状上,而最适宜于作为背景或大面积区域。

(4) 当相邻的颜色间唯一的差别是蓝色亮度的不同时,用户就很难分辨出客体间的边缘,所以,应避免在相邻使用仅在蓝色亮度上有差异的两种颜色。

(5) 对颜色组合的选择是非常复杂的,通常最好不要选用下列的颜色组合:

蓝/黄,红/绿,红/蓝,绿/蓝

(6) 将亮色放在暗色旁边以保持与黑白显示器兼容。

### **2.2 VGA 图形显示适配器简介**

VGA 图形显示适配器,通常简称 VGA (Video Graphics Array,即视频图形阵列),是 IBM 公司于 1987 年推出的图形显示器。它与 MDA,CGA 保持兼容,但在分辨率、颜色、容量、速度等方面有了明显的改进和提高,成为国际市场上个人计算机图形卡的一种标准。绝大多数计算机制造商均以与 EGA/VGA 保持兼容作为其产品设计目标,而开发 PC 机软件的公司则几乎无一例外地都以 EGA/VGA 作为其软件运行的基本要求。

VGA 具有 256 个调色寄存器(颜色选择寄存器)。每个寄存器有 18 位,一共有 262 144 种不同的颜色(最多可同时显示 256 种)。

VGA 具有自己的 ROM BIOS(Basic Input Output System),它们为系统中其它软件提供各种信息显示操作的服务,替代并大大扩充了原先系统内 BIOS 中的 INT 10H 的全部处理功能,它包括模式选择、页面切换、输出字符及属性、设置调色寄存器和色彩选择寄存器等。

#### 2.2.1 VGA 的显示模式

为了满足多种多样的应用需求,VGA 具有多种不同分辨率和不同数目颜色的图形显示模式,它们由软件进行控制和选择。显示模式通常由三个因素决定:显示内容、显示分辨率和

显示颜色。按显示内容分,显示模式可以分成两大类——文字模式(Text Mode)和图形模式(Graphics Mode);文字模式时显示存储器中存放的是字符的编码,显示操作以字符为单位进行;图形模式时,显示存储器中存放的是被显示的图画和文字的映象,显示操作以像素为单位进行。按显示的颜色分,显示模式有单色(两种颜色)和彩色(多种颜色)两大类。若按分辨率分,则显示模式往往有若干种。

VGA 的显示模式如表 2-1 和表 2-2。

表 2-1 VGA 的各种文字显示模式

显示模式	颜色	字符分辨率	象素分辨率	字符点阵	页面数目	缓冲区首址
0,1	彩色	40×25	320×200	8×8	8	B8000
0*,1*	彩色	40×25	320×350	8×14	8	B8000
0+,1+	彩色	40×25	360×400	9×16	8	B8000
2,3	彩色	80×25	640×200	8×8	8	B8000
2*,3*	彩色	80×25	640×350	8×14	8	B8000
2+,3+	彩色	80×25	720×400	9×16	8	B0000
7	单色	80×25	720×350	9×14	8	B0000
7+	单色	80×25	720×400	9×16	8	B0000
非标准	彩色	80×43	640×350	8×8		
非标准	彩色	80×50	640×400	8×8		

表 2-2 VGA 的各种图形显示模式

显示模式	颜色数目	象素分辨率	象素位数	字符点阵	页面数目	缓冲区首址
6	2	640×200	1	8×8	1	B8000
FH	2	640×350	2	8×14	2	A0000
11H	2	640×480	1	8×16	1	A0000
4,5	4	320×200	2	8×8	1	A0000
DH	16	320×200	4	8×8	8	A0000
EH	16	640×200	4	8×8	4	A0000
10H	16	640×350	4	8×14	2	A0000
12H	16	640×480	4	8×16	1	A0000
13H	256	320×200	4	8×8	1	A0000

### 2.2.2 VGA 的结构

VGA 本身不带处理器,由中央处理器(80x86)负责把物体的几何描述转换成显示存储器中的映象(位图)。VGA 的结构图如图 2-1。

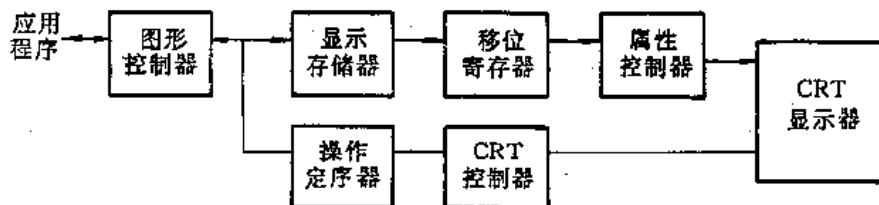


图 2-1 VGA 结构方框图

它是由下列六个主要部分组成的：

- 显示存储器：这是容量为 64KB—256KB(有些可达 512KB 甚至 1MB)的动态随机存储器,它分成 4 个彩色平面,用来存放被显示图形的位图；
- 图形控制器：它位于中央处理器和显示存储器之间,能对被写入显示存储器去的数据执行与、或、异或和循环移位等逻辑运算,从而大大简化了画图操作；
- CRT 控制器：用来产生各种定时信号,控制 CRT 监视器的显示刷新操作；
- 串-并转换器：它从显示存储器中一次取出一个或多个字节的显示信息,把它转换成为串行信号送到属性控制器中去；
- 属性控制器：包含颜色对照表(LUT),它负责把显示存储器中的像素值翻译成为送往 CRT 去的彩色信息；
- 操作定序器：它对 VGA 所有的操作进行控制和协调,也对彩色平面的有效与否进行控制。

其中,属性控制器的主要作用是决定在屏幕上显示的图形或字符的颜色。其核心是一张颜色对照表(也叫配色器,调色器),它把显示存储器读出的 4 个二进制位的像素数据转换成 6 位颜色信息。当软件选择不同的显示模式时,BIOS 将使用相应的数据装入颜色对照表中。例如,在单色模式时,颜色表中仅有两种不同的颜色;CGA 模式时,颜色表中装入的是 CGA 所支持的几种颜色;而在 EGA/VGA 独特的一些模式时,颜色表中将会有更加丰富的颜色,通过编程,软件还可以在任何时候重新定义颜色表中的颜色。

VGA 的属性控制器是把显示存储器中读出的像素数据转换成 8 位的颜色号,然后送到 VGA 所特有的数模转换器(DAC)中去,DAC 中另有一张颜色对照(含 256 个颜色选择寄存器),通过查表可以得到 18 位的颜色信号,最后分三组(红、绿、蓝),进行数模转换,产生三个颜色的模拟信号提供给 VGA 模拟式监视器之用。

### 2.2.3 VGA 的寄存器

VGA 的各种图形和文字功能都是通过内部的许多寄存器进行控制的。这些寄存器均为一个字节宽，它们通过 CPU 的输入输出指令进行读出和写入。由于 VGA 的显示模式多，功能丰富。因此，寄存器的数量多，结构也较复杂。VGA 大约有 60 多个寄存器，它们大多数既可读又可写。

VGA 的寄存器按其功能和位置分布可分成以下几组：

CRT 控制器的寄存器；

操作定序器的寄存器；

图形控制器的寄存器；

属性控制器的寄存器；

外部寄存器(不包含在大规模集成电路芯片上的)；

另外，还有数模转换寄存器组，用来把离散的颜色信号转换成模拟信号送到模拟式监视器去。

为了避免占用过多的 CPU 输入\输出空间，除了外部寄存器各有它们自己唯一的输入/输出端口地址外，其它各组寄存器的存取操作均需分两步进行。首先把欲读写的寄存器在组内的编号(称为索引号)输出到该组的地址(索引)寄存器中，然后再对该组的数据寄存器进行读或写操作，每一组的索引寄存器和数据寄存器均有它们自己的输入/输出地址。

表 2-3 是 VGA 部分寄存器的输入/输出地址一览表。

表 2-3 VGA 部分寄存器的输入/输出地址表

寄存器名	输入/输出	单色显示模式	彩色显示模式
操作定序索引寄存器	输出	3C4H	3C4H
操作定序数据寄存器	入/出	3C5H	3C5H
CRT 控制索引寄存器	输出	3B4H	3D4H
CRT 控制数据寄存器	入/出	3B5H	3D5H
图形控制索引寄存器	输出	3CEH	3CEH
图形控制数据寄存器	入/出	3CFH	3CFH
属性控制索引/数据寄存器	入/出	3C0H	3C0H
DAC 颜色表读索引寄存器	输出	3C7H	3C7H
DAC 颜色表写索引寄存器	输出	3C8H	3C8H
DAC 颜色表数据寄存器	入/出	3C9H	3C9H





0—255。程序需要向某个颜色寄存器写入时,首先把该颜色寄存器的索引号用语句 `outportb()` 送到颜色表地址寄存器(输出地址为 3C8H),此时颜色表处于写模式。然后连续三次使用输出命令 `outportb()` 把颜色数据写到颜色表数据寄存器(输出地址为 3C9H)中,第一次是红色的亮度值,第二次是绿色亮度值,第三次是蓝色亮度值,每个值均为 6 位宽。这样,18 位的颜色值就被写入到索引号所指向的那个颜色寄存器。程序需要从某个颜色寄存器中读出其内容时,过程大体相似。首先,把颜色寄存器的索引号用输出指令送到颜色表地址寄存器(输出地址为 3C7H),此时,颜色表转换成读模式。然后再连续三次使用输入语句 `inportb()` 读颜色表数据寄存器(输入地址为 3C9H),此时,CPU 三次取得的数据分别是索引号所指向的那个颜色寄存器的红色、绿色和蓝色的亮度值。上面所说的颜色寄存器的读、写操作完成之后,颜色表地址寄存器将自动加 1,指向下一个颜色寄存器。

由于颜色寄存器只能在读模式的时候读取其内容,在写模式的时候修改其内容,否则操作无法正确进行。为了了解颜色表处于何种工作状态,程序可以使用输入语句 `inportb()` 读入 DAC 状态寄存器(输入地址为 3C7H)的内容,若最低两位为 00,则表示颜色表处于读模式,若最低两位为 11,则表示颜色表处于写模式。

## 2.3 屏幕 16 种颜色的使用

VGA 的屏幕具有两种模式:即文本模式和图形模式。Turbo C/Borland C 可以分别对这两种模式进行颜色的设置。

### 2.3.1 文本窗口颜色的设置

文本窗口颜色的设置包括背景颜色的设置和前景颜色的设置,使用的函数及其调用格式为:

设置背景色: `void textbackground(int Color);`

设置前景色: `void textcolor(int Color);`

VGA 在彩色模式下,可同时使用 16 种颜色,这 16 种颜色系统设置了初值,有关这些颜色的定义见表 2-4。

表 2-4

有关屏幕颜色的定义

符号常数	数值	含义	字符或背景	符号常数	数值	含义	字符或背景
BLANK	0	黑	两者均可	DARKGRAY	8	深灰	只用于字符
BLUE	1	蓝	两者均可	LIGHTBLUE	9	淡蓝	只用于字符
GREEN	2	绿	两者均可	LIGHTGREEN	10	淡绿	只用于字符
CYAN	3	青	两者均可	LIGHTCYAN	11	淡青	只用于字符
RED	4	红	两者均可	LIGHTRED	12	淡红	只用于字符
MAGENTA	5	洋红	两者均可	LIGHTMAGENTA	13	淡洋红	只用于字符
BROWN	6	棕	两者均可	YELLOW	14	黄	只用于字符
LIGHTGRAY	7	淡灰	两者均可	WHITE	15	白	只用于字符
BLINK	128	闪烁	只用于字符				

在上表中的符号常数与相应的数值等价,二者可以互换。例如:设定蓝色背景,可以使用 `textbackground(1)`,也可以使用 `textbackground(BLUE)`,两者没有任何区别,只不过后者可读性好些。

Turbo C/Borland C 另外还提供了一个函数,可以同时设置文本的字符颜色和背景颜色,这个函数的调用格式为:

```
void textattr(int attr);
```

其中: `attr` 的值表示颜色形式编码信息,每一位代表的含义如下:

位:	7	6	5	4	3	2	1	0
	B	b	b	b	c	c	c	c
	↑							
	闪烁	背景颜色			前景颜色			

字节低四位 `cccc` 设置字符的颜色(0—15),4—6位 `bbb` 设置字符的背景颜色,第7位 `B` 设置字符是否闪烁。假定要设置蓝底黄字,则定义方法如下:

```
textattr(YELLOW+(BLUE<<4));或
```

```
textattr(1e);
```

若再要求闪烁,则定义变为:

```
textattr(128+YELLOW+(BLUE>>4));或
```

```
textattr(BLINK YELLOW BLUE);
```

说明:

(1) 对于背景,只有 0—7 共 8 种颜色,若大于 7 小于 15 的颜色值,则代表的颜色与减 7 后的值对应的颜色相同。

(2) 用 `textbackground()` 和 `textcolor()` 函数设置了背景与字符的颜色后,在没有用 `clrscr()` 函数清除窗口之前,颜色不会改变,直到使用了函数 `clrscr()`,整个窗口和随后输出到窗口中的文本字符才会变成新的颜色。

下面的例程是关于文本方式下颜色的设置的:

```
/*
COLOR16T.C—— 文本方式下的 16 种颜色显示
*/
#include "conio.h"
#include "process.h"
void Color16T();
int main()
{
    textbackground(0);          /* 设置背景颜色 */
    clrscr();
    Color16T();
    getch();
    textbackground(0);
    window(1,1,80,25);
    clrscr();
}
```

```

    exit (0);
}
void Color16T()
{
    int i;
    for (i = 1; i < 8; i++)
    {
        window(10+5*i,5+i,30+5*i,15+i); /* 开窗口 */
        textbackground(i);
        clrscr();
    }
    for (i = 0; i < 16; i++) /* 最后一个窗口输出文字 */
    {
        if (i < 8) gotoxy(1,2+i); /* 分两行显示 */
        else gotoxy(12,2+i-8);
        textcolor(i); /* 设置输出文字颜色 */
        cprintf("Color: %d", i); /* 输出文字 */
    }
}

```

该程序在屏幕上不同位置画出了 7 个窗口,其背景色分别使用了 7 种不同的颜色,然后在最后一个窗口中显示了 16 种颜色的文字。

### 2.3.2 图形方式下颜色的设置

对于图形模式下的屏幕颜色设置,同样也分为背景颜色的设置和前景颜色的设置。在 Turbo C/Borland C 中,分别用下面两个函数:

设置背景色: void far setbkcolor(int Color);

设置前景色: void far setcolor(int Color);

其中,Color 为图形方式下颜色的规定数值,对 EGA/VGA 显示适配器,有关颜色的符号常数及数值如表 2-5 所示。

表 2-5 有关屏幕颜色的符号常数表

符号常数	数值	含义	符号常数	数值	含义
BLANK	0	黑	DARKGRAY	8	深灰
BLUE	1	蓝	LIGHTBLUE	9	淡蓝
GREEN	2	绿	LIGHTGREEN	10	淡绿
CYAN	3	青	LIGHTCYAN	11	淡青
RED	4	红	LIGHTRED	12	淡红
MAGENTA	5	洋红	LIGHTMAGENTA	13	淡洋红
BROWN	6	棕	YELLOW	14	黄
LIGHTGRAY	7	淡灰	WHITE	15	白

清除图形屏幕内容使用清屏函数,其调用格式如下:

```
void far cleardevice(void);
```

有关图形方式下屏幕 16 种颜色的设置见下例:

```
/*
   Color16G.c——图形方式下显示 EGA/VGA 设置的 16 种颜色
*/
#include "dos.h"
#include "conio.h"
#include "process.h"
#include "stdio.h"
#include "graphics.h"
void InitGra();
void Quit();
void Color16G();
int main()
{
    InitGra();
    Color16G();
    getch();
    Quit();
    return 0;
}
void InitGra() /* 初始化屏幕为图形方式 */
{
    int GraphDrive = DETECT, GraphMode;
    registerbgidriver(EGAVGA_driver);
    initgraph(&GraphDrive, &GraphMode, "");
}
void Quit()
{
    closegraph();
    exit(0);
}
void Color16G()
{
    int i, x = 0;
    for (i = 0; i < 16; i++)
    {
        setfillstyle(1, i); /* 设置填充颜色 */
    }
}
```

```

        bar(x,20,x+40,460); /* 画矩形块 */
        x += 40;
    }
}

```

## 2.4 改变 VGA 设置的 16 种颜色

VGA 在高分辨率模式下可以使用 16 种颜色,一般情况下能设计出比较好的界面,但有时还希望界面更加友好,色彩更逼真一些,则可以通过修改 VGA 的 DAC(数字模拟变换器)颜色寄存器中的数值,来改变初始设置的 16 种颜色,DAC 颜色寄存器是一个 18 位的寄存器,红、绿、蓝三个分量分别占 6 位(最大即为 63),卡上有 256 个这样的寄存器,分别对应于 256 个色号,在高分辨模式时只能用 16 个,这 16 个寄存器系统设置的初值如表 2-6。

表 2-6 颜色寄存器的三色分量初值表

颜色号	红色量	绿色量	蓝色量	颜色号	红色量	绿色量	蓝色量
0	0	0	0	8	0	0	21
1	0	0	42	9	0	0	63
2	0	42	0	10	0	42	21
3	0	42	42	11	0	42	63
4	42	0	0	12	42	0	21
5	42	0	42	13	42	0	63
6	42	42	0	14	42	42	21
7	42	42	42	15	42	42	63

修改这些设置的初值即可达到修改颜色目的。

按照 2.2 中的介绍,只要将 3C8 赋予索引号(颜色号),然后依次赋予 3C9 以红、绿、蓝三色值(0—63),即可改变该索引号所对应的颜色;而将 3C7 赋予索引号,然后依次读 3C9 并赋给表示红、绿、蓝三个分量的整型变量,则可读出某颜色号的红、绿、蓝三色值。

具体实现的方法可通过调用 VGA BIOS 中断进行,也可通过 VGA 寄存器编程,有些中文 DOS 下只能通过 VGA 寄存器编程。使用 VGA BIOS 中断编程时,只要赋予各入口参数以相应的数值:

```

AH = 10H
AL = 10H
BX = 颜色号
DH = 红色分量
CH = 绿色分量
CL = 蓝色分量

```

然后执行: INT 10H 即可。

下面是用 Turbo C/Borland C 调用 VGA BIOS 中断修改寄存器颜色的函数(其形式参数分别为: 颜色号、红色、绿色、蓝色分量的亮度值):

```
void ChgColor(int ColorNo,int Red,int Green,int Blue)
{
    union REGS In;          /* 定义联合类型          */
    In.h.ah = 0x10;
    In.h.al = 0x10;          /* 子功能号          */
    In.x.bx = ColorNo;        /* 颜色号            */
    In.h.dh = Red;            /* 红色分量          */
    In.h.ch = Green;          /* 绿色分量          */
    In.h.cl = Blue;           /* 蓝色分量          */
    int86(0x10,&In,&In);      /* 调用 BIOS 10H 号中断 */
}
```

如果使用 VGA 寄存器编程,也很简单,并且速度快。但应注意:一定不要将地址写错了,否则,可能会导致严重的后果:

```
void ChgColor(int ColorNo,int Red,int Green,int Blue)
{
    outportb(0x3c8,ColorNo); /* DAC 颜色表写索引寄存器中写入颜色号 */
    outportb(0x3c9,Red);      /* DAC 颜色表数据寄存器中依次写入红、 */
    outportb(0x3c9,Green);    /* 绿、蓝三色亮度值          */
    outportb(0x3c9,Blue);
}
```

需要指出的是,能够同时修改的颜色号不是 0—15,而是 0—6,20,56—63。

Chg16.C 是通过修改 VGA 的 16 种颜色为蓝色亮度不同的演示程序;

Getcolor.c 可得到某一颜色号的红、绿、蓝三色值。

/\*

Chg16.c——改变 VGA 设置的 16 种颜色。

\*/

```
#include "dos.h"
#include "conio.h"
#include "stdlib.h"
#include "graphics.h"
void InitGra(void);
void Quit(void);
void DrBar(void);
void ChgColor(int ColorNo,int Red,int Green,int Blue);
void ResetColor(void);
int main(void)
```

```

{
    InitGra();
    DrBar();
    getch();
    Quit();
    return(0);
}

void InitGra(void)
{
    int GraphMode, DraphDrive = DETECT;
    registerbgidriver(EGAVGA_driver);
    initgraph(&DraphDrive, &GraphMode, "");
}

void Quit(void)
{
    closegraph();
    exit(0);
}

void DrBar(void)
{
    int i, Bl = 10;
    int y = 20;
    char ClNo[16] = {0,1,2,3,4,5,6,20,56,57,58,59,60,61,62,63};
    for (i = 1; i < 6; i++)
    {
        ChgColor(ClNo[i], 0, 0, Bl += 3);
        setfillstyle(1, ClNo[i]);
        bar(120, y, 520, y+35);
        y += 35;
    }
    for (i = 8; i < 16; i++)
    {
        ChgColor(ClNo[i], 0, 0, Bl += 3);
        setfillstyle(1, ClNo[i]);
        bar(120, y, 520, y+35);
        y += 35;
    }
}

void ResetColor(void)

```

```

{
    int i;
    char Red[16] = {0,0,0,0,42,42,42,42,0,0,0,0,42,42,42,42};
    char Grn[16] = {0,0,42,42,0,0,42,42,0,0,42,42,0,0,42,42};
    char Blu[16] = {0,42,0,42,0,42,0,42,21,63,21,63,21,63,21,63};
    for (i = 0; i < 16; i++)
        ChgColor(i, Red[i], Grn[i], Blu[i]);
}

void Color16()
{
    int i, x = 0;
    for (i = 0; i < 16; i++)
    {
        setfillstyle(1, i);
        bar(x, 20, x+40, 460);
        x += 40;
    }
}

/* 通过 VGA BIOS 中断进行修改 */
void ChgColor(int ColorNo, int Red, int Green, int Blue)
{
    union REGS In;
    In.h.ah = 0x10;
    In.h.al = 0x10;          /* 子功能号          */
    In.x.bx = ColorNo;       /* 颜色号          */
    In.h.dh = Red;           /* 红色分量        */
    In.h.ch = Green;         /* 绿色分量        */
    In.h.cl = Blue;          /* 蓝色分量        */
    int86(0x10, &In, &In);  /* 调用 BIOS 10H 号中断 */
}

/* 下面的函数直接修改寄存器, 速度更快 */
void ChgColor(int ColorNo, int Red, int Green, int Blue)
{
    outportb(0x3c8, ColorNo); /* DAC 颜色表写索引寄存器中写入颜色号 */
    outportb(0x3c9, Red);     /* DAC 颜色表数据寄存器中依次写入红、 */
    outportb(0x3c9, Green);    /* 绿、蓝三色亮度值          */
    outportb(0x3c9, Blue);
}

*/

```



```

/*
    Getcolor.c——读出某种颜色的红、绿、蓝三色分量之值
    使用方法：X>Getcolor <颜色号>
*/
#include "dos.h"
#include "stdio.h"
#include "stdlib.h"
void GetColor(int ColorNo,char * R,char * G,char * B);
int main (int argc,char * argv[])
{
    int Colorn;
    char Red,Grn,Blu;
    Colorn = atoi(argv[1]);
    GetColor(Colorn,&Red,&Grn,&Blu);
    printf("No= %d,Red= %d,Green= %d,Blue= %d\n",Colorn,Red,Grn,Blu);
    return 0;
}
void GetColor(int ColorNo,char * R,char * G,char * B)
{
    outport(0x3c7,ColorNo);
    *R = inportb(0x3c9);
    *G = inportb(0x3c9);
    *B = inportb(0x3c9);
}

```

如果使用 VGA BIOS 中断进行读取颜色初值,入口参数为:

AH = 10H

AH = 15H

BX = 颜色号

出口参数:

DH = 红色分量

CH = 绿色分量

CL = 蓝色分量

## 2.5 半色调明暗处理技术

在屏幕界面设计中,为了使图象画面更具有真实感,通常需要模拟任意两种颜色之间的明暗层次以产生实体模型的真实明暗效果。例如,从黑色过渡到蓝色再到深蓝色。可以改变 VGA 设置的 16 种颜色来实现,但在高分辨率模式时,同时能够改变的颜色数目太少,受到一些限

制。半色调明暗处理技术是指在填充中按照指定的填充模式产生不同级别的明暗浓淡色彩。例如,如果对屏幕上每两个像素位置选择其一填充,则会产生 50% 的明暗效果,即颜色饱和度介于黑色和本色之间,依此类推。

半色调明暗处理技术的步骤为:

- (1) 构造半色调矩阵  $8 \times 8$ ;
- (2) 填充实体;
- (3) 使用线条技术除去边界线。

在 Turbo C/Borland C 中,半色调明暗技术的实现可以通过函数 `setfillpattern()` 来实现,其调用格式为:

```
void setfillpattern(char * pattern,\nint color);
```

该函数设置用户定义的填充图模和颜色以填充封闭图形。

其中: `pattern` 是一个指向 8 个字节的指针数组,它定义了填充操作中使用的模式,这 8 个字节定义了  $8 \times 8$  点阵的图形。

每个字节的 8 位二进制数表示水平 8 点,8 个字节表示 8 行,然后以此为模型向整个封闭区域填充。例如,实填充模式可定义为: `{0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF,0XFF}` 50% 填充可定义为: `{0X55,0XAA,0X55,0XAA,0X55,0XAA,0X55,0XAA}` (如图 2-3 所示,图中为清楚起见,0 没有写出)。

下面是 16 种颜色明暗处理的演示例程:

/\*

HafTon.c——半色调明暗处理的演示程序

\*/

```
#include "stdio.h"
```

```
#include "conio.h"
```

```
#include "graphics.h"
```

```
void HafBnd(int Color,int y);
```

```
void InitGra(void);
```

```
char Fill[][8]={0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,/* 0% 填充 */\n                0x00,0x20,0x00,0x00,0x00,0x02,0x00,0x00,/* 3% 填充 */\n                0x20,0x00,0x02,0x00,0x80,0x00,0x08,0x00,/* 6% 填充 */\n                0x20,0x02,0x80,0x08,0x20,0x02,0x80,0x08,/* 12% 填充 */\n                0x44,0x11,0x44,0x11,0x44,0x11,0x44,0x11,/* 25% 填充 */\n                0xAA,0x44,0xAA,0x11,0xAA,0x11,0xAA,0x11,/* 37% 填充 */\n                0x55,0xAA,0x55,0xAA,0x55,0xAA,0x55,0xAA,/* 50% 填充 */\n                0x55,0xBB,0x55,0xEE,0x55,0xBB,0x55,0xEE,/* 62% 填充 */\n                0xBB,0xEE,0xBB,0xEE,0xBB,0xBB,0xBB,0xBB,/* 75% 填充 */}
```

	1		1		1		1	0X55
1		1		1		1		0XAA
	1		1		1		1	0X55
1		1		1		1		0XAA
	1		1		1		1	0X55
1		1		1		1		0XAA
	1		1		1		1	0X55
1		1		1		1		0XAA

图 2-3 填充矩阵

```

        0xDF,0xFD,0x7F,0xF7,0xDF,0xFD,0x7F,0xF7,/* 87% 填充 */
        0xFF,0xDF,0xFF,0xDF,0xFF,0xDF,0xFF,0xFF,/* 93% 填充 */
        0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF};/* 100% 填充 */

int main(void)
{
    int i,y = 16;
    InitGra();
    for (i = 1;i < 16;i++)
    {
        HafBnd(i,y);
        y += 30;
    }
    getch();
    closegraph();
    return 0;
}

void HafBnd(int Color,int y)
{
    int i,x = 0;
    for (i = 1;i < 12;i++)
    {
        setfillpattern(Fill[i],Color);
        bar(x,y,x+58,y+20);
        x += 58;
    }
}

void InitGra(void)
{
    int GraphMode,DrpghDrive = DETECT;
    registerbgidriver(EGAVGA-driver);
    initgraph(&DrpghDrive,&GraphMode,"");
}

```

## 2.6 VGA 256 色编程技术

VGA 具有 256 种颜色能够同时使用的模式 13H。该模式的屏幕分辨率为  $320 \times 200$ ，虽然分辨率降低，但由于同时能够显示的颜色数量增多，所以在编写界面程序时也是经常用到的。模式 13H 时的存储映象如图 2-4。

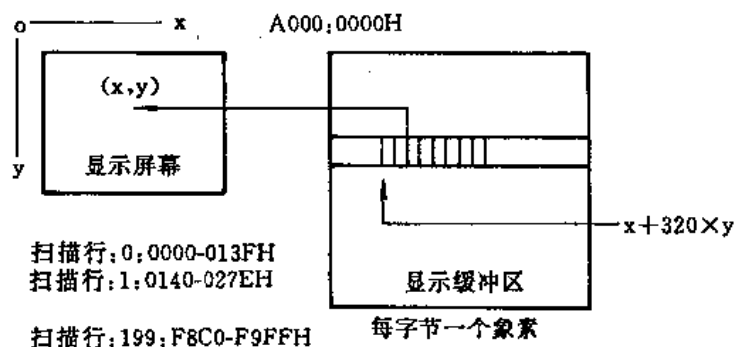


图 2-4 VGA 256 色模式显示存储映象

其基地址为 A000:0000H, 屏幕与显示存储器是线性映射关系, 屏幕上一个象素对应于显示存储器一个字节, 屏幕上象素点 (x,y) 与显示存储器中对应字节的地址关系如下:

字节地址 =  $x + 320 \times y$

其中: x: 0—319(水平), y: 0—199(垂直)

Turbo C/Borland C 没有提供这种模式的图形库函数, 但我们可根据以上的介绍编写出作图最基本的函数——画点函数:

```
void PutPoint(int x,int y,int Color) {    /* 画点函数 */
    char far * p;                        /* 定义远程指针 */
    p = (char far *) (0x0a0000000L);    /* 缓冲区首址赋 P */
    * (x+y * 320+p) = Color;            /* 在屏幕上写点 */
}
```

当然, 我们也可以利用 VGA BIOS 中断编写该模式下的画点函数, 只是调用 BIOS 中断的速度很慢。调用的入口参数为:

AH = 0CH;  
BH = 页面号  
AL = 颜色值  
CX = 横坐标 X  
DX = 纵坐标 Y

再执行: INT 10H。用 Turbo C/Borland C 编写画点函数如下:

```
void PutPoint(int x,int y,int Color)    /* 画点函数 */
{
    union REGS In;                      /* 定义联合类型 */
    In.h.ah = 0x0C;                     /* 子功能号 */
    In.h.al = Color;                     /* 入口参数送寄存器 */
    In.x.cx = x;                         /* X 坐标 */
    In.x.dx = y;                         /* Y 坐标 */
    int88(In);                           /* 调用 BIOS 中断 */
}
```

```

    In.x.dx = y;          /* Y 坐标          */
    In.h.bh = 0;          /* 页号          */
    int86(0x10,&In,&In); /* 执行 10H 中断 */
}

```

然后再根据画点的函数,编写其它的图素函数,VGA256.C 显示出了 VGA 初始设置的 256 种颜色,然后从 256 种颜色中取出 81 种,画出了 81 个矩形。

/\*

VGA256.c—— VGA 256 色编程

\*/

```
#include "dos.h"
```

```
#include "conio.h"
```

```
#include "stdio.h"
```

```
void InitScr();
```

```
void RstScr();
```

```
void PutPoint(int x,int y,int Color);
```

```
void Rect(int x1,int y1,int x2,int y2,int Color);
```

```
void LineV(int x1,int y1,int x2,int y2,int Color);
```

```
int main()
```

```
{
```

```
    int x1,y1,x2,y2,i,j;
```

```
    x1 = y1 = 0;
```

```
    x2 = 319;
```

```
    y2 = 199;
```

```
    InitScr();
```

```
    for (i = 0;i < 256;i++)
```

```
        LineV(i,0,i,199,i);
```

```
    for( i = 18;i < 100;i++)
```

```
        Rect(x1++,y1++,x2--,y2--,i);
```

```
    for( i = 18;i < 50;i++)
```

```
        Rect(x1--,y1--,x2++,y2++,i);
```

```
    getch();
```

```
    RstScr();
```

```
}
```

```
void InitScr()
```

```
{
```

```
    union REGS In;
```

```
    In.x.ax = 0x13;          /* 进入 13H 模式 */
```

```
    int86(0x10,&In,&In);
```

```
}
```

```

void RstScr()
{
    union REGS In;
    In.x.ax = 0x03;      /* 退出 13H 模式 */
    int86(0x10,&In,&In);
}
/* 直接写视频缓冲区 */
void PutPoint(int x,int y,int Color) /* 画点函数 */
{
    char far *p;
    p = (char far *) (0x0a0000000L);
    * (x+y * 320+p) = Color;
}
/* 利用 VGA BIOS 中断在屏幕上画点,速度慢 */
void PutPoint(int x,int y,int Color)
{
    union REGS In;
    In.h.ah = 0x0C;
    In.h.al = Color;
    In.x.cx = x;
    In.x.dx = y;
    In.h.bh = 0;
    int86(0x10,&In,&In);
}
/*
void LineV(int x1,int y1,int x2,int y2,int Color) /* 画一垂直线 */
{
    int i;
    for (i = 0;i < 199;i++)
        PutPoint(x1,i,Color);
}
void Rect(int x1,int y1,int x2,int y2,int Color) /* 画一矩形 */
{
    int i;
    for(i = x1;i <= x2;i++)
    {
        PutPoint(i,y1,Color);
        PutPoint(i,y2,Color);
    }
}

```

```

for(i = y1; i <= y2; i++)
{
    PutPoint(x1,i,Color);
    PutPoint(x2,i,Color);
}
}

```

## 2.7 改变 VGA 设置的 256 种颜色

在 2.2 中已介绍,颜色寄存器共有 256 个,在高分辨率模式下,只能使用其中的 16 种颜色,而在 13H 模式下,则可同时使用 256 种颜色,并且可以按 2.3 中介绍的方法来改变这 256 种颜色,在 2.3 中介绍的函数 ChgColor(),在 13H 模式下同样适应。下面的例程是通过修改 256 种颜色的初值画出的蓝天:

其实现的方法如下:

从上至下画水平线,线的颜色逐渐变化:

蓝色分量从 0 依次增加到最大 63;

然后增加绿色分量,依次从 0 增加到最大 63;

再增加红色分量,依次从 0 增加到最大 63;

所画出的画面下部分呈白色微红,往上是蓝色亮度上的加深,逐渐过渡为深蓝色,正如黎明时的天空。

/\*

CHG256.C——改变 VGA 设置的 256 种颜色,画天空

\*/

```
#include "dos.h"
```

```
#include "conio.h"
```

```
void InitScr(void);
```

```
void RestScr(void);
```

```
void DrSky(void);
```

```
void PutPoint(int x,int y,int Color);
```

```
void ChgColor(int ColorNo,int Red,int Green,int Blue);
```

```
void Line(int x1,int y1,int x2,int y2,int Color);
```

```
int main (void)
```

```
{
```

```
    InitScr();
```

```
    DrSky();
```

```
    getch();
```

```
    RestScr();
```

```
    return(0);
```

```

}
void InitScr(void)
{
    union REGS Regs;
    Regs.x.ax = 0x13;
    int86(0x10,&Regs,&Regs) ;
}
void RestScr(void)
{
    union REGS Regs;
    Regs.x.ax = 0x03;
    int86(0x10,&Regs,&Regs) ;
}

void DrSky(void)
{
    int Red,Green,Blue,i,y = 190;
    Blue = Red = Green = 0;
    for (i = 0;i < 188;i++)
    {
        if (Blue < 63) Blue = Blue+1;
        else if (Green < 63) Green = Green+1;
        else Red = Red+1;
        ChgColor(i,Red,Green,Blue);
    }
    for (i = 188;i > 0;i--)
    {
        Line(10,y,300,y,i);
        y--;
    }
}

void ChgColor(int ColorNo,int Red,int Green,int Blue)
{
    outportb(0x3c8,ColorNo);
    outportb(0x3c9,Red);
    outportb(0x3c9,Green);
    outportb(0x3c9,Blue);
}

void PutPoint(int x,int y,int Color) /* 画点 */

```



```

{
    char far *p ;
    p = (char far *) (0x0a0000000L) ;
    * (x+y*320+p) = Color ;
}
void Line(int x1,int y1,int x2,int y2,int Color) /* 画直线 */
{
    int i ;
    for(i = x1 ;i <= x2 ;i++) PutPoint(i,y1,Color) ;
}

```

## 2.8 颜色与视觉效果——立体块及按钮设计

当今的计算机软件界面越来越漂亮直观,向形象化发展,更接近用户。立体块及按钮的动感效果便是一个方面,它是图形用户接口中一个不可少的界面技术,实现按钮的技术和鼠标的驱动使得图形软件增色很多,交互效果好。

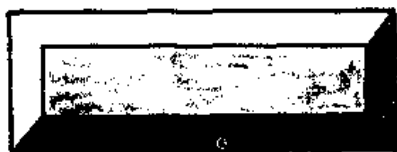


图 2-5 立体按钮

在实现按钮技术中需要了解何以产生逼真的立体感。我们可以设想,一束光从屏幕的左上角向屏幕的右下角照射时,一个凸出的按钮的左边和上边应该反射白光而呈白色,而右边和下边应是深灰色,中间突出的平面应该是浅灰色,此画法在屏幕上的效果即是凸出的感觉;而当按钮被按下时,则左、上边颜色与右、下边对换,此时的效果是凹入的感觉。

交替显示这两种情况,选定一定的延时时间,就会产生明显的动感效果。

立体按钮及动感的设计将在第六章作详细介绍,本部分灵活运用黑、白、淡灰、深灰设计了淡雅而美观的立体块,请参阅 3DBX.C 的运行结果:

```

/*
3DBX.C——画立体块
*/
#include "conio.h"
#include "graphics.h"
void InitGra(void);
void FillScr(void);
void Intfac1(void);
void Intfac2(void);
void Intfac3(void);
void Intfac4(void);
void Box1(int x,int y,int l,int h);

```

```

void Box2(int x,int y,int l,int h);
void Box3(int x,int y,int l,int h);
void Box4(int x,int y,int l,int h);
int main(void)
{
    InitGra();
    FillScr();Intfac1();
    getch();
    FillScr();Intfac2();
    getch();
    FillScr();Intfac3();
    getch();
    FillScr();Intfac4();
    getch();
    closegraph() ;
    return(0);
}
void InitGra(void)
{
    int GraphMode,DrpghDrive = DETECT;
    registerbgidriver(EGAVGA _ driver) ;
    initgraph(&DrpghDrive,&GraphMode,"") ;
}
void FillScr(void)
{
    setfillstyle(1,7);
    bar(0,0,639,479);
}
void Intfac1(void)
{
    int x,y,l,h,i,j;
    y = 5;
    l = 45;h = 25;
    for (j = 0;j < 13;j++)
    {
        x = 6;
        for (i = 0;i < 11;i++)
        {
            Box1(x,y,l,h);

```

```

        x = x+l+13;
    }
    y = y+h+12;
}
}
void Intfac2(void)
{
    int x,y,l,h,i,j;
    y = 5;
    l = 45;h = 25;
    for (j = 0;j < 13;j++)
    {
        x = 6;
        for (i = 0;i < 11;i++)
        {
            Box2(x,y,l,h);
            x = x+l+13;
        }
        y = y+h+12;
    }
}
void Intfac3(void)
{
    int x = 0,y = 16;
    int l = 639,h = 50;
    int i;
    for (i = 0;i < 6;i++)
    {
        Box3(x,y,l,h);
        y = y+h+40;
    }
}
void Intfac4(void)
{
    int x = 0,y = 16;
    int l = 639,h = 50;
    int i;
    for (i = 0;i < 6;i++)
    {

```

```

        Box4(x,y,l,h);
        y = y+h+40;
    }
}

void Box1(int x,int y,int l,int h)
{
    setcolor(15);
    line(x,y,x+l,y);
    line(x,y,x,y+h);
    setcolor(8);
    line(x+l,y+h,x+l,y);
    line(x+l,y+h,x,y+h);
}

void Box2(int x,int y,int l,int h)
{
    setcolor(8);
    line(x,y,x+l,y);
    line(x,y,x,y+h);
    setcolor(15);
    line(x+l,y+h,x+l,y);
    line(x+l,y+h,x,y+h);
}

void Box3(int x,int y,int l,int h)
{
    setcolor(15);
    line(x,y,x+l,y);
    line(x+l,y+1,x+l-1,y+1);
    setcolor(8);
    line(x,y+h,x+l,y+h);
    line(x,y+h-1,x+l,y+h-1);
}

void Box4(int x,int y,int l,int h)
{
    setcolor(8);
    line(x,y,x+l,y);
    line(x+l,y+1,x+l-1,y+1);
    setcolor(15);
    line(x,y+h,x+l,y+h);
    line(x,y+h-1,x+l,y+h-1);
}

```

```
}
```

要设计立体块,最少要有4种颜色。采用16种颜色中的4种(黑、白、淡灰、深灰)进行设计是很方便的,但要设计其它颜色的立体块,就没有另外合适的4种颜色了,这时可以修改初始设置的顏色,以获得3种只在亮度上不同的相近颜色(及1种背景色)。程序B3DBX.C显示的是通过修改VGA设置的顏色寄存器的初值而设计的蓝色立体块,读者可按这种方法举一反三,设计出其它各种颜色的立体块来。

```
/*
```

B3DBX.C——通过修改顏色寄存器之值设计蓝色立体块

```
*/
```

```
#include "dos.h"
```

```
#include "stdio.h"
```

```
#include "process.h"
```

```
#include "conio.h"
```

```
#include "graphics.h"
```

```
void InitGra();
```

```
void Quit();
```

```
void Big3DBox(int x,int y,int l,int h);
```

```
void ChgColor(int ColorNo,int Red,int Green,int Blue);
```

```
int main()
```

```
{
```

```
    InitGra();
```

```
    ChgColor(1,0,60,60); /* 修改VGA初始设置的顏色 */
```

```
    ChgColor(2,0,30,30);
```

```
    ChgColor(3,0,45,45);
```

```
    ChgColor(7,36,36,36);
```

```
    setfillstyle(1,7);
```

```
    bar(0,0,639,479);
```

```
    Big3DBox(140,25,360,430);
```

```
    getch();
```

```
    Quit();
```

```
    return 0;
```

```
}
```

```
void InitGra()
```

```
{
```

```
    int GraphDrive = DETECT,GraphMode;
```

```
    registerbgidriver(EGAVGA __driver);
```

```
    initgraph(&GraphDrive,&GraphMode,"");
```

```
}
```

```
void Quit()
```

```

{
    closegraph();
    exit (0);
}

void Big3DBox(int x,int y,int l,int h)
{
    int i;
    setcolor(2);
    rectangle(x,y,x+l,y+h);
    setfillstyle(1,2);
    bar(x+l,y+l,x+l-1,y+h-1);
    setcolor(1);
    for (i=0;i<6;i++)
    {
        line(x+i,y+i,x+l-i,y+i);
        line(x+i,y+i,x+i,y+h-i);
    }
    setfillstyle(1,3);
    bar(x+6,y+6,x+l-6,y+h-6);
    line(x+l-1,y+h-1,x+l-6,y+h-6);
    setcolor(2);
    line(x+l,y+1,x+6,y+6);
    setcolor(1);
    rectangle(x+20,y+20,x+l-20,y+h-20);
    setcolor(2);
    rectangle(x+19,y+19,x+l-21,y+h-21);
}

void ChgColor(int ColorNo,int Red,int Green,int Blue)
{
    outportb(0x3c8,ColorNo);
    outportb(0x3c9,Red);
    outportb(0x3c9,Green);
    outportb(0x3c9,Blue);
}

```

# 汉字显示技术

## 第 3 章

本章内容：

### 3.1 中文 DOS 下的汉字显示

#### 3.1.1 利用 C 语言的库函数显示彩色汉字

#### 3.1.2 利用 BIOS 显示彩色汉字

### 3.2 西文 DOS 下的汉字显示

#### 3.2.1 点阵汉字

##### 3.2.1.1 16 点阵汉字的显示

##### 3.2.1.2 24 点阵汉字的显示

##### 3.2.1.3 点阵汉字的无级放大、旋转与倾斜显示

##### 3.2.1.4 小字库的建立

#### 3.2.2 矢量汉字

##### 3.2.2.1 矢量汉字的显示

##### 3.2.2.2 矢量汉字的无级放大、旋转与倾斜显示

##### 3.2.2.3 矢量汉字小字库的建立

### 3.3 立体汉字与空心汉字

### 3.4 256 色汉字显示

在中国使用计算机,决定了的计算机软件应当具有汉字提示。本章将介绍在西文 DOS 和中文 DOS 下显示各种汉字的方法。

### 3.1 中文 DOS 下的汉字显示

在汉字操作系统下,屏幕处于图形模式(但未进行图形初始化时,不能直接使用图形函数)。此时,只有标准的文本输入输出函数(如 printf(), puts(), putchar(), gets(), getch()等)可用。也可用 window()函数定义一个文本窗口;gotoxy()函数定位输入、输出汉字的位置;textcolor()设置汉字前景色;textbackground()设置汉字背景色。在中文操作系统下,汉字显示常用的方法有:

- (1) 利用 C 语言的库函数;
- (2) 利用 BIOS 中断;
- (3) 直接写视频缓冲区显示汉字。

较常用的方法是前两种,本书将介绍这两种方法。

#### 3.1.1 利用 C 语言的库函数显示彩色汉字

C 语言的输出函数(printf(), putchar()等)采用 unsigned char 或 int 类型表述字符,采用中断 INT 10H 实现字符显示。只要在图形方式下,汉字在 C 语言程序中可正常运行。有关汉字程序的编制可按下述步骤:

- (1) 在中文环境下利用文字编辑软件编辑带有汉字的 C 源程序;
- (2) 在西文环境下用 Turbo C/Borland C 编译,形成可执行文件;
- (3) 在中文环境下运行可执行文件。

值得指出的是,现在有些汉字系统是不占基本内存的,其显示方式是直接写视频缓冲区,如中国龙,PTDOS,UCDOS 3.0 等。在这些汉字系统下可直接运行 Turbo C/Borland C,这就省去了上述的麻烦。

C 语言的窗口字符串输出函数 cprintf(), cputs()等,在编译时缺省编译成按直接视频缓冲区传送方式显示字符串,因而不能显示汉字。但是 Turbo C/Borland C 在头文件 CONIO.H 中设置了一个变量 directvideo,该变量控制输出方式:当其为 TRUE(或缺省)时,所有的屏幕输出都跳过 DOS 和 BIOS 的屏幕输出例行程序,直接到屏幕 RAM 区;当其为 FALSE 时,屏幕的输出则借助于视频中断 10H(因为中文 DOS 较好地保留并修改了 10H 号中断)。因此,只要在图形或文本方式下设置 directvideo 变量为 FALSE,则 cprintf()和 cputs()等函数就可以很好地显示汉字。

下面是一个用 printf(), cprintf()和 cputs()函数显示彩色汉字的示例程序:

```
/*  
FUNCC.C—— 利用 C 语言的库函数显示彩色汉字  
*/  
#include "dos.h"
```



```

#include "stdio.h"
#include "conio.h"
#define FALSE 0
int main()
{
    clrscr();
    directvideo = FALSE;
    textattr(0x1e);
    printf("您好,欢迎您再来! \n"); /* 显示黑白汉字 */
    cprintf("祝您好运! \n"); /* 显示彩色汉字 */
    textattr(0x16);
    cputs("早上好,下午好,晚上好! \n"); /* 显示彩色汉字 */
    getch();
    return 0;
}

```

### 3.1.2 利用 BIOS 显示彩色汉字

中文 DOS 一般都对显示模块 INT 10H 作了较好的修改,使其具有汉字处理能力。因此,利用 BIOS 10H 号中断的功能调用在屏幕上显示彩色汉字是非常容易的。

BIOS 10H 号中断的 09 号功能显示字符的入口参数为:

AH = 09; 功能号  
 BH = 显示页  
 CX = 输出字符数  
 AL = 输出的字符  
 BL = 输出字符的属性

BIOS 10H 中断的 02 号功能置光标的位置,入口参数如下:

AH = 02; 功能号  
 DH = 行  
 DL = 列  
 BH = 显示页

下面是利用 BIOS 显示彩色汉字的例程:

```

/*
    BIOSCC.C——利用 BIOS 显示彩色汉字
*/
#include "dos.h"
#include "conio.h"
void BIOSCC(int x,int y,char *p,int Tcolor,int Bcolor);
void DispBox(int x1,int y1,int x2,int y2,int Tcolor,int Bcolor);

```

```

int main(void)
{
    window(3,2,40,9);
    DispBox(4,2,40,10,15,1);
    BIOSCC(8,6,"新年好,祝您在新的一年里好运!",15,1);
    getch();
    window(1,1,80,25);
    clrscr();
    return(0);
}

```

/\* 利用 BIOS 显示汉字的函数

x,y:汉字的起点坐标;

p:汉字串;

Tcolor:前景颜色;

Bcolor:背景颜色.

\*/

```

void BIOSCC(int x,int y,char *p,int Tcolor,int Bcolor)

```

```

{
    union REGS r;
    while (*p)
    {
        r.h.ah = 2;
        r.h.dh = y;
        r.h.dl = x++;
        int86(0x10,&r,&r);
        r.h.ah = 9;
        r.h.bh = 0;
        r.x.cx = 1;
        r.h.al = *p++;
        r.h.bl = Tcolor+(Bcolor<<4);
        int86(0x10,&r,&r);
    }
}

```

/\* 画矩形框函数

x1,y1:矩形左上角坐标;

x2,y2:矩形右上角坐标;

Tcolor:线框前景颜色;

Bcolor:线框背景颜色.

```

*/
void DispBox(int x1,int y1,int x2,int y2,int Tcolor,int Bcolor)
{
    char * char[]={ "┌", "┐", "└", "┘", "├", "┤", "┬", "┴" };
    int i;
    for (i = x1; i < x2; i += 2)
    {
        BIOSCC(i,y1,Char[1],Tcolor,Bcolor);
        BIOSCC(i,y2,Char[1],Tcolor,Bcolor);
    }
    for (i = y1; i < y2; i ++ )
    {
        BIOSCC(x1,i,Char[3],Tcolor,Bcolor);
        BIOSCC(x2,i,Char[3],Tcolor,Bcolor);
    }
    BIOSCC(x1,y1,Char[0],Tcolor,Bcolor);
    BIOSCC(x2,y1,Char[2],Tcolor,Bcolor);
    BIOSCC(x2,y2,Char[5],Tcolor,Bcolor);
    BIOSCC(x1,y2,Char[4],Tcolor,Bcolor);
}

```

以上介绍的是屏幕为汉字文本时的输出。当屏幕为图形方式时,这种方法仍然可以使用。下面的 GFUNCC.C,便是加入了图形初始化的函数后,利用 C 语言的库函数显示汉字的:

```

/*
    GFUNCC.C——图形文本方式显示汉字
*/
#include "dos.h"
#include "conio.h"
#include "stdio.h"
#include "graphics.h"
unsigned char s[12];
void InitGra(void);
void Sub();
int main()
{
    char Str[6];
    InitGra();
    directvideo = 0;
    setcolor(14);
    rectangle(15,34,320,181);

```

```

window(3,3,40,10);
textbackground(1);
textcolor(12);
clrscr();
gotoxy(3,2);
cprintf("您贵姓?");
scanf("%s",Str);
gotoxy(3,4);
cprintf("您好,%s 先生!");
Sub();
closegraph();
return 0;
}

void InitGra(void)
{
    int GraphDrive = DETECT,GraphMode;
    registerbgidriver(EGAVGA _driver);
    initgraph(&GraphDrive,&GraphMode,"");
}

void Sub()
{
    char *Str[]={"工作单位:", "职业:"};
    char c,s1[40],s2[10];
    setcolor(12);           /* 设置作图颜色 */
    rectangle(231,140,559,299); /* 图形函数 */
    window(32,9,68,15);    /* 文本窗口函数 */
    textbackground(2);
    textcolor(14);
    clrscr();
    setfillstyle(1,2);
    bar(232,142,558,298);
    gotoxy(1,2);
    cprintf("自我介绍一下好吗?");
    gotoxy(28,2);
    for(;;)
    {
        c = getch();
        if (c == 'Y' || c == 'y')
        {

```

```

        gotoxy(5,4);
        cputs(Str[0]);
        gotoxy(15,4);
        scanf("%s",s1);
        gotoxy(5,6);
        cprintf("%s",Str[1]);
        gotoxy(11,6);
        scanf("%s",s2);
        break;
    }
}
}

```

说明:

(1) 在图形方式下,文本前景或文本背景的颜色与图形前景颜色或图形屏幕背景颜色无关,为了改变颜色,应分别按相应的函数设置。

(2) 在图形方式时,不能用有关图形方式下文本输出的函数输出汉字。即,不能用下面的函数:outtext()和 outtextxy()。

(3) 在图形方式下的数据输入是无光标的,为了建立图形光标可参阅 7.4。

## 3.2 西文 DOS 下的汉字显示

在中文 DOS 下可用 C 语言的库函数和 BIOS 显示汉字,但有时用户不希望在汉字系统下运行程序。因为在汉字操作系统下,存在着以下一些问题:

- 汉字系统本身将占据一些内存,如果应用程序开发得稍大,就可能无法加载;
- 一般的汉字系统是通过调用 BIOS 中断来显示汉字的,显示的速度较慢;
- 在汉字 DOS 下,汉字的显示仅为 16×16 点阵,显示方式单一,并且汉字的行距、字距均不能灵活控制,很难开发出良好的用户界面;
- 各种汉字系统所采取的显示模式不尽相同,要使所开发的软件在各种汉字操作系统下都能正常工作,难度较大。

针对这些问题,本章介绍在西文 DOS 下显示汉字的技术,这些技术在中文 DOS 下同样可以使用,并且显示的汉字可以达到尽善尽美的程度。

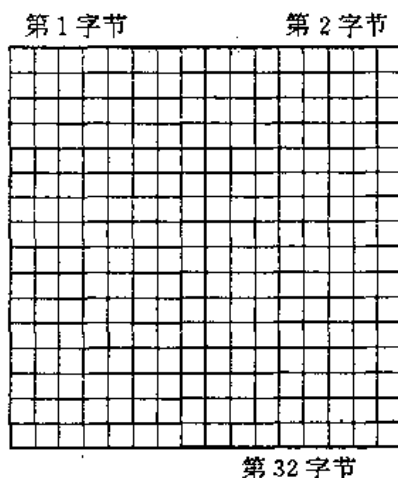
这些技术包括:在西文 DOS 下显示 16×16 点阵汉字、24×24 点阵汉字、矢量汉字;对 24×24 点阵汉字和矢量汉字按任意倍数放大、旋转、倾斜显示;同时可以使汉字在屏幕上任意位置、以各种不同字体、不同颜色显示。

### 3.2.1 点阵汉字

#### 3.2.1.1 16点阵汉字的显示

##### (1) 16点阵汉字字模的存储格式

在 UC DOS 3.0 中,有一个 16×16 点阵的汉字库 HZK16,主要用于屏幕显示。字库中的汉字按 16×16 点阵模式存储,每个汉字由 16×16=256 个点组成,占用 16×2=32 个字节单元。字节每一位表示一个点的属性(1—有亮点,0—无亮点),连续的两个字节表示该汉字字模的一行,32 个字节的排列顺序如图 3-1 所示。



##### (2) 西文 DOS 下显示 16 点阵汉字的实现

计算机是以编码方式处理和使用字符的。对于西文字符采用一个字节表示,(即 ASCII 码)一般只

图 3-1 16 点阵汉字的 32 个字节排列顺序

用 7 位表示 128 个字符,而把最高位作为奇偶校验(或不用)。我国国标规定汉字用内码表示。内码为两个字节。为了保证和西文兼容,汉字系统的内码必须同时允许 ASCII 码和汉字内码的使用,两者之间不应冲突,所以目前规定每个字节只用 7 位,若两个字节的最高位均为 1,则该字符为汉字。

国标对汉字库作了统一规定,将汉字库分成若干个区,每个区有 94 个汉字,每个汉字均有一个确定的区码和位码,知道了区位码也就相当于知道了汉字在字库中的位置。由于汉字的内码与区位码有一定的关系,所以只要通过汉字的内码就可以得到汉字的区位码,从而也就可以获得该汉字的字模。然后在图形方式下,读取字模中的每一个字节的每一位即可按画点或画线的方式在屏幕上显示出汉字(本章只讲述画点方式,为了提高汉字的显示速度,可通过建立小字库实现,见 3.2.1.4)。

设某个汉字的内码位 zzbb,则该汉字在字库中位置(记录号)为:

$$\text{Rec} = (\text{zz} - 161) \times 94 + (\text{bb} - 161)$$

得到记录号后乘以 32 则为该汉字在字库中字模第一个字节的位置,连续读取 32 个字节,就可以得到这个汉字的字模,对每个字节的每一位进行判断,如果为 1 就画点否则就不画,这样就可以显示出汉字来。下面的例程在屏幕上显示 16 点阵黄色汉字,字间距为 2 个像素点,所用字库为 UC DOS 3.0 的 HZK16。

/\*

PUTCC16.C——西文 DOS 下显示 16 点阵汉字

\*/

#include "stdio.h"

#include "stdlib.h"

#include "conio.h"

#include "graphics.h"

```

FILE *fp;
void InitGra(void);
int OpenLIB(void);
void Quit(void);
void PutCC16(int,int,int,int,char * Str);
void ErrMsg();
int main(void)
{
    char * Str = "西文 DOS 下显示 16 点阵汉字";
    InitGra();
    if (! OpenLIB()) ErrMsg();
    PutCC16(0,100,2,YELLOW,Str);
    getch();
    Quit();
    return(0);
}
void InitGra(void)
{
    int GraphDrive = DETECT,GraphMode;
    registerbgidriver(EGAVGA _ driver);
    initgraph(&GraphDrive,&GraphMode,"");
}
int OpenLIB()
{
    if ((fp = fopen("c:\\ucdos\\hzk16","rb")) == NULL) return (0);
    return (1);
}
void Quit(void)
{
    closegraph();
    fcloseall();
    exit(0);
}
/* x,y:显示汉字的起点坐标;
   Wid:字间距;
   Color:汉字的颜色;
   Str:汉字串
*/
void PutCC16(int x,int y,int Wid,int Color,char * Str)

```

```

{
    unsigned Zcode,Bcode;                /* 区码,位码 */
    int i,j,k,Rec;
    long Len;
    char Buf[32];
    while (*Str)                          /* 直到字串显示完 */
    {
        if ((*Str & 0x80) && (*(Str+1) & 0x80)) /* 是汉字 */
        {
            Zcode = (*Str-0xa1) & 0x07f;      /* 区码 */
            Bcode = (*(Str+1)-0xa1) & 0x07f;    /* 位码 */
            Rec = Zcode * 94 + Bcode;          /* 记录号 */
            Len = Rec * 32L;                  /* 在字库中位置 */
            fseek(fp,Len,SEEK-SET);           /* 定位字模首字节 */
            fread (Buf,1,32,fp);              /* 连续读取 32 字节 */
            for (i = 0; i < 16; i++)          /* 字模垂直方向 16 个字节 */
                for (j = 0; j < 2; j++)        /* 水平方向的 2 个字节 */
                    for (k = 0; k < 8; k++)    /* 每个字节的 8 位 */
                        if (Buf[i * 2 + j] >> (7-k) & 1) /* 判断是否为 1 */
                            putpixel(x+j * 8+k,y+i,Color);
            x = x+16+Wid;
            Str += 2;                          /* 指向下一个汉字 */
        }
    }
    return;
}

void ErrMsg()
{
    printf("Open LIB File Error!");
    getch();
    Quit();
}

```

### 3.2.1.2 24 点阵汉字的显示

#### (1) 24 点阵汉字字模的存储结构

24 点阵字库主要用于打印,字库中每个汉字由  $24 \times 24 = 576$  个点组成,占用  $24 \times 3 = 72$  个连续的字节单元,字节的每一位表示一个点的属性(1—有亮点,0—无亮点),连续的 3 个字节表示该汉字字模的一列,这 72 个字节的排列顺序如图 3-2 所示。



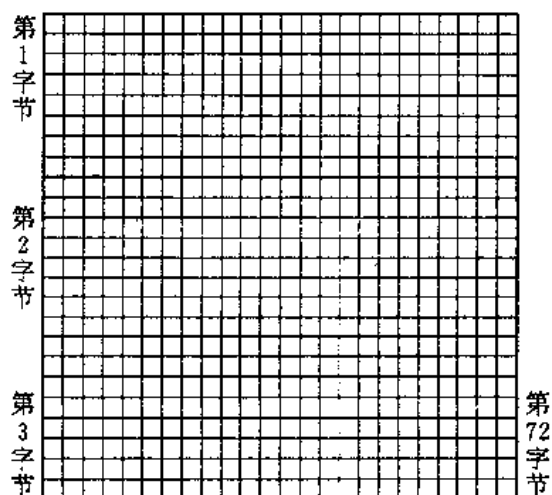


图 3-2 24 点阵汉字的 72 个字节排列顺序

## (2) 西文 DOS 下显示 24 点阵汉字的实现

24 点阵与 16 点阵汉字的内码是一样的,不同的只是每个汉字字模的存储个数。因此,确定 24 点阵汉字记录号仍按 16 点阵的方法,得到记录号后乘以 72 即为汉字字模第一个字节在字库中的位置。连续读取 72 个字节即可得到该字模,再根据每个字节的每一位是否为 1,判断是否在屏幕上画点,这样就可以显示出 24 点阵汉字,下面是一个例子:

/\*

PutCC24.C——图形方式下显示 24 点阵汉字

\*/

#include "stdio.h"

#include "graphics.h"

#include "conio.h"

FILE \*fp;

void InitGra(void);

void Quit();

int OpenLIB(void);

void PutCC24(int,int,int,int,char \* Str);

void ErrMsg();

int main(void)

{

char \*Str = "图形方式下显示 24 点阵汉字";

InitGra();

if (! OpenLIB()) ErrMsg();

PutCC24(0,150,2,YELLOW,Str);

getch();

Quit();

```

    return 0;
}

void InitGra(void)
{
    int GraphMode, DraphDrive = DETECT;
    registerbgidriver(EGAVGA_driver);
    initgraph(&DraphDrive, &GraphMode, "");
}

void Quit()
{
    fcloseall();
    closegraph();
}

int OpenLIB(void)          /* 打开 24 点阵宋体字库 */
{
    if ((fp = fopen("c:\\ucdos\\clib24s", "rb")) == NULL) return 0;
    return 1;
}

/* 显示 24 点阵汉字
   x,y:起点坐标;
   Wid:字间宽度(象素点数);
   Color:汉字的颜色;
   Str:汉字串.
*/
void PutCC24(int x, int y, int Wid, int Color, char * Str)
{
    unsigned Zcode, Bcode;          /* 区码,位码 */
    int i, j, k, Rec;
    long Len;
    char Buf[72];
    while (* Str)                    /* 直到字符串显示完 */
    {
        if ((* Str & 0x80) && (*( Str+1) & 0x80)) /* 是汉字 */
        {
            Zcode = (* Str - 0xa1) & 0x07f;        /* 区码 */
            Bcode = (*( Str+1) - 0xa1) & 0x07f;     /* 位码 */
            Rec = Zcode * 94 + Bcode;               /* 记录号 */
            Len = Rec * 72L;                         /* 在字库中位置 */
            fseek(fp, Len, SEEK_SET);

```

```

        fread (Buf,1,72,fp);                /* 连续读取 72 字节 */
        for (i = 0; i<24; i++)              /* 水平方向的 24 个点 */
            for (j = 0; j<3; j++)            /* 垂直方向的 3 个字节 */
                for (k = 0; k<8; k++)        /* 每个字节的 8 位 */
                    if (Buf[i * 3+j]>>(7-k) & 1)
                        putpixel(x+i,y+j * 8+k,Color);
        x = x+24+Wid;
        Str += 2;
    }
}
return;
}

```

```

void ErrMsg()
{
    printf("Open LIB File Error!");
    getch();
    Quit();
}

```

需要说明的是,24 点阵汉字有不同字体之分,常用的有:宋、楷、仿宋、黑体四种。在 UC-DOS 中,其对应的字库名分别为:CLIB24S,CLIB24K,CLIB24F,CLIB24H。只要在 OpenLIB ()中打开不同字体的字库文件,就可按所选的字体显示汉字:

```

void OpenLIB(int LibNam){
    switch(LibNam)
    {
        case ST:
            fp = fopen("c:\\ucdos\\CLIB24S","rb"); break;
        case KT:
            fp = fopen("c:\\ucdos\\CLIB24K","rb"); break;
        case FS:
            fp = fopen("c:\\ucdos\\CLIB24F","rb"); break;
        case HT:
            fp = fopen("c:\\ucdos\\CLIB24H","rb"); break;
    }
}

```

### 3.2.1.3 点阵汉字的无级放大、旋转与倾斜显示

上面介绍的是按 1:1 的比例显示汉字,但有时为了特殊显示效果,需要对汉字进行放大、

旋转、倾斜显示,并选用不同的字体,本节就将讲述这些技巧。

(1) 要对汉字进行放大,包括水平和垂直两个方向的放大。为了简单可以采用成倍数放大,即将某一汉字水平、垂直各放大 1 倍。相当于将字模中每一位在水平方向变成两点,这样在水平方向 24 点就变成了 48 点;同样,垂直方向放大一倍,相当于字模中每一位在垂直方向变成两点,这样在垂直方向 24 点就变成了 48 点。

下面的程序实现了按任意整数倍数显示放大汉字的程序:

```
/*
PutCCEx.C——图形方式下显示 24 点阵放大汉字
*/
#include "stdio.h"
#include "graphics.h"
#include "conio.h"
#define ST 1
#define KT 2
#define FS 3
#define HT 4
FILE *fp;
void InitGra(void);
int OpenLIB(int);
void ErrMsg();
void Quit();
void PutCCEx(int,int,int,int,int,int,char * Str);
int main(void)
{
    char * Str = "图形方式下显示 24 点阵放大汉字";
    InitGra();
    if (!OpenLIB(ST)) ErrMsg();
    PutCCEx(0,150,2,2,2,YELLOW,Str);
    getch();
    Quit();
    return 0;
}
void InitGra(void)
{
    int GraphMode,DrpghDrive = DETECT;
    registerbgidriver(EGAVGA_driver);
    initgraph(&DrpghDrive,&GraphMode,"");
}
void Quit()
```

```

{
    fcloseall();
    closegraph();
}
int OpenLIB(int LibNam)
{
    switch(LibNam)
    {
        case ST:
            if ((fp = fopen("c:\\ucdos\\CLIB24S","rb")) == NULL) return 0;
            break;
        case KT:
            if ((fp = fopen("c:\\ucdos\\CLIB24K","rb")) == NULL) return 0;
            break;
        case FS:
            if ((fp = fopen("c:\\ucdos\\CLIB24F","rb")) == NULL) return 0;
            break;
        case HT:
            if ((fp = fopen("c:\\ucdos\\CLIB24H","rb")) == NULL) return 0;
            break;
    }
    return 1;
}
void ErrMsg()
{
    printf("Open LIB File Error!");
    getch();
    Quit();
}
/* 放大显示汉字
   x,y:汉字起点坐标;
   Wid:汉字间距(象素点数);
   Xt,Yt:X方向和Y方向的放大倍数;
   Color:汉字的颜色;
   Str:汉字串;
*/
void PutCCEX(int x,int y,int Wid,int Xt,int Yt,int Color,char *Str)
{
    unsigned Zcode,Bcode;
    /* 区码,位码 */

```

```

int Rec,i1,i2,i3,i4,i5;
long Len;
char Buf[72];
while (* Str)                                /* 直到字符串显示完 */
{
    if ((* Str & 0x80) && (* (Str+1) & 0x80)) /* 是汉字 */
    {
        Zcode = (* Str - 0xa1) & 0x07f;      /* 区码 */
        Bcode = (* (Str+1) - 0xa1) & 0x07f;   /* 位码 */
        Rec = Zcode * 94 + Bcode;             /* 记录号 */
        Len = Rec * 72L;                     /* 在字库中位置 */
        fseek(fp,Len,SEEK_SET);
        fread (Buf,1,72,fp);                 /* 72 字节 */
        for (i1 = 0; i1 < 24 * Xt; i1 += Xt)
            for (i4 = 0; i4 < Xt; i4++)
                for (i2 = 0; i2 < 3; i2++)
                    for (i3 = 0; i3 < 8; i3++)
                        if (Buf[i1/Xt * 3 + i2] >> (7 - i3) & 1)
                            for (i5 = 0; i5 < Yt; i5++)
                                putpixel(x+i1+i4,y+i2 * 8 * Yt+i3 * Yt+i5,Color);
        x = x + 24 * Xt + Wid;
        Str += 2;
    }
}
}

```

程序运行后,屏幕上显示出“图形方式下显示 24 点阵汉字”的宋体黄色大字,水平方向和垂直方向分别放大了 2 倍,如果放大倍数太大,则显示出的汉字明显可看到锯齿状,这主要是因为字模的点阵太少。如果采用高点阵字库,如 48 点阵,96 点阵或更高,则显示出的大汉字的质量会明显改善。当然也可以对 24 点阵汉字进行平滑处理,但需要较复杂的过程,并且显示速度会受到影响,在本书中对此不作介绍。对于程序中用到的较大的汉字,可以采用矢量汉字,见 3.2.2。

(2) 汉字的倾斜显示就是将原字模数据乘以一个错切矩阵  $T$

$$T = \begin{pmatrix} 1 & 0 \\ \operatorname{tg}\theta & 1 \end{pmatrix} \quad \text{其中, } \theta \text{ 为倾斜角;}$$

$$(x_1 \ y_1) \times T = (x_1 + y_1 \operatorname{tg}\theta \ y_1),$$

即汉字倾斜一个  $\theta$  角后,坐标由原来的  $(x_1, y_1)$  变成了  $(x_2, y_2)$ :

$$\begin{cases} x_2 = x_1 + y_1 \operatorname{tg}\theta \\ y_2 = y_1 \end{cases}$$

要想将汉字旋转还必须乘以一个旋转矩阵,

$$R = \begin{pmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{pmatrix} \quad \alpha \text{ 为旋转角。}$$

$$(x_2 \ y_2) \times R = (x_2\cos\alpha - y_2\sin\alpha \quad x_2\sin\alpha + y_2\cos\alpha)$$

即汉字旋转一个角  $\alpha$  后,坐标由原来的  $(x_2, y_2)$  变成了  $(x_3, y_3)$ :

$$\begin{cases} x_3 = x_2\cos\alpha - y_2\sin\alpha \\ y_3 = x_2\sin\alpha + y_2\cos\alpha \end{cases}$$

这里要注意的是以上的变换均是相对于坐标原点进行的,而在 C 语言的图形屏幕中,屏幕的左上角是坐标原点。为了不使显示超出屏幕,在程序中应将坐标原点移到某一指定的位置。

还应注意,当放大、旋转、倾斜同时存在时,应先放大,后旋转和倾斜。

下面的程序是按照上述的介绍进行编制的,可以任意对汉字进行放大(如  $1.2 \times 0.8$ )、旋转与倾斜(详细的说明请参看程序):

```
/*
   RCCDSP.C——24 点阵汉字的无级变倍放大、旋转、倾斜显示
*/
#include "stdio.h"
#include "graphics.h"
#include "conio.h"
#include "math.h"
#include "float.h"
#define BB 3.141592/180
FILE *fp;
void InitGra(void);
void OpenLIB(void);
void ErrMsg();
void Quit();
void RCC(int,int,float,float,int,int,int,int,char *S);
int main(void)
{
    char *S = "汉字的无级放大旋转倾斜";
    InitGra();
    OpenLIB();
    RCC(0,150,1.2,0.8,0.5,5,5,YELLOW,S);
    getch();
    Quit();
    return 0;
}
void InitGra(void)
{
```

```

int GraphMode, DraphDrive = DETECT;
registerbgidriver(EGAVGA_driver);
initgraph(&DraphDrive, &GraphMode, "");
}

void Quit()
{
    fcloseall();
    closegraph();
}

void OpenLIB(void)
{
    if ((fp = fopen("c:\\ucdos\\clib24s", "rb")) == NULL) ErrMsg();
}

/*
    x, y: 显示汉字的起点坐标;
    Xt, Yt: X 和 Y 方向的放大系数(任意实数)
    z: 字间距;
    A1: 旋转角;
    A2: 倾斜角;
    Cl: 汉字的颜色;
    S: 汉字串;
*/

void RCC(int x, int y, float Xt, float Yt, int z, int A1, int A2, int Cl, char *S)
{
    unsigned Zcode, Bcode; /* 区码, 位码 */
    int i, j, k, Rec, x4, y4;
    float x1, y1, x2, y2, x3, y3;
    float Alf, Bat, n = -1.0;
    long Len;
    char Buf[72];
    Alf = -A1 * BB;
    Bat = A2 * BB;
    while (*S) /* 直到字串显示完 */
    {
        if ((*S & 0x80) && (*(S+1) & 0x80)) /* 是汉字 */
        {
            Zcode = (*S - 0xa1) & 0x07f; /* 区码 */
            Bcode = (*(S+1) - 0xa1) & 0x07f; /* 位码 */
            n++;
            Rec = Zcode * 94 + Bcode; /* 记录号 */
            Len = Rec * 72L; /* 在字库中位置

```



```

fseek(fp,Len,SEEK__SET);
fread (Buf,1,72,fp);          /* 72 字节 */
for (i = 0; i < 24; i++)
    for (j = 0; j < 3; j++)
        for (k = 0; k < 8; k++)
            if (Buf[i * 3+j] >> (7-k) &.1) {
                x1 = Xt * (i+24 * n);
                y1 = Yt * (j * 8+k);          /* 放大 */
                y2 = y1;
                x2 = x1+y1 * tan(Bat);        /* 倾斜 */
                x3 = x2 * cos(Alf)-y2 * sin(Alf);
                y3 = x2 * sin(Alf)+y2 * cos(Alf); /* 旋转 */
                x4 = x3+x;
                y4 = y3+y;                    /* 平移 */
                putpixel(x4,y4,C1);
            }
        x = x+24+z;
        S += 2;
    }
}
return;
}
void ErrMsg()
{
    printf("Open LIB File Error!");
    getch();
    Quit();
}

```

程序运行后屏幕上显示出“汉字的无级放大旋转倾斜”的宋体黄色大字,水平方向放大了 1.2 倍,垂直方向放大了 0.8 倍,并分别倾斜和旋转 5°。

#### 3.2.1.4 小字库的建立

本书提供的显示汉字的方法是通过画点来实现的,也可以采用其它方法(如通过定义画线方式,用画线来显示汉字)。影响显示汉字速度的主要原因,是读取汉字库中的字模占用了较多的时间。因此采用建立小字库,然后再从所建立的小字库中读出字模并显示汉字,可明显提高汉字的显示速度。

##### (1) 建立 16 点阵小字库

对于 16 点阵汉字,可以用 CR16.C 建立一个小字库 SML16.DAT,采用的方法是根据汉字串中的汉字在字库中的位置,读出汉字字模,然后将该字模依次写入小字库。对建立的小字

库,可用 RDC16.C 查看一下是否正确(小字库的长度应是  $32 \times$  汉字数),在以后使用该小字库时,也用该程序读取:

```
/*
    Cr16.C——建立 16 点阵小字库
*/
#include "stdio.h"
#include "stdlib.h"
#include "graphics.h"
FILE *fp1, *fp2;
void InitGra(void);
void OpenLIB(void);
void ErrMsg();
void Quit(void);
void Crt16LIB(int x,int y,int Color,char *Str);
int main(void)
{
    char *Str = "小字库文件";
    InitGra();
    OpenLIB();
    Crt16LIB(0,100,YELLOW,Str);
    getch();
    Quit();
    return(0);
}
void InitGra(void)
{
    int GraphDrive = DETECT,GraphMode;
    registerbgidriver(EGAVGA _driver);
    initgraph(&GraphDrive,&GraphMode,"");
}
void OpenLIB(void)
{
    if ((fp1 = fopen("c:\\ucdos\\hzk16","rb")) == NULL) ErrMsg();
    fp2 = fopen("SML16.dat","wb");
}
void Quit(void)
{
    closegraph();
    fcloseall();
}
```

```

    exit(0);
}
/* 建立小字库函数,边写小字库边显示汉字
   x,y:显示汉字的起点坐标;
   Color:汉字的前景色;
   Str:要建立小字库的汉字串.
*/
void Crt16LIB(int x,int y,int Color,char * Str)
{
    unsigned Code1,Code2;           /* 区码,位码 */
    int i,j,k,Rec;
    long Len;
    char Buf[32];
    while (* Str)                   /* 直到字符串显示完 */
    {
        if ((* Str & 0x80) && (*( Str+1) & 0x80)) /* 是汉字 */
        {
            Code1 = (* Str-0xa1) & 0x07f;        /* 区码 */
            Code2 = (*( Str+1)-0xa1) & 0x07f;      /* 位码 */
            Rec = Code1 * 94 + Code2;              /* 记录号 */
            Len = Rec * 32L;                       /* 在字库中位置 */
            fseek(fp1,Len,SEEK _SET);
            fread (Buf,1,32,fp1);                  /* 32 字节 */
            fwrite(Buf,1,32,fp2);
            for (i = 0; i < 16; i++)                /* 边建立边显示 */
                for (j = 0; j < 2; j++)
                    for (k = 0; k < 8; k++)
                        if (Buf[i * 2 + j] >> (7-k) & 1)
                            putpixel(x+j * 8+k,y+i,Color);
            x += 16;
            Str += 2;
        }
    }
    return;
}

void ErrMsg()
{
    printf("Open LIB File Error!");
    getch();
}

```

```

    Quit();
}
/*
    RDC16.C —— 读出已建立的 16 点阵小字库
*/
#include "stdio.h"
#include "stdlib.h"
#include "conio.h"
#include "graphics.h"
#include "process.h"
FILE    * fp ;
void Rd16LIB(int,int,int,int,int);
void InitGra(void);
void OpenLIB(void);
void Quit(void);
int main(void)
{
    InitGra();
    OpenLIB();
    Rd16LIB(0,100,YELLOW,0,5);
    getch();
    Quit();
    return(0);
}
/* 读取建立的 16 点阵小字库并显示
   x,y:起点坐标
   Color:汉字的颜色
   St:汉字在小字库中的起始位置(第一个汉字的位置为 0);
   Ed:汉字在小字库中的终止位置;
*/
void Rd16LIB(int x,int y,int Color,int St,int Ed)
{
    int m,i,j,k;
    char Buf[32] ;
    for (m = St; m < Ed; m++)
    {
        fseek(fp,(long) (m * 32),SEEK _SET) ;
        fread(Buf,1,32,fp) ;
        for (i = 0; i < 16; i++)

```

```

        for (j = 0; j < 2; j++)
            for (k = 0; k < 8; k++)
                if (Buf[i * 2 + j] >> (7 - k) & 1)
                    putpixel(x + j * 8 + k, y + i, Color);
        x += 16;
    }
}

void InitGra(void)
{
    int GraphDrive = DETECT, GraphMode;
    registerbgidriver(EGAVGA_driver);
    initgraph(&GraphDrive, &GraphMode, "");
}

void OpenLIB(void)
{
    fp = fopen("sml16.dat", "rb");
}

void Quit(void)
{
    closegraph();
    fcloseall();
    exit(0);
}

```

## (2) 建立 24 点阵小字库

对于 24 点阵汉字, 可以用 CR24.C 建立一个小字库 SML24.DAT, 其建立的方法同 Crt16.C 是一样的, 只是读出和写入的字节是以 72 字节为单位的, 可用 RD24.C 查看一下所建立的字库是否正确(小字库的长度应是 72×汉字数):

```

/*
    CR24.C——建立 24 点阵小字库
*/
#include "stdio.h"
#include "graphics.h"
#include "conio.h"
FILE * fp1, * fp2;
void InitGra(void);
void OpenLIB(void);
void Quit();
void ErrMsg();
void Crt24LIB(int, int, int, int, char * Str);

```

```

int main(void)
{
    char * Str = "小字库文件";
    InitGra();
    OpenLIB();
    Crt24LIB(0,150,0,YELLOW,Str);
    getch();
    Quit();
    return 0;
}

void InitGra(void)
{
    int GraphMode,DrpghDrive = DETECT;
    registerbgidriver(EGAVGA _driver);
    initgraph(&DrpghDrive,&GraphMode,"");
}

void Quit()
{
    fcloseall();
    closegraph();
}

void OpenLIB(void)
{
    if ((fp1 = fopen("c:\\ucdos\\clib24s","rb")) == NULL) ErrMsg();
    fp2 = fopen("SML24.dat","wb");
}

/* 建立小字库函数,边写小字库边显示汉字
   x,y:显示汉字的起点坐标;
   Wid:汉字间距;
   Color:汉字的前景色;
   Str:要建立小字库的汉字串.
*/

void Crt24LIB(int x,int y,int Wid,int Color,char * Str)
{
    unsigned Zcode,Bcode;          /* 区码,位码 */
    int i,j,k,Rec;
    long Len;
    char Buf[72];
    while (* Str)                  /* 直到字符串显示完 */

```

```

{
    if (( * Str & 0x80) && ( * (Str+1) & 0x80))        /* 是汉字 */
    {
        Zcode = ( * Str - 0xa1) & 0x07f;                /* 区码 */
        Bcode = ( * (Str+1) - 0xa1) & 0x07f;            /* 位码 */
        Rec = Zcode * 94 + Bcode;                        /* 记录号 */
        Len = Rec * 72L;                                /* 在字库中位置 */
        fseek(fp1, Len, SEEK__SET);
        fread (Buf, 1, 72, fp1);                        /* 72 字节 */
        fwrite(Buf, 1, 72, fp2);
        for (i = 0; i < 24; i++)
            for (j = 0; j < 3; j++)
                for (k = 0; k < 8; k++)
                    if (Buf[i * 3 + j] >>> (7-k) & 1)
                        putpixel(x+i, y+j * 8+k, Color);
        x = x + 24 + Wid;
        Str += 2;
    }
}

return;
}

void ErrMsg()
{
    printf("Open LIB File Error!");
    getch();
    Quit();
}

/*
    RD24.C—— 用于显示建立的 24 * 24 点阵汉字小字库, 以检查其正确性
*/
#include "stdio.h"
#include "graphics.h"
#include "conio.h"
#include "process.h"
FILE * fp;
void InitGra(void);
void RdSml24(int x, int y, int Color, int St, int Ed);
int main()
{

```

```

InitGra();
fp = fopen("sml24.dat","rb");
RdSml24(0,100,YELLOW,0,5);
getch();
closegraph();
fcloseall();
exit (0);
}
void InitGra(void)
{
    int GraphMode,DraphDrive = DETECT;
    registerbgidriver(EGAVGA _driver);
    initgraph(&DraphDrive,&GraphMode,"");
}
/* 读取建立的 24 点阵小字库并显示
   x,y:起点坐标
   Color:汉字的颜色
   St:汉字在小字库中的起始位置(第一个汉字的位置为 0);
   Ed:汉字在小字库中的终止位置;
*/
void RdSml24(int x,int y,int Color,int St,int Ed)
{
    int m,i,j,k,Rec;
    long Len;
    char Buf[72];
    for (m = St; m < Ed; m++)
    {
        fseek(fp,(long) (m * 72),SEEK _SET);
        fread(Buf,1,72,fp);
        for (i = 0; i < 24; i++)
            for (j = 0; j < 3; j++)
                for (k = 0; k < 8; k++)
                    if (Buf[i * 3+j] >> (7-k) & 1)
                        putpixel(x+i,y+j * 8+k,Color);
        x = x+24;
    }
    return;
}

```



### 3.2.2 矢量汉字

上面介绍的高点阵汉字在用户界面中确实增色不少,但当放大倍数较大时,就明显可见锯齿状,还是美中不足。本节介绍利用华光电子出版系统中的矢量字库进行汉字显示,可以做任意倍数的放大,非常漂亮,并且显示速度相当快。

#### 3.2.2.1 矢量汉字的显示

##### (1) 华光矢量汉字库存储结构

矢量汉字与点阵汉字的最大区别在于:点阵汉字库,记录的是汉字的字模信息,每个汉字的字模长度都相同,而矢量汉字库记录的是汉字笔划信息,汉字越复杂,记录的信息就越多。根据笔划点的信息,在屏幕上画封闭图形并填充写出汉字来。

华光电子出版系统的矢量汉字字库共有四种字体:宋体、黑体、仿宋体和楷体,每种字体中共有汉字 6 763 个。与 GB2312-80 一致,其对应的字库文件分别为:SSL.SLP,HTL.SLP,FSL.SLP 和 KTL.SLP。在字库文件中,存储的汉字是从第 16 区到 87 区,每区 94 个汉字,其中,第 55 区的最后 5 个,本是无字的,文件中以 0 填补。

在每个字库文件的开头处均有一个索引表,包含了 6768 个表项,一个表项对应一个汉字,其中包含了每个汉字的轮廓数据长度及在字库文件中的位置信息,索引表项从第 16 区开始,按每区 94 个字的顺序编排的,因此表项数为:  $(87-15) \times 94 = 6\,768$  个。

每个表项有四个字节,可由下述结构定义:

```
typedef struct {
    unsigned char Length;
    unsigned char Pos1;
    unsigned int Pos2;
} SlpItm;
```

其中第一个字节 Length 是代表以 4 个字节为一个单位的某汉字轮廓数据的长度,后三个字节表示汉字轮廓数据在字库文件中的位置:

Pos1 是高位字节,Pos2 是后 2 个字节的无符号整数。

假设汉字的机内码 2 个字节分别为:H 和 L,则其区码 Z 和位码 B 分别为:

$$Z = H - 160; \quad B = L - 160;$$

由汉字区码 Z 和位码 B,查找汉字索引表项位置的方法如下:

记录号  $Rec = ((Z - 16) \times 94 + B - 1) \times 4;$

根据以上位置,用 fseek 函数在库文件中定位,读出某汉字的 SlpItm 项,则某汉字轮廓数据的存放位置为:

$$DatPos = ((long)SlpItm.Pos1 \ll 16) + (long)SlpItm.Pos2;$$

汉字的轮廓,一般由一条或多条直线组成,多条轮廓线是顺序存放的,每条轮廓线坐标数据的前面,都有一个字节代表本轮廓线中包括的坐标点数,然后是 x,y 坐标值,x 坐标值在前,y 坐标值在后,x,y 各占一个字节,当遇到轮廓线数据超过索引表项中表示的长度或坐标点数为 0 时,表示本汉字数据结束。

对于黑体、宋体和仿宋体 3 种字体,是笔划轮廓,即每一笔划有一条轮廓线,而楷体字,则是整体轮廓线,即相连在一起的笔划为同一轮廓。在进行填充时,应分别区分这两种方式。

## (2) 矢量汉字的显示及无级缩放

矢量汉字的最大优点是可以进行无级缩小和放大,只要指定其 X 和 Y 方向的大小(像素点数) $X_s, Y_s$ ,即可以按比例画出汉字来。也可以通过改变不同的  $X_s, Y_s$  实现汉字的扁、长等显示效果,华光矢量汉字字库的原始汉字点阵是  $96 \times 96$ ,故可以由下式对  $x, y$  坐标作比例变换:

$$Xscale = (\text{float}) X_s/96;$$
$$Yscale = (\text{float}) Y_s/96;$$
$$xt = x * Xscale;$$
$$yt = y * Yscale;$$

这样就实现了汉字在  $x, y$  两个方向的任意放大与缩小。应当注意,矢量汉字的大小不应小于 18,太小时容易模糊。要显示小字时可采用点阵汉字,见 3.2.1。

下面是显示矢量汉字的程序:

```
/*
PutSlp.c——华光矢量汉字库中汉字的显示
*/
#include "slp.h"
void InitGra();
void Quit();
void PutSlpCC(int,int,int,int,int,int,int,int,char * Str,int);
int OpenSlp(int);
void ErrMsg();
int main()
{
    InitGra();
    setbkcolor(0);
    cleardevice();
    if (! OpenSlp(ST)) ErrMsg();
    PutSlpCC(46,80,0,90,96,0,14,4,"欢迎莅临指导",ST);
    if (! OpenSlp(KT)) ErrMsg();
    PutSlpCC(15,290,0,76,90,0,14,12,"请您提出宝贵意见",KT);
    getch();
    Quit();
    return 0;
}
void InitGra()
{
    int GraphDrive = DETECT,GraphMode;
```

```

    registerbgidriver(EGAVGA _driver);
    initgraph(&GraphDrive,&GraphMode,"");
}
void Quit()
{
    closegraph();
    fcloseall();
    exit (0);
}
/*  x0,y0:    汉字左下角的坐标;
   z0:       汉字间距(像素点);
   Xs,Ys:    X和Y方向的大小(像素点);
   BkColor:  屏幕背景色;
   BdColor:  汉字边框色;
   FColor:   汉字的颜色;
   Str:      要显示的汉字串;
   Font:     汉字的字体;
*/
void PutSlpCC(x0,y0,z0,Xs,Ys,BkColor,BdColor,FColor,Str,Font)
int x0,y0,z0,Xs,Ys,BkColor,BdColor,FColor,Font;
char * Str;
{
    SlpItm Hi;
    unsigned char Zcode,Bcode,Tl;
    long int Pos;
    float xt,yt,Xscale,Yscale;
    char Coord[1024];
    int xy[256],Length,Rec,i,j,PointNum,Nt;
    switch(Font)
    {
        case ST:fp = fpS; break;
        case KT:fp = fpK; break;
        case HT:fp = fpH; break;
        case FS:fp = fpF; break;
    }
    Xscale = (float) Xs/96;
    Yscale = (float) Ys/96;
    setcolor(BdColor);
    while (* Str)

```

/\* 矢量汉字是以 96 点阵作为基础的 \*/

```

{
    Zcode = *Str++;          /* 区码 */
    Bcode = *Str++;          /* 位码 */
    Rec = (Zcode-176)*94+Bcode-161; /* 计算汉字索引表项在字库中的位置 */
    fseek(fp,(long)Rec*4,SEEK_SET); /* 定位表项 */
    fread(&Hi,4,1,fp);        /* 读出索引表项 */
    Length = Hi.CdLth*4;
    Pos = ((long)Hi.Pos1<<16)+Hi.Pos2; /* 确定汉字数据在字库中的位置 */
    fseek(fp,Pos,SEEK_SET);
    for (i = 0; i < Length; i++) Coord[i] =getc(fp); /* 读汉字数据 */
    i = 0;
    while (i < Length-1)
    {
        Tl = Coord[i];
        if (Tl == 0) break;
        PointNum = Tl;          /* 笔划多边形顶点数 */
        Tl *= 2;
        for (j = 0; j < Tl; j += 2)
        {
            xt = Xscale * Coord[++i]; /* 笔划顶点坐标 */
            yt = Yscale * Coord[++i];
            xy[j] = x0+(int)(xt+0.5);
            xy[j+1] = y0+(int)(yt+0.5);
        }
        setfillstyle(1,FColor);
        if (Font == KT)          /* 楷体汉字处理 */
        {
            Nt = 0;
            for (j = 0; j < PointNum; j++)
                if (getpixel(xy[2*j],xy[2*j+1]) == FColor) Nt++;
            if ((float)Nt/PointNum > 0.9) setfillstyle(1,BkColor);
        }
        i++;
        fillpoly(PointNum,xy);    /* 写汉字 */
    }
    x0 = x0+z0+(int)(Xscale*96);
}
}

int OpenSlp(int Font)

```

```

{
    switch(Font)
    {
        case ST:
            if ((fpS = fopen("d:\\slp\\ssl.slp","rb")) == NULL) return (0);
            break;
        case KT:
            if ((fpK = fopen("d:\\slp\\ktl.slp","rb")) == NULL) return (0);
            break;
        case HT:
            if ((fpH = fopen("d:\\slp\\htl.slp","rb")) == NULL) return (0);
            break;
        case FS:
            if ((fpF = fopen("d:\\slp\\fsl.slp","rb")) == NULL) return (0);
            break;
    }
    return (1);
}

```

void ErrMsg()

```

{
    printf("Open Slp File Error!");
    getch();
    Quit();
}

```

其头文件 SLP.H 的内容如下:

```

/*
    PutSlp.C 和 RSlpDsp.C 的头文件 slp.h
*/
#include "stdio.h"
#include "conio.h"
#include "process.h"
#include "graphics.h"
#include "math.h"
#include "float.h"
#define ST 1
#define FS 2
#define HT 3
#define KT 4
#define BB 3.14159265/180

```

```
typedef struct {
    unsigned char CdLth;
    unsigned char Pos1;
    unsigned int Pos2;
} SlpItm;
FILE *fp, *fpS, *fpK, *fpH, *fpF;
```

### 3.2.2.2 矢量汉字的无级放大、旋转与倾斜显示

同点阵汉字一样,矢量汉字也可以进行旋转、倾斜显示。只要在画线和填充时,将所对应的点乘以一个旋转矩阵 T 和错切矩阵 R(见 3.2.1)即可实现汉字的旋转与倾斜。下面的程序 RSLPDSP.C 可以将矢量汉字进行任意放大、旋转与倾斜显示:

```
/*
    RSlpDsp.C——华光矢量汉字库中汉字的放大、倾斜及旋转显示
*/
#include "slp.h"
void InitGra();
void Quit();
void PutSlpCC(int,int,int,int,int,int,int,int,int,int,char * Str,int);
int OpenSlp(int);
void ErrMsg();
int main()
{
    InitGra();
    setbkcolor(0);
    cleardevice();
    if (! OpenSlp(ST)) ErrMsg();
    PutSlpCC(46,80,0,5,-5,90,96,0,14,4,"欢迎莅临指导",ST);
    if (! OpenSlp(KT)) ErrMsg();
    PutSlpCC(15,290,0,5,-5,76,90,0,14,12,"请您提出宝贵意见",KT);
    getch();
    Quit();
    return 0;
}
void InitGra()
{
    int GraphDrive = DETECT,GraphMode;
    registerbgidriver(EGAVGA __driver);
    initgraph(&GraphDrive,&GraphMode,"");
}
```

```

void Quit()
{
    closegraph();
    fcloseall();
    exit (0);
}
/*  x0,y0:    汉字左下角的坐标;
    z0:      汉字间距(像素点数);
    Xs,Ys:   X 和 Y 方向的大小(像素点数);
    Af1:     旋转角(度);
    Af2:     倾斜角(度);
    Xsize:   x 方向大小;
    Ysize:   y 方向大小;
    BkColor: 屏幕背景色;
    BdColor: 汉字边框色;
    FColor: 汉字的颜色;
    Str:     要显示的汉字串;
    Font:    汉字的字体;
*/
void PutSlpCC(x0,y0,z0,Af1,Af2,Xsize,Ysize,BkColor,BdColor,FColor,Str,Font)
int x0,y0,z0,Af1,Af2,Xsize,Ysize,BkColor,BdColor,FColor,Font;
char * Str;
{
    SlpItm Hi;
    unsigned char Code1,Code2,Tl;
    long int Pos;
    float xt,yt,Xscale,Yscale,Thta,Sina,Cosa ;
    char Coord[1024];
    int xy[256],Length,Rec,i,j,PointNum;
    switch(Font)
    {
        case ST:fp = fpS; break;
        case KT:fp = fpK; break;
        case HT:fp = fpH; break;
        case FS:fp = fpF; break;
    }
    Xscale = (float) Xsize/96;
    Yscale = (float) Ysize/96;
    Thta = Af1 * BB;

```

```

Sina = sin(Af2 * BB);
Cosa = cos(Af2 * BB);
setcolor(BdColor);
while (* Str)
{
    Code1 = * Str++;
    Code2 = * Str++;
    Rec = (Code1-176) * 94 + Code2-161;
    fseek(fp, (long)Rec * 4, SEEK _ SET);
    fread(&Hi, 4, 1, fp);
    Length = Hi. CdLth * 4;
    Pos = ((long)Hi. Pos1 << 16) + Hi. Pos2;
    fseek(fp, Pos, SEEK _ SET);
    for (i = 0; i < Length; i++) Coord[i] =getc(fp);
    i = 0;
    while (i < Length-1)
    {
        Tl = Coord[i];
        if (Tl == 0) break;
        PointNum = Tl;
        Tl *= 2;
        for (j = 0; j < Tl; j += 2)
        {
            xt = Xscale * Coord[++i];
            yt = Yscale * Coord[++i];
            xt = xt + yt * tan(Thta);          /* 旋转变换 */
            xt = xt * Cosa - yt * Sina;       /* 倾斜变换 */
            yt = xt * Sina + yt * Cosa;
            xy[j] = x0 + (int)(xt + 0.5);
            xy[j+1] = y0 + (int)(yt + 0.5);
        }
        setfillstyle(1, FColor);
        if (Font == KT)
        {
            int Nt = 0;
            for (j = 0; j < PointNum; j++)
            if (getpixel(xy[2 * j], xy[2 * j+1]) == FColor) Nt++;
            if ((float)Nt/PointNum > 0.9) setfillstyle(1, BkColor);
        }
    }
}

```



```

        i++;
        fillpoly(PointNum,xy);
    }
    x0 = x0+z0+(int)(Xscale * 96);
}
}
int OpenSlp(int Font)
{
    switch(Font)
    {
        case ST:
            if ((fpS = fopen("d:\\slp\\ssl.slp","rb")) == NULL) return (0);
            break;
        case KT:
            if ((fpK = fopen("d:\\slp\\ktl.slp","rb")) == NULL) return (0);
            break;
        case HT:
            if ((fpH = fopen("d:\\slp\\htl.slp","rb")) == NULL) return (0);
            break;
        case FS:
            if ((fpF = fopen("d:\\slp\\fsl.slp","rb")) == NULL) return (0);
            break;
    }
    return (1);
}
void ErrMsg()
{
    printf("Open Slp File Error!");
    getch();
    Quit();
}

```

### 3.2.2.3 矢量汉字小字库的建立

矢量汉字漂亮美观,显示速度快,但它的最大缺点是文件太大,如 SSL.SLP 为 1.85 兆字节,HTL.SLP 为 2.25 兆字节。在一般的应用中,矢量汉字只是用在显示标题或特殊效果之处,不需要很多汉字。在应用软件中,如果仅为了显示几个汉字而带上一个庞大的矢量字库是非常没有必要的。同点阵汉字一样,我们也可以按 3.2.1.4 介绍的方法,建立小字库,只存放需要的几个汉字的矢量信息,当需要显示时,直接从小字库中读取信息即可。小字库的建立方法是根据汉字在矢量字库中的位置,读出汉字的矢量信息(点坐标数据),依次写入小字库,小

字库中没有记录表项,记录的是字体、汉字的轮廓数据的长度和坐标数据信息。CRSLP.C 是建立矢量小字库的源程序:

```
/*
    CrSlp.c——建立华光矢量小字库
*/
#include "slp.h"
FILE *fp1,*fp2;
void Quit(void);
void CrSlp(char *Str,int Font);
int OpenSlp(int);
void ErrMsg();
int main(void)
{
    fp2 = fopen("smslp.s","wb");
    if (!OpenSlp(ST)) ErrMsg();
    CrSlp("欢迎莅临指导",ST);
    Quit();
    return(0);
}
void Quit(void)
{
    fcloseall();
    exit(0);
}
/*
    Str: 要建立小字库的汉字;
    Font: 汉字字体
*/
void CrSlp(char *Str,int Font)
{
    SlpItm Hi;
    unsigned char Zcode,Bcode;
    long int Pos;
    char Coord[1024];
    int xy[256],Length,Rec,i;
    switch(Font)
    {
        case ST:fp1 = fpS; break;
        case KT:fp1 = fpK; break;
```

```

    case HT:fp1 = fpH; break;
    case FS:fp1 = fpF; break;
}
putc(Font,fp2);          /* 汉字字体写入小字库 */
while (*Str)
{
    Zcode = *Str++;
    Bcode = *Str++;
    Rec = (Zcode-176)*94+Bcode-161;
    fseek(fp1,(long)Rec*4,SEEK_SET);
    fread(&Hi,4,1,fp1);
    Length = Hi.CdLth*4;
    putc(Hi.CdLth,fp2);    /* 汉字轮廓数据的长度写入小字库 */
    Pos = ((long)Hi.Pos1<<16)+Hi.Pos2;
    fseek(fp1,Pos,SEEK_SET);
    for (i = 0; i<Length; i++)
    {
        Coord[i] = getc(fp1);
        putc(Coord[i],fp2); /* 坐标点数据信息写入小字库 */
    }
}
}

int OpenSlp(int Font)
{
    switch(Font)
    {
        case ST:
            if ((fpS = fopen("d:\\slp\\ssl.slp","rb")) == NULL) return (0);
            break;
        case KT:
            if ((fpK = fopen("d:\\slp\\ktl.slp","rb")) == NULL) return (0);
            break;
        case HT:
            if ((fpH = fopen("d:\\slp\\htl.slp","rb")) == NULL) return (0);
            break;
        case FS:
            if ((fpF = fopen("d:\\slp\\fsl.slp","rb")) == NULL) return (0);
            break;
    }
}

```

```

        return (1);
    }
void ErrMsg()
{
    printf("Open Slp File Error!");
    getch();
    Quit();
}

    需要显示时可用 RDSL.P.C;
/*
    RdSlp.c —— 显示已建立的矢量小字库中的汉字
*/
#include "slp.h"
FILE *fp;
void InitGra();
void Quit();
void PutSlpCC(int,int,int,int,int,int,int,FILE *fp);
void OpenSlp();
int main()
{
    OpenSlp();
    InitGra();
    setbkcolor(7);
    cleardevice();
    PutSlpCC(20,80,100,110,9,12,12,fp);
    getch();
    Quit();
    return 0;
}
void InitGra()
{
    int GraphDrive = DETECT,GraphMode;
    registerbgidriver(EGAVGA_driver);
    initgraph(&GraphDrive,&GraphMode,"");
}
void Quit()
{
    closegraph();
    fcloseall();

```

```

    exit (0);
}
/* 显示已建立的小字库中的汉字
   x0,y0:起点坐标;
   Xs,Ys:X方向和Y方向的大小;
   BkCl:背景颜色;
   BdCl:边框颜色;
   FCl:汉字前景颜色;
   fp:小字库文件名.
*/
void PutSlpCC(x0,y0,Xs,Ys,BkCl,BdCl,FCl,fp)
int x0,y0,Xs,Ys,BkCl,BdCl,FCl;
FILE *fp;
{
    unsigned char Tl;
    float xt,yt,Xscale,Yscale;
    char Coord[1024];
    int xy[256],Length,i,j,PointNum,Font;
    Xscale = (float)Xs/96;
    Yscale = (float)Ys/96;
    setcolor(BdCl);
    Font = getc(fp);
    while (!feof(fp))
    {
        Length = 4 * getc(fp);
        for (i = 0; i < Length; i++)
            Coord[i] = getc(fp);
        i = 0;
        while (i < Length-1)
        {
            Tl = Coord[i];
            if (Tl == 0) break;
            PointNum = Tl;
            Tl *= 2;
            for (j = 0; j < Tl; j += 2)
            {
                xt = Xscale * Coord[++i];
                yt = Yscale * Coord[++i];
                xy[j] = x0 + (int)(xt+0.5);
            }
        }
    }
}

```

```

        xy[j+1] = y0+(int)(yt+0.5);
    }
    setfillstyle(1,FCl);
    if (Font == KT)
    {
        int Nt = 0;
        for (j = 0; j < PointNum; j++)
            if (getpixel(xy[2*j],xy[2*j+1]) == FCl) Nt++;
        if ((float)Nt/PointNum>0.9) setfillstyle(1,BkCl);
    }
    i++;
    fillpoly(PointNum,xy);
}
x0 = x0+(int)(Xscale*96);
}
}
void OpenSlp()
{
    fp = fopen("smslp.s", "rb");
}

```

### 3.3 立体汉字与空心汉字

根据上述介绍的汉字显示方法,我们很容易地进行一些特殊效果的汉字显示,如立体汉字与空心汉字。

立体汉字可按下面方式来实现:

先以一种深色显示出汉字;然后分别在 X 方向和 Y 方向左移和向上移一定量的象素点(一般为汉字大小的 1/10),再以一种浅颜色显示出同样大小的字,例如对 24 点阵汉字,可用下面的例程段来实现:

```

PutCC24(x,y,2,YELLOW,Str);
PutCC24(x-2,y-2,2,YELLOW,Str);

```

对于空心字,则可以按下述方法简单地实现:

以某一坐标点(x,y)作为中心点,上下左右各相差一个象素点的位置,分别在屏幕上再把此汉字以同一种颜色显示出来,最后再在(x,y)处以另外一种颜色再显示一次,这样就只剩下一个外轮廓,即空心汉字。

```

PutCC24(x+2,y,0,YELLOW,Str);
PutCC24(x-2,y,0,YELLOW,Str);
PutCC24(x,y+2,0,YELLOW,Str);

```

```
PutCC24(x,y-2,0,YELLOW,Str);
PutCC24(x,y,0,YELLOW,Str);
```

### 3.4 256 色汉字显示

在当今的许多电视画面中,汉字的显示采用了多种技术,比如同一个汉字,往往采用不同的颜色显示,或者从上到下,或者从左到右,颜色由深而浅或由一种颜色逐渐过渡到另一种颜色,看上去非常漂亮,在本节中我们将介绍这种技术。我们曾在 2.6 中介绍了 VGA 在 13H 模式下同时显示 256 种颜色的方法,在此结合西文 DOS 下的汉字显示技术,来进行编制 256 色汉字程序。在西文 DOS 下显示汉字的关键是读出字模后,根据字模信息画点,在图形方式下可用 Turbo C/Borland C 提供的画点函数 putpixel() 进行画点,在 13H 模式下就不能再用该函数,而要用 2.6 中编制的 PutPoint() 函数:

```
void PutPoint(int x,int y,int Color)    /* 画点函数 */
{
    char far *p;
    p = (char far *) (0x0a00000000L);
    * (x+y*320+p) = Color;
}
```

只要将要显示的汉字字模读出,再根据每一位的信息(1 或 0),采用函数 PutPoint() 在屏幕上画点,即可写出汉字。在显示汉字时,如果每显示一行(一列)像素点后变换一种颜色,再显示下一行(列)像素点,则显示完汉字后的汉字颜色即为汉字的点阵数(24 点阵为 24 种颜色),通过行(列)之间不同的颜色组合,就可以显示出各种各样颜色的汉字。CC256.C 采用 24 种颜色显示了“感谢您使用本书”的 24 点阵汉字,颜色的变化是按行进行的,每一行与下一行像素点的颜色由下式确定:

$$\text{Color} = y + j * 8 + k + c;$$

其中,y 为要显示汉字的左上角纵坐标;

j 为字模垂直方向的 3 个字节中的一个;

k 为每个字节的 8 位中的一位;

c 是一个常数,改变 y+c 之值可以显示不同颜色组合的汉字。

/\*

CC256.C——256 色汉字显示程序

\*/

```
#include "dos.h"
```

```
#include "stdio.h"
```

```
#include "conio.h"
```

```
FILE *fp;
```

```
void OpenLIB(void);
```

```
void CC256(int,int,int,char *Str);
```

```

void ErrMsg();
void InitScr();
void RstScr();
void PutPoint(int x,int y,int Color);
void Quit();
int main(void)
{
    char *Str = "感谢您使用本书";
    OpenLIB();
    InitScr();
    CC256(70,80,2,Str);
    getch();
    Quit();
    return 0;
}
void InitScr()
{
    union REGS In;
    In.x.ax = 0x13;                /* 进入 13H 模式 */
    int86(0x10,&In,&In);
}
void RstScr()
{
    union REGS In;
    In.x.ax = 0x03;                /* 退出 13H 模式 */
    int86(0x10,&In,&In);
}
void OpenLIB(void)                /* 打开 24 点阵宋体字库 */
{
    if ((fp = fopen("c:\\ucdos\\clib24s","rb")) == NULL) ErrMsg();
}
void CC256(int x,int y,int Wid,char *Str)
{
    unsigned Zcode,Bcode;          /* 区码,位码 */
    int i,j,k,Rec,Color;
    long Len;
    char Buf[72];
    while (*Str)                    /* 直到字符串显示完 */
    {

```



```

if (( * Str & 0x80) && ( * (Str+1) & 0x80)) /* 是汉字 */
{
    Zcode = ( * Str - 0xa1) & 0x07f; /* 区码 */
    Bcode = ( * (Str+1) - 0xa1) & 0x07f; /* 位码 */
    Rec = Zcode * 94 + Bcode; /* 记录号 */
    Len = Rec * 72L; /* 在字库中位置 */
    fseek(fp, Len, SEEK_SET);
    fread (Buf, 1, 72, fp); /* 72 字节 */
    for (i = 0; i < 24; i++)
        for (j = 0; j < 3; j++)
            for (k = 0; k < 8; k++)
                if (Buf[i * 3 + j] >> (7 - k) & 1)
                {
                    Color = y + j * 8 + k - 46;
                    PutPoint(x + i, y + j * 8 + k, Color);
                }
    x = x + 24 + Wid;
    Str += 2;
}
}
return;
}
/* 直接写视频缓冲区 */
void PutPoint(int x, int y, int Color) /* 画点函数 */
{
    char far *p;
    p = (char far *) (0x0a0000000L);
    * (x + y * 320 + p) = Color;
}
void Quit()
{
    RstScr();
    fcloseall();
}
void ErrMsg()
{
    printf("Open LIB File Error!");
    getch();
    Quit();
}

```

# 窗口程序设计

## 第 4 章

本章内容：

4.1 文本窗口

4.2 图形窗口

4.3 弹出式窗口系统

4.3.1 弹出式窗口

4.3.2 弹出式窗口的改进

窗口在用户界面设计中是经常用到的。例如在菜单设计中,将各个菜单项显示于一个窗口中,供用户选择;在计算过程中,将计算结果显示在一个窗口内;再如,当用户需要帮助时,也可以将帮助信息显示在一个窗口之中。本章将介绍文本窗口,图形窗口,以及弹出式高级窗口的设计。

## 4.1 文本窗口

Turbo C/Borland C 默认的文本窗口为整个屏幕,共有 80 列,25 行的文本单元,每个单元包括一个字符和一个属性,字符即 ASCII 码字符,属性规定该字符的颜色和强度。Turbo C/Borland C 可以定义屏幕上的一个矩形区域作为窗口,使用 `window()` 函数定义,窗口定义之后,用有关窗口的输入输出函数就可以只在此窗口内进行操作而不超出窗口的边界。

(1) `window()` 函数调用格式如下:

```
void window(int x1,int y1,int x2,int y2);
```

函数原型在 `conio.h` 中,该函数在屏幕上开一个以左上角(`x1,y1`),以右下角(`x2,y2`)为矩形区域的窗口,坐标是相对整个屏幕而言的。Turbo C/Borland C 规定整个屏幕的左上角坐标为(1,1),右下角坐标为(80,25),水平方向为 X 轴,垂直方向为 Y 轴。如果坐标值超过了屏幕坐标的界限,则窗口的定义就失去了意义,不再起作用,但程序编译时并不提示出错。有关文本窗口颜色的设置可参阅第二章。

(2) 窗口内文本的输出函数

```
int cprintf("<格式化字符串>",<变量表>);
```

```
int cputs(char *string);
```

```
int putchar(int ch);
```

`cprintf()` 函数输出一个格式化的字符串或数值到窗口中。它与 `printf()` 函数的用法完全一样,区别在于 `cprintf()` 函数的输出受到窗口的限制,而 `printf()` 函数的输出为整个屏幕。

`cputs()` 函数输出一个字符串到屏幕上,它与 `puts()` 函数用法完全一样,只是受窗口大小限制。

`putchar()` 函数输出一个字符到窗口内。

说明:

使用以上几个函数,当输出超出窗口的右边界时会自动转到下一行的开始处继续输出,当窗口填满内容时仍没有结束输出时,窗口屏幕将会自动逐行上卷直到输出结束为止。

(3) 窗口内文本的输入函数

```
int getche(void);
```

该函数从键盘上获得一个字符,在屏幕上显示的时候,如果字符超过了窗口右边界,则会自动转到下一行的开始位置。

利用 `window()` 函数可以方便地开一个窗口,但其功能有限,比如,希望开一个窗口,可以带单线或双线的边框线,还可以带立体阴影等。为此,本节建立了一个 `TWind()` 函数,可以实现这些功能,其源码如下:

```
void Wind(int x1,int y1,int x2,int y2,int FrmTp,int IsBk,int BCl,int TCl,\int BgCl)
```

```

int i;
int c[2][6]={{0xda,0xc4,0xbf,0xb3,0xc0,0xd9},/* 单线字符 ASCII 码 */
             {0xc9,0xcd,0xbb,0xba,0xc8,0xbc}}; /* 双线字符 ASCII 码 */
textcolor(TCl);
textbackground(BCl);
window(x1,y1,x2,y2);
clrscr();
if (FrmTp) /* 有边框线 */
{
    window(1,1,80,25);
    gotoxy(x1,y1);
    putch(c[FrmTp-1][0]);
    for (i = x1+1;i < x2;i++)
        putch(c[FrmTp-1][1]);
    putch(c[FrmTp-1][2]);
    for (i = y1+1;i < y2;i++)
    {
        gotoxy(x1,i);
        putch(c[FrmTp-1][3]);
        gotoxy(x2,i);
        putch(c[FrmTp-1][3]);
    }
    gotoxy(x1,y2);
    putch(c[FrmTp-1][4]);
    for (i = x1+1;i < x2;i++)
        putch(c[FrmTp-1][1]);
    putch(c[FrmTp-1][5]);
}
if (IsBk) /* 有阴影背景 */
{
    textcolor(BgCl);
    textbackground(0);
    for (i = y1+1;i < y2+1;i++)
    {
        gotoxy(x2+1,i);
        putch('/xb1'); /* 阴影块 */
    }
}

```

```

    for (i = x1+1; i < x2+2; i++)
    {
        gotoxy(i,y2+1);
        putchar('/xb1');
    }
}
window(x1+1,y1+1,x2-1,y2-1);
}

```

函数 TWind() 中各入口参数说明如下:

x1,y1: 窗口左上角坐标;

x2,y2: 窗口右下角坐标;

FrmTp: 窗口边框类型: 0—无边框; 1—单线边框; 2—双线边框;

IsBk: 是否带阴影背景;

BcI: 字符颜色;

TcI: 背景颜色;

BgCl: 阴影背景的颜色。

## 4.2 图形窗口

像文本方式下可以定义屏幕窗口一样, 图形方式下也可以在屏幕的某一区域设定窗口, 只是设定的窗口为图形窗口而已, 其后的有关图形操作都将以这个窗口的左上角(0,0)作为坐标原点, 而且可以通过设置使窗口之外的区域为不可接触。这样, 所有的图形操作就被限定在窗口内进行。

```
void far setviewport(int x1,int y1,int x2,int y2,int clipflag);
```

设定一个以坐标点(x1,y1)为左上角, 以(x2,y2)为右下角的图形窗口, 其中 x1,y1,x2,y2 是相对于整个屏幕坐标的。若 clipflag 为非 0, 则设定的图形窗口以外的部分不可接触, 若 clipflag 为 0, 则图形窗口以外可以接触。

```
void far clearviewport(void);
```

清除现行图形窗口内的内容。

```
void far getviewsettings(struct viewporttype far *viewport);
```

获得关于现行窗口的信息, 并将其存于 viewporttype 定义的结构变量 viewport 中, 其中 viewporttype 的结构说明如下:

```

struct viewporttype {
    int left,top,right,bottom;
    int clipflag;
}

```

```
void far bar(int x1,int y1,int x2,int y2);
```

确定一个以(x1,y1)为左上角, (x2,y2)为右下角的矩形窗口, 再按规定图模和颜色填充。

此函数不画出边框,即以填充色为边框。

```
void far bar3d(int x1,int y1,int x2,int y2,int depth,int topflag);
```

当 topflag 为非 0 时,画出一个三维的长方体,长方体第三维的方向不随任何参数而改变,即始终为 45° 的方向。

同文本方式一样,为了方便地使用,我们也可以编制一个综合的函数 GWind(), 可以带单线或双线的边框线,还可以带立体阴影。其源码如下:

```
void GWind(int x1,int y1,int x2,int y2,int FrmTp,int IsBk,int BCl,int TCl,int BgCl)
{
    int i;
    setfillstyle(1,BCl);
    bar(x1,y1,x2,y2);
    setcolor(TCl);
    if (FrmTp) /* 有边框线 */
    {
        rectangle(x1+2,y1+2,x2-2,y2-2); /* 单线 */
        if (FrmTp == 2) rectangle(x1+4,y1+4,x2-4,y2-4); /* 双线 */
    }
    if (IsBk) /* 有阴影背景 */
    {
        setfillstyle(1,BgCl);
        bar(x1+7,y2+1,x2+7,y2+8);
        bar(x2+1,y1+7,x2+8,y2+8);
    }
}
```

函数 GWind() 中:

x1,y1,x2,y2 分别为窗口左上角,右下角的坐标点;  
FrmTp: 边框类型: 0—不带边框, 1—单线边框, 2—双线边框;  
IsBk: 是否带阴影背景. 0—不带, 非 0—带阴影背景;  
BCl: 窗口背景颜色;  
TCl: 窗口文字颜色;  
BgCl: 背景阴影的颜色。

## 4.3 弹出式窗口系统

### 4.3.1 弹出式窗口

弹出式窗口是几乎所有计算机软件界面的基础。这种窗口直观,醒目,符合人的视觉习惯,可以把帮助信息,警告信息显示在一个窗口内,用户很容易接受它,觉得程序看得见,使抽象的软件形象化,具体化,用户会感觉到软件很亲切,这就实现了美化界面的目的,使你的软件趋于

完美。

弹出一个窗口,首先要给出窗口的大小,这可以有两种方式:一是给出两个对角坐标;二是给出一个角的坐标,再给出大小,这样才能使全部窗口有个共同的位置基准,不至于产生混乱。

知道窗口所占屏幕区域后,要把这一区域内即将覆盖的信息存于内存。这样,在删除窗口时,能够恢复原屏幕的全部信息。存完要被覆盖的信息后,就可以画出你的窗口,进行一系列所需的操作。

要删除窗口时,首先就是恢复窗口覆盖的屏幕区。然后是一些细节如颜色的设置等的恢复。在同一个屏幕上由于需要,可能要开许多窗口,我们用栈这种数据结构来存储一系列的窗口。栈的特点是插入和删除只能在一端进行,即后进先出(LIFO)或先进后出(FILO)。

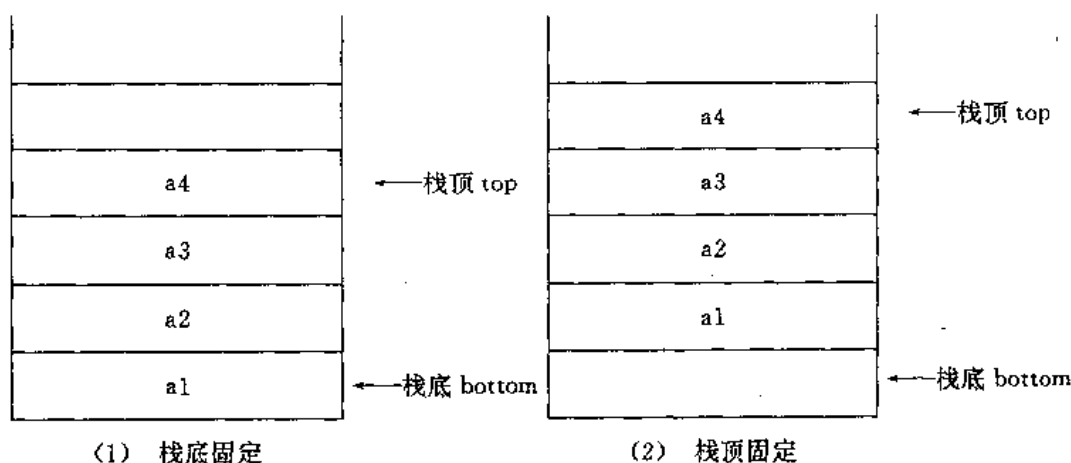


图 4-1 栈的工作方式

栈有两种工作方式,一种如图 4-1(1)所示:栈底是固定的,栈顶随入栈元素的多少而变化;另一种工作方式象弹匣一样,栈顶是固定不变的,当压入元素后,元素依次向栈底移动,如图 4-1(2)所示。后一方式移动很多,效率低,本章采用前一种方式。

栈的操作有很多,典型的有入栈和出栈。栈的溢出也有上溢和下溢。上溢是指当栈已经满了再压入元素时,无法压入。下溢是指当栈已经空了,如果再删除栈中元素。

本章的窗口系统也是用一种结构指针栈来存储多个窗口,栈中的每一个元素都是一个能表示窗口的结构。定义这样的结构有多种可能,因为,一个窗口的表达除了基本的因素,如位置大小,存储指针外,还可以同时考虑前景色,背景色,边框线型与颜色等。本章定义的窗口结构如下:

```
typedef struct WINDOW {          /* 定义窗口的数据类型 */
    int x1,y1,x2,y2;             /* 窗口的左上角及右下角坐标 */
    int Cx,Cy;                   /* 窗口内当前光标的位置 */
    char * Sav;                  /* 存储将被窗口覆盖部分的信息 */
} WIND;
```

Cx,Cy 是窗口中的光标位置。在图形方式时屏幕是无光标的,此处的(Cx,Cy)是相当于光标的点;

Sav 是个字符型指针, 用来存储将被窗口覆盖的屏幕信息, 以便在窗口删除后屏幕能恢复如初。

在程序中定义了整型变量作为窗口栈指针:

```
int Gwd;
```

在弹出任何窗口之前要对栈指针初始化, 即栈指针要赋 0 值。在每次调用开窗口的函数时, 将前一窗口的相应特性值压入栈中。即在第一次开窗口时, 把初始化的屏幕存入了栈中, 窗口的栈底元素 Swd[0], 永远是弹出所有窗口前的整个屏幕。如果栈的大小定义为 10, 则可以弹出 10 个窗口, 但实际压入窗口栈的窗口只有 9 个, 因为第一个栈元素被整个屏幕占用, 即当前窗口是不被压入栈的。如图 4-2 及图 4-3 所示, 弹出两个窗口时, 整屏和窗口 WIN1 分别被压入栈中, 栈顶指针为 2, 但 WIN2 已分配了内存位并未被压入栈中。因为当前窗口有各种操作, 如写一些信息等, 只有在弹出下一窗口时, 把当前窗口压入栈中, 这才保存了完整的被窗口覆盖的信息, 即把对窗口的各种操作保存下来。

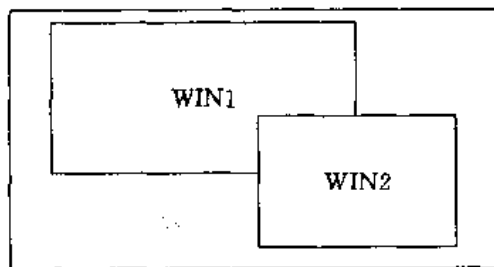


图 4-2

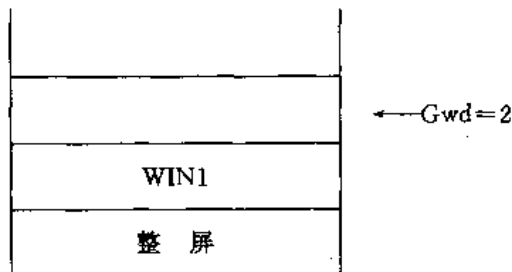


图 4-3

在删除窗口时, 首先把栈指针减 1。如图 4-4 和图 4-5 所示, 此时, 有两个窗口, 要删除窗口 WIN2, 首先要把栈指针减 1, 即  $Gwd = 1$ 。

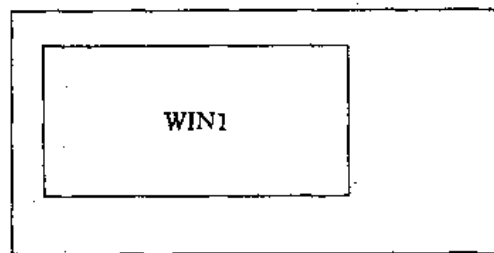


图 4-4

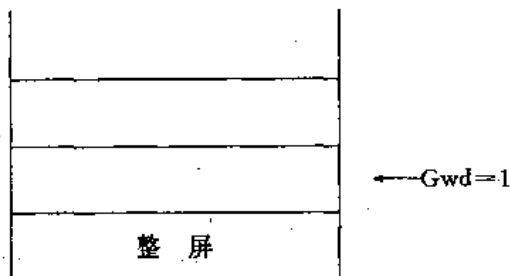


图 4-5

下面是完整的弹出式窗口的源程序。首先弹出四个窗口, 然后删除最后一个窗口, 再将剩余的窗口删去最后一个, 最后删除所有剩余的窗口。程序中已经作了注释:

```
/*  
    POPWIN.C——弹出式窗口  
*/
```



```

#include "graphics.h"
#include "alloc.h"
#include "conio.h"
#define NUM 10                                /* 定义窗口的最大数 */
typedef struct WINDOW {                      /* 定义窗口的数据类型 */
    int x1,y1,x2,y2;                         /* 窗口的左上角及右下角坐标 */
    int Cx,Cy;                               /* 窗口内当前光标的位置 */
    char * Sav;                              /* 存储将被窗口覆盖部分的信息 */
    int Color;                               /* 窗口前景色 */
} WIND;
WIND * Swd[NUM];                            /* 定义窗口栈 */
int SavWind(int x1,int y1,int x2,int y2);
int PopWd(int,int,int,int,int,int,int,int);
int DelWd();
void DelAll();
void InitGra(void);
void Quit();
int Gwd = 0;                                /* 窗口栈指针 */
int main()
{
    InitGra();                               /* 初始化屏幕为图形方式 */
    PopWd(10,10,100,100,0,YELLOW,1,BLUE); /* 弹出第一个窗口 */
    getch();
    PopWd(80,80,200,200,0,YELLOW,1,GREEN); /* 弹出第二个窗口 */
    getch();
    PopWd(150,150,300,300,0,YELLOW,1,RED); /* 弹出第三个窗口 */
    getch();
    PopWd(260,260,400,400,0,WHITE,1,YELLOW); /* 弹出第四个窗口 */
    getch();
    DelWd();                                 /* 删除最后一个窗口 */
    getch();
    DelWd();                                 /* 删除剩余的最后一个窗口 */
    getch();
    DelAll();                               /* 删除剩余的所有窗口 */
    getch();
    Quit();
    return 0;
}
int SavWind(int x1,int y1,int x2,int y2) /* 保存原有的窗口信息 */

```

```

{
    Swd[Gwd]->Sav = (char *) malloc(imagesize(x1,y1,x2,y2));
    if (Swd[Gwd]->Sav == NULL) return 0; /* 为窗口分配内存 */
    getimage(x1,y1,x2,y2,Swd[Gwd]->Sav);
    return (1);
}
/* 弹出一个窗口
    x1,y1,x2,y2:      窗口的屏幕坐标;
    BdTp,BdCl:        边框的线型与颜色;
    BkFil,FilCl:       窗口的填充方式及颜色。
*/
int PopWd(int x1,int y1,int x2,int y2,int BdTp,int BdCl,int BkFil,int FilCl)
{
    struct viewporttype OldV;
    int Oldx,Oldy,SavCl;
    if (Gwd >= NUM) return 0; /* 判断栈是否已满 */
    if ((Swd[Gwd] = (struct WINDOW *)malloc(sizeof(struct WINDOW))) ==
        NULL) return 0;
    getviewsettings(&OldV); /* 保存当前视区设置 */
    Oldx = getx();
    Oldy = gety();
    setviewport(0,0,639,479,1);
    /* 设置整屏为当前视区,以保证正确地存储窗口区 */
    SavCl = getcolor();
    if (SavWind(x1,y1,x2,y2) == 0)
    {
        setviewport(OldV.left,OldV.top,OldV.right,OldV.bottom,1);
        moveto(Oldx,Oldy);
        free(Swd[Gwd]);
        return 0;
    }
    setcolor(BdCl);
    setlinestyle(BdTp,0,NORM _ WIDTH);
    setfillstyle(BkFil,FilCl);
    bar3d(x1,y1,x2,y2,0,0); /* 画出当前新窗口 */
    setviewport(x1,y1,x2,y2,1);
    Swd[Gwd]->x1 = OldV.left; /* 保存新弹出的窗口信息 */
    Swd[Gwd]->y1 = OldV.top;
    Swd[Gwd]->x2 = OldV.right;
}

```

```

    Swd[Gwd]->y2 = OldV.bottom;
    Swd[Gwd]->Cx = Oldx;
    Swd[Gwd]->Cy = Oldy;
    Swd[Gwd]->Color = SavCl;
    Gwd++;
    return 1;
}

int DelWd()                                /* 删除一个窗口 */
{
    struct WINDOW *Ptr;
    if (Gwd <= 0) return 0;                /* 窗口是否为空 */
    Gwd--;                                  /* 栈顶指针减1 */
    Ptr = Swd[Gwd];
    putimage(0,0,Ptr->Sav,COPY_PUT);        /* 恢复被覆盖的屏幕信息 */
    setviewport(Ptr->x1,Ptr->y1,Ptr->x2,Ptr->y2,1);
    moveto(Ptr->Cx,Ptr->Cy);
    setcolor(Ptr->Color);
    free(Ptr->Sav);                          /* 释放内存 */
    free(Swd[Gwd]);
    return 1;
}

void DelAll()                              /* 删除所有的窗口 */
{
    while(DelWd());
}

void InitGra(void)                         /* 初始化屏幕为图形方式 */
{
    int GraphDrive = DETECT,GraphMode;
    registerbgidriver(EGAVGA_driver);
    initgraph(&GraphDrive,&GraphMode,"");
}

void Quit()
{
    closegraph();
    exit(0);
}

```

### 4.3.2 弹出式窗口的改进

4.3.1 中介绍了一个弹出式窗口系统,由于采用了栈的数据结构,使其灵活性受到一定的限制,本节将其改进,使窗口系统更具有灵活性,主要实现以下几个功能:

- 把任意窗口弹到当前窗口;
- 删除任意窗口;
- 删除所有窗口;
- 向任意方向移动窗口;
- 在屏幕中心产生一个窗口。

为了实现以上功能,我们定义一个新的屏幕窗口结构,用以记录屏幕窗口的变化情况:

```
typedef struct Scr{  
    int x0; /* 当前窗口标志 */  
    int Buf; /* 当前弹出的窗口数 */  
    int Lst[MAX]; /* 用以存储所有的窗口标志,并按窗口在屏幕上的顺序排列 */  
} SWIND;
```

窗口栈结构的定义为:

```
typedef struct Win{ /* 定义窗口的数据类型 */  
    int x1,y1,x2,y2; /* 窗口的左上角及右下角坐标 */  
    int Cx,Cy; /* 窗口内当前光标的位置 */  
    char *WdBuf; /* 存储窗口内容 */  
    int Color; /* 窗口前景色 */  
} WIND;
```

如图 4-6 所示,先产生 5 个窗口:

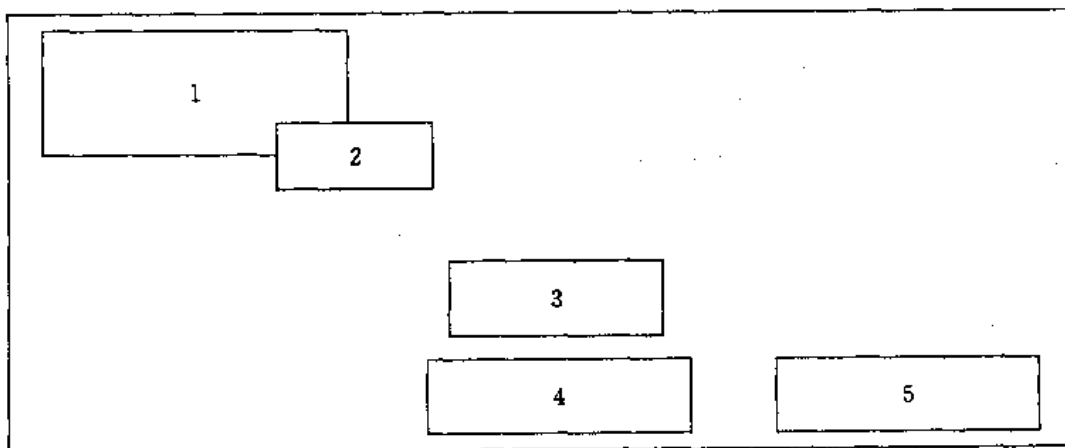


图 4-6 窗口示意图

此时:

SWIND.x0 = 5;

SWIND.Buf = 5;

SWIND.Lst[] = {0,1,2,3,4,5,6,7,8,9,10};

如果把窗口 1 弹为当前窗口,则把标志 1 移到当前位置,SWIND.x0 变为 1, 其它标志依次前移,即变为:

SWIND.Lst[] = {0,2,3,4,5,1,6,7,8,9,10};

然后,按标志在 SWIND.Lst[]中的顺序,依次重画 SWIND.Buf 个窗口。

如果要把窗口 4 上移 100,则是把窗口 4 对应的栈元素结构中记录窗口位置的坐标进行修改,然后再重新画出所有的窗口,这样,看起来窗口就移动了;

如果要删除窗口 2,此时应先把窗口 2 的标志移到最上面来,再把 SWIND.x0 和 SWIND.Buf修改即可,最后再重画。此时:

SWIND.x0 = 1;

SWIND.Buf = 4;

SWIND.Lst[] = {0,3,4,5,1,2,6,7,8,9,10};

即窗口 1 仍为当前窗口,当前屏幕仅剩 4 个窗口。

如果要删除全部窗口,可以从当前窗口删起,直到SWIND.Buf=0 为止。

下面是完整的源程序:

/\*

HWIN.C——高级窗口设计

\*/

#include "graphics.h"

#include "alloc.h"

#include "conio.h"

#define MAX 10

/\* 定义窗口的最大数 \*/

typedef struct Win{

/\* 定义窗口的数据类型 \*/

int x1,y1,x2,y2;

/\* 窗口的左上角及右下角坐标 \*/

int Cx,Cy;

/\* 窗口内当前光标的位置 \*/

char \* WdBuf;

/\* 存储窗口内容 \*/

int Color;

/\* 窗口前景色 \*/

} WIND;

typedef struct Scr{

int x0;

int Buf;

int Lst[MAX];

} SWIND;

int SavWd();

int PopWd(int,int,int,int,int,int,int,int);

int DelTopWd();

```

void DelAllWd();
void InitGra(void);
void Quit();
void InitWd();
int DelWd(int Wnd);
int WdPos(int Wnd);
int PopCentWd(int,int,int,int,int,int);
int WdPopTop(int);
int WdRedraw(int);
int WdRedrawAll();
int SavTopWd();
void WdUp(int,int);
void WdDown(int,int);
void WdRight(int,int);
void WdLeft(int,int);
SWIND CWd;
WIND *SWd[MAX];
int main()
{
    int Wd1,Wd2,Wd3,Wd4,Wd5;
    InitWd();
    InitGra(); /* 初始化屏幕为图形方式 */
    Wd1=PopWd(15,12,50,50,0,BLUE,1,RED); /* 弹出窗口 1,红色,绿边框 */
    Wd2=PopWd(25,22,63,61,0,BLUE,1,YELLOW); /* 弹出窗口 2,黄色,绿边框 */
    Wd3=PopWd(115,112,150,150,0,WHITE,1,BLUE);
    /* 弹出窗口 3,蓝色,白边框 */
    Wd4=PopWd(125,222,263,261,0,WHITE,1,GREEN);
    /* 弹出窗口 4,绿色,白边框 */
    Wd5=PopCentWd(50,50,0,RED,1,WHITE); /* 弹出窗口 5,白色,红边框 */
    getch();
    WdPopTop(Wd1); /* 窗口 1 弹到当前窗口 */
    getch();
    WdUp(Wd4,100); /* 窗口 4 上移 100 */
    getch();
    WdDown(Wd2,200); /* 窗口 2 下移 200 */
    getch();
    WdLeft(Wd5,100); /* 窗口 5 左移 100 */
    getch();
    WdRight(Wd1,100); /* 窗口 1 右移 100 */

```

```

    getch();
    DelTopWd();          /* 删除最当前窗口 */
    getch();
    DelWd(Wd2);          /* 删除窗口 2 */
    getch();
    DelAllWd();          /* 删除所有窗口 */
    getch();
    Quit();
    return 0;
}

int SavWd()              /* 将窗口内容存入指针 */
{
    struct viewporttype OldV;
    int x1,y1,x2,y2;
    getviewsettings(&OldV);
    if (!CWd.x0)
    {
        x1 = 0;
        y1 = 0;
        x2 = 1;
        y2 = 1;
    }
    else
    {
        x1 = OldV.left;
        y1 = OldV.top;
        x2 = OldV.right;
        y2 = OldV.bottom;
    }
    SWd[CWd.x0]->WdBuf = (char *) (malloc(imagesize(x1,y1,x2,y2)));
    if (SWd[CWd.x0]->WdBuf == NULL) return 0;
    setviewport(0,0,639,479,1);
    getimage(x1,y1,x2,y2,SWd[CWd.x0] -> WdBuf);
    return 1;
}

/* 弹出一个窗口 */
int PopWd(int x1,int y1,int x2,int y2,int BdTp,int BdCl,int BkFil,int FilCl)
{
    struct viewporttype OldV;

```

```

int Oldx,Oldy,SavCl;
if (CWd.Buf >= MAX) return 0;
if ((SWd[CWd.x0] = (WIND * )malloc(sizeof(WIND))) == NULL) return 0;
getviewsettings(&OldV);
Oldx = getx(); Oldy = gety();
if (! SavWd())
{
    setviewport(OldV.left,OldV.top,OldV.right,OldV.bottom,1);
    moveto(Oldx,Oldy);
    free(SWd[CWd.x0]);
    return 0;
}
setviewport(0,0,639,479,1);
SavCl = getcolor();
setcolor(BdCl);
setlinestyle(BdTp,0,NORM _ WIDTH);
setfillstyle(BkFil,FilCl);
bar3d(x1,y1,x2,y2,0,0);
setviewport(x1,y1,x2,y2,1);
SWd[CWd.x0]->x1 = OldV.left; /* 保存新弹出的窗口信息 */
SWd[CWd.x0]->y1 = OldV.top;
SWd[CWd.x0]->x2 = OldV.right;
SWd[CWd.x0]->y2 = OldV.bottom;
SWd[CWd.x0]->Cx = Oldx;
SWd[CWd.x0]->Cy = Oldy;
SWd[CWd.x0]->Color = SavCl;
CWd.Buf++;
CWd.x0 = CWd.Lst[CWd.Buf];
setcolor(SavCl);
return (CWd.x0);
}

int DelTopWd() /* 删除当前窗口 */
{
    CWd.Buf--;
    CWd.x0 = CWd.Lst[CWd.Buf];
    if (CWd.Buf <= 0) return 0;
    setviewport(SWd[CWd.x0]->x1,SWd[CWd.x0]->y1,
    SWd[CWd.x0]->x2,SWd[CWd.x0]->y2,1);
    setcolor(SWd[CWd.x0]->Color);

```



```

    free(SWd[CWd.x0]);
    WdRedrawAll();
    return 1;
}

void DelAllWd()                /* 删除所有的窗口 */
{
    while (DelTopWd());
    cleardevice();
    return;
}

void InitGra(void)             /* 初始化屏幕为图形方式 */
{
    int GraphDrive = DETECT, GraphMode;
    registerbgidriver(EGAVGA _driver);
    initgraph(&GraphDrive, &GraphMode, "");
}

void Quit()
{
    closegraph();
    exit (0);
}

void InitWd()                  /* 参数初始化 */
{
    int i;
    CWd.x0 = 0;
    CWd.Buf = 0;
    for (i = 0; i < MAX; i++) CWd.Lst[i] = i;
}

int DelWd(int Wnd)             /* 删除指定窗口 */
{
    int Loc, i;
    if (! Wnd) return ;
    if (Wnd == CWd.x0)
    {
        DelTopWd();
        return;
    }
    Loc = WdPos(Wnd);
    if (Loc == -1) return;

```

```

    for (i = Loc; i < CWd.Buf+1; i++) CWd.Lst[i] = CWd.Lst[i+1];
    CWd.Lst[CWd.Buf] = Wnd;
    CWd.Buf--;
    CWd.x0 = CWd.Lst[CWd.Buf];
    WdRedrawAll();
    if (SWd[Wnd] != NULL)
    {
        free(SWd[Wnd]);
        SWd[Wnd] = NULL;
    }
    return;
}

int WdPos(int Wnd)          /* 搜索窗口在队列中的位置 */
{
    int i, Loc = -1;
    for (i = 0; i < CWd.Buf+1; i++)
    {
        if (CWd.Lst[i] == Wnd)
        {
            Loc = i;
            break;
        }
    }
    return (Loc);
}

/* 在屏幕中间弹出一个窗口 */
int PopCentWd(int Wid, int Hit, int BdTp, int BdCl, int BkFl, int FlCl)
{
    int x, y;
    x = (639 - Wid) / 2;
    y = (479 - Hit) / 2;
    return(PopWd(x, y, x+Wid, y+Hit, BdTp, BdCl, BkFl, FlCl));
}

int WdPopTop(int Wnd)      /* 将指定窗口弹到当前位置 */
{
    int oldx, oldy, savcolor;
    int Loc, i;
    if (! Wnd) return 0;
    if (Wnd == CWd.x0) return 0;

```

```

Loc = WdPos(Wnd);
if (Loc == -1) return 0;
if (! SavTopWd()) return 0;
for (i = Loc; i < CWd.Buf; i++) CWd.Lst[i] = CWd.Lst[i+1];
CWd.Lst[CWd.Buf] = Wnd;
WdRedraw(Wnd);
CWd.x0 = CWd.Lst[CWd.Buf];
setviewport(SWd[CWd.x0]->x1, SWd[CWd.x0]->y1, SWd[CWd.x0]->x2, \
    SWd[CWd.x0]->y2, 1);
if (SWd[CWd.x0] != NULL) free(SWd[CWd.x0]);
return 0;
}

int WdRedraw(int Wnd)          /* 重画指定的窗口 */
{
    if (SWd[Wnd]->WdBuf == NULL) return 0;
    if (WdPos(Wnd) == -1) return 0;
    putimage (SWd[Wnd] -> x1, SWd[Wnd] -> y1, SWd[Wnd] -> WdBuf,
        COPY_PUT);
    return 0;
}

int WdRedrawAll()             /* 重画所有窗口 */
{
    int i;
    void *Buf;
    struct viewporttype OldV;
    int x1, y1, x2, y2;
    getviewsettings(&OldV);
    x1 = OldV.left;
    y1 = OldV.top;
    x2 = OldV.right;
    y2 = OldV.bottom;
    Buf = (char *) malloc(imagesize(x1, y1, x2, y2));
    if (! Buf) return;
    setviewport(0, 0, 639, 479, 1);
    getimage(x1, y1, x2, y2, Buf);
    cleardevice();
    for (i = 0; i < CWd.Buf; i++) WdRedraw(CWd.Lst[i]);
    putimage(x1, y1, Buf, COPY_PUT);
    setviewport(x1, y1, x2, y2, 1);

```

```

    return;
}
int SavTopWd() /* 把当前窗口和内容存入栈中 */
{
    struct viewporttype OldV;
    int Oldx, Oldy, SavCl;
    if (CWd.Buf >= MAX) return 0;
    if ((SWd[CWd.x0] = (WIND *)malloc(sizeof(WIND))) == NULL) return 0;
    getviewsettings(&OldV);
    Oldx = getx();
    Oldy = gety();
    if (!SavWd())
    {
        setviewport(OldV.left, OldV.top, OldV.right, OldV.bottom, 1);
        moveto(Oldx, Oldy);
        free(SWd[CWd.x0]);
        return 0;
    }
    SWd[CWd.x0]—>x1 = OldV.left; /* 保存新弹出的窗口信息 */
    SWd[CWd.x0]—>y1 = OldV.top;
    SWd[CWd.x0]—>x2 = OldV.right;
    SWd[CWd.x0]—>y2 = OldV.bottom;
    SWd[CWd.x0]—>Cx = Oldx;
    SWd[CWd.x0]—>Cy = Oldy;
    SWd[CWd.x0]—>Color = SavCl;
    return (1);
}
void WdUp(int Wnd, int Dlt) /* 窗口上移 Dlt 个象素点 */
{
    int WdMv;
    if (CWd.x0 == Wnd) SavTopWd();
    WdMv = SWd[Wnd]—>y1;
    if (!WdMv) return;
    if (WdMv - Dlt <= 0) Dlt = WdMv;
    SWd[Wnd]—>y1 -= Dlt;
    SWd[Wnd]—>y2 -= Dlt;
    WdRedrawAll();
    if (CWd.x0 == Wnd)
    {

```

```

    setviewport(0,0,639,479,1);
    putimage (SWd[Wnd]->x1,SWd[Wnd]->y1+Dlt,SWd[Wnd]->WdBuf,
        XOR_PUT);
    putimage (SWd[Wnd]->x1,SWd[Wnd]->y1,SWd[Wnd]->WdBuf,
        COPY_PUT);
    free(SWd[Wnd]);
}
return;
}

void WdDown(int Wnd,int Dlt) /* 窗口下移 Dlt 个像素点 */
{
    int WdMv;
    if (CWd.x0 == Wnd) SavTopWd();
    WdMv = SWd[Wnd]->y2;
    if (WdMv == 479) return ;
    if (479-WdMv <= Dlt) Dlt = 479-WdMv;
    SWd[Wnd]->y1 += Dlt;
    SWd[Wnd]->y2 += Dlt;
    WdRedrawAll();
    if (CWd.x0 == Wnd)
    {
        setviewport(0,0,639,479,1);
        putimage (SWd[Wnd]->x1,SWd[Wnd]->y1-Dlt,SWd[Wnd]->WdBuf,
            XOR_PUT);
        putimage (SWd[Wnd]->x1,SWd[Wnd]->y1,SWd[Wnd]->WdBuf,
            COPY_PUT);
        free(SWd[Wnd]);
    }
    return;
}

void WdRight(int Wnd,int Dlt) /* 窗口右移 Dlt 个像素点 */
{
    int WdMv;
    if (CWd.x0 == Wnd) SavTopWd();
    WdMv=SWd[Wnd]->x2;
    if (WdMv == 639) return ;
    if (639-WdMv <= Dlt) Dlt = 639-WdMv;
    SWd[Wnd]->x1 += Dlt;
    SWd[Wnd]->x2 += Dlt;

```

```

WdRedrawAll();
if (CWd.x0 == Wnd)
{
    setviewport(0,0,639,479,1);
    putimage(SWd[Wnd] -> x1 - Dlt, SWd[Wnd] -> y1, SWd[Wnd] -> WdBuf,
        XOR_PUT);
    putimage(SWd[Wnd] -> x1, SWd[Wnd] -> y1, SWd[Wnd] -> WdBuf,
        COPY_PUT);
    free(SWd[Wnd]);
}
return;
}

void WdLeft(int Wnd, int Dlt) /* 窗口左移 Dlt 个象素点 */
{
    int WdMv;
    if (CWd.x0 == Wnd) SavTopWd();
    WdMv = SWd[Wnd] -> x1;
    if (!WdMv) return;
    if (WdMv < Dlt) Dlt = WdMv;
    SWd[Wnd] -> x1 -= Dlt;
    SWd[Wnd] -> x2 -= Dlt;
    WdRedrawAll();
    if (CWd.x0 == Wnd)
    {
        setviewport(0,0,639,479,1);
        putimage(SWd[Wnd] -> x1 + Dlt, SWd[Wnd] -> y1, SWd[Wnd] -> WdBuf,
            XOR_PUT);
        putimage(SWd[Wnd] -> x1, SWd[Wnd] -> y1, SWd[Wnd] -> WdBuf,
            COPY_PUT);
        free(SWd[Wnd]);
    }
    return;
}

```

# 关于鼠标的程序设计

## 第 5 章

本章内容：

### 5.1 鼠标的功能介绍及功能函数设计

#### 5.1.1 鼠标的常用功能介绍

#### 5.1.2 鼠标的基本函数设计

### 5.2 鼠标的应用编程

除键盘外,微型计算机最常用的输入设备是鼠标。鼠标具有简单、灵活、快速输入和定位的特点。它是通过 RS-232 接口同计算机联接的,本书中选用的鼠标型号为 ARTEC(3 键式),这种鼠标在使用前只要运行驱动程序即可:

C:\MOUSE>AMOUSE /1 (1#口)

## 5.1 鼠标的功能介绍及功能函数设计

### 5.1.1 鼠标的常用功能介绍

鼠标被安装后,应用程序就可以通过 INT 33H 中断调用来使用鼠标了。INT 33H 常用的中断功能如表 5-1。

表 5-1 INT 33H 常用中断功能表

功 能	入口参数	出口参数	参 数 描 述
AX=0000 鼠标初始化	无	AX BX	状态(-1:成功;0:未安装) 按钮的数目(1:按下,0:释放)
AX=0001 开始显示光标	无	无	
AX=0002 停止显示光标	无	无	
AX=0003 读取光标位置与按钮状态	无	BX (CX, DX)	按钮状态(B0:左;B1:右;B2:中) 光标位置
AX=0004 设置光标位置	(CX, DX)	无	光标位置(x, y)
AX=0007 设置 X 的最小和最大值	CX DX	无	x 的最小值 x 的最大值
AX=0008 设置 Y 的最小和最大值	CX DX	无	y 的最小值 y 的最大值
AX=0010H 设置光标出现的区域	(CX, DX) (SI, DI)	无	区域的左上角坐标 区域的右下角坐标



## 5.1.2 鼠标的基本函数设计

利用上面介绍的 INT 33H 的功能, 我们可以用 C 语言通过调用 INT 33H 中断来编制基本功能函数。在 C 语言中, 调用中断可采用的方法有:

- C 语言中嵌入汇编程序;
- 利用全程变量赋预寄存器值, 然后通过函数 `geninterrupt()` 调用中断实现;
- 利用 C 语言提供的联合类型 REGS, 通过函数 `int86()` 调用中断。

由于后一种方法编程比较方便, 这些函数采用该方法编写:

(1) 初始化鼠标, 并返回其状态;

0——没安装鼠标或没运行鼠标驱动程序;

非0——安装了鼠标且已运行了驱动程序。

```
int InitMouse()
```

```
{
    union REGS Inr, Outr;
    Inr.x.ax = 0;
    int86(0x33, &Inr, &Outr);
    return Outr.x.ax;
}
```

(2) 显示鼠标的光标, 字符方式为一矩形块, 图形方式为一个白色箭头;

出入口参数: 无

```
void ShowMouse()
```

```
{
    union REGS Inr, Outr;
    Inr.x.ax = 1;
    int86(0x33, &Inr, &Inr);
}
```

(3) 隐去鼠标的光标, 以便在屏幕上输出信息。

```
void HideMouse()
```

```
{
    union REGS Inr, Outr;
    Inr.x.ax = 2;
    int86(0x33, &Inr, &Inr);
}
```

(4) 读取鼠标的按键信息并返回按键时光标的屏幕坐标。

f: 按键标志,

x, y: 返回按键时光标的屏幕坐标。

```
void ReadMouse(int *f, int *x, int *y)
```

```
{
```

```

union REGS Inr, Outr;
Inr.x.ax = 3;
int86(0x33, &Inr, &Outr);
*f = Outr.x.bx;
*x = Outr.x.cx;
*y = Outr.x.dx;
}

```

(5) 设置鼠标光标在屏幕上的坐标。

入口参数: x, y: 要设置的坐标点。

出口参数: 无。

```

void SetMouseCoord(int x, int y)

```

```

{
    union REGS Inr, Outr;
    Inr.x.cx = x;
    Inr.x.dx = y;
    Inr.x.ax = 4;
    int86(0x33, &Inr, &Inr);
}

```

(6) 设置鼠标在屏幕上的最大移动范围。

```

void SetMouseArea(int Xmin,          /* X 方向最小值 */
                  int Ymin,          /* Y 方向最小值 */
                  int Xmax,          /* X 方向最大值 */
                  int Ymax)          /* Y 方向最大值 */

```

```

{
    union REGS Inr, Outr;
    Inr.x.cx = Xmin;
    Inr.x.dx = Xmax;
    Inr.x.ax = 7;
    int86(0x33, &Inr, &Inr);
    Inr.x.cx = Ymin;
    Inr.x.dx = Ymax;
    Inr.x.ax = 8;
    int86(0x33, &Inr, &Inr);
}

```

(7) 设置鼠标在屏幕上不显示光标的区域。

```

void MouseHideArea(int x1, int y1,    /* 要隐含区域的左上角坐标 */
                   int x2, int y2)    /* 要隐含区域的右下角坐标 */

```

```

{

```

```

union REGS Inr, Outr;
Inr.x.cx = x1;
Inr.x.dx = y1;
Inr.x.si = x2;
Inr.x.di = y2;
Inr.x.ax = 0x10;
int86(0x33, &Inr, &Inr);
}

```

(8) 判断鼠标光标的坐标点(x, y)是否在以(x1, y1)为左上角, 以(x2, y2)为右下角的矩形区域之内。

```

int MsInBox(int x1, int y1, int x2, int y2, int x, int y)
{
    return((x >= x1 && x <= x2 && y >= y1 && y <= y2) ? 1 : 0);
}

```

## 5.2 鼠标的应用编程

关于鼠标的程序设计是比较容易的。只要将上节中介绍的功能函数加入到应用程序中, 然后对其调用, 就如同调用 C 语言的库函数一样方便。鼠标编程中应注意的一点是, 每次鼠标的按键会产生多个按键信号, 但每次只需一个信号即可, 其它的信号在以后的程序段中有可能导致错误, 因此应将其它信号滤掉。可以采用延时的方法:

```
delay(500);
```

下面是一个鼠标的基本使用例行程序, 利用鼠标来实现图形方式下的一些操作。程序运行时:

- (1) 首先显示鼠标的光标(箭头);
- (2) 按左键后设置鼠标光标不显示的区域为(100,100,200,200);
- (3) 按左键后将鼠标光标移到屏幕右上角(600,10)处;
- (4) 按左键后设置鼠标光标的移动范围为(100,100,400,300);
- (5) 按左键后不显示鼠标光标, 而显示十字光标, 移动鼠标时, 十字光标随之移动, 按左键退出程序。

```
/*
```

```
MSCRS.C——鼠标的应用例
```

```
*/
```

```
#include "graphics.h"
```

```
#include "conio.h"
```

```
#include "dos.h"
```

```
#include "math.h"
```

```
#define TRUE 1
```

```

#define FALSE    0
#define LEFT     1
#define RIGHT    2
void InitGra();
void Cross();
int InitMouse();
void ShowMouse();
void HideMouse();
void ReadMouse(int *f, int *x, int *y);
void SetMouseCoord(int x, int y);
void SetMouseArea(int, int, int, int);
void MouseHideArea(int, int, int, int);
void ErrMsg();
void Quit();
int main()
{
    int Button, x, y;
    InitGra();                      /* 图形初始化 */
    if (! InitMouse()) ErrMsg();    /* 如果没安装鼠标, 则退出 */
    ShowMouse();
    while (Button != LEFT) ReadMouse(&Button, &x, &y); delay(300);
    MouseHideArea(100, 100, 200, 200);
    while (Button != LEFT) ReadMouse(&Button, &x, &y); delay(300);
    SetMouseCoord(600, 10);
    while (Button != LEFT) ReadMouse(&Button, &x, &y); delay(300);
    SetMouseArea(100, 100, 400, 300);
    while (Button != LEFT) ReadMouse(&Button, &x, &y); delay(300);
    Cross();
    Quit();
    return (0);
}

void InitGra()                      /* 初始化屏幕为图形方式 */
{
    int GarphDriver, GarphMode;
    GarphDriver = DETECT;
    registerbgidriver(EGAVGA _driver); /* 图形程序独立运行 */
    initgraph(&GarphDriver, &GarphMode, "");
}

void Cross()

```

```

{
    int Button, Oldx, Oldy;
    int x, y, First = TRUE;
    Button = Oldx = Oldy = 0;
    setwritemode(XOR_PUT);          /* 设置向屏幕写方式: 异或 */
    HideMouse();                   /* 隐去鼠标光标 */
    setcolor(15);
    while (Button != LEFT)         /* 等待鼠标的左键被按下 */
    {
        ReadMouse(&Button, &x, &y);
        if (Oldx != x || Oldy != y) /* 如果鼠标有移动 */
        {
            if (!First)            /* 如果不是第一次画出是十字光标 */
            {
                line(Oldx, 0, Oldx, 479); /* 抹去原位置的十字光标 */
                line(0, Oldy, 639, Oldy);
            }
            line(x, 0, x, 479);      /* 新位置重画十字光标 */
            line(0, y, 639, y);
            Oldx = x;                /* 保存新位置的坐标值 */
            Oldy = y;
            First = FALSE;
        }
    }
    delay(300);                    /* 延时300毫秒, 以滤掉多余的按键信号 */
    line(x, 0, x, 479);            /* 抹掉十字光标 */
    line(0, y, 639, y);
    setwritemode(COPY_PUT);        /* 写方式还原, 即设置为覆盖方式 */
}

void Quit()
{
    closegraph();
    exit(0);
}

int InitMouse()
{
    union REGS Inr, Outr;
    Inr.x.ax = 0;

```

```

    int86(0x33, &Inr, &Outr);
    return Outr.x.ax;
}
void ShowMouse()
{
    union REGS Inr, Outr;
    Inr.x.ax = 1;
    int86(0x33, &Inr, &Inr);
}
void HideMouse()
{
    union REGS Inr, Outr;
    Inr.x.ax = 2;
    int86(0x33, &Inr, &Inr);
}
/*
    读取鼠标的按键信息并返回按键时光标的屏幕坐标.
    f:  按键标志,
    x, y: 返回按键时光标的屏幕坐标.
*/
void ReadMouse(int *f, int *x, int *y)
{
    union REGS Inr, Outr;
    Inr.x.ax = 3;
    int86(0x33, &Inr, &Outr);
    *f = Outr.x.bx;
    *x = Outr.x.cx;
    *y = Outr.x.dx;
}
void SetMouseCoord(int x, int y)
{
    union REGS Inr, Outr;
    Inr.x.cx = x;
    Inr.x.dx = y;
    Inr.x.ax = 4;
    int86(0x33, &Inr, &Inr);
}
/*
    设置鼠标在屏幕上的移动区域.

```

```

*/
void SetMouseArea(int Xmin,          /* X 方向最小值 */
                  int Ymin,          /* Y 方向最小值 */
                  int Xmax,          /* X 方向最大值 */
                  int Ymax)          /* Y 方向最大值 */
{
    union REGS Inr, Outr;
    Inr.x.cx = Xmin;
    Inr.x.dx = Xmax;
    Inr.x.ax = 7;
    int86(0x33, &Inr, &Inr);
    Inr.x.cx = Ymin;
    Inr.x.dx = Ymax;
    Inr.x.ax = 8;
    int86(0x33, &Inr, &Inr);
}
/*
    设置鼠标在屏幕上不显示光标的区域.
*/
void MouseHideArea(int x1, int y1,   /* 要隐含区域的左上角坐标 */
                   int x2, int y2)   /* 要隐含区域的右下角坐标 */
{
    union REGS Inr, Outr;
    Inr.x.cx = x1;
    Inr.x.dx = y1;
    Inr.x.si = x2;
    Inr.x.di = y2;
    Inr.x.ax = 0x10;
    int86(0x33, &Inr, &Inr);
}
void ErrMsg()
{
    printf("No Mouse Error!");
    getch();
    Quit();
}

```

# 菜单程序设计

## 第 6 章

本章内容:

- 6.1 西文 DOS 下的菜单设计
  - 6.1.1 西文 DOS 下的西文下拉式菜单设计
  - 6.1.2 西文 DOS 下的汉字下拉式菜单设计
  - 6.1.3 西文 DOS 下的汉字弹出式下拉菜单设计
- 6.2 中文 DOS 下的汉字下拉式菜单设计
- 6.3 用鼠标选择立体按钮菜单
- 6.4 鼠标和键盘兼容的下拉式菜单设计
- 6.5 交替运行其它语言程序时的菜单设计



菜单的设计在用户界面设计中占有相当一部分内容。设计一个高质量的菜单不仅能使系统漂亮美观,更主要的是能使操作者使用方便,并避免一些因误操作而带来的严重后果。本章主要介绍用 Turbo C/Borland C 编写各种菜单的技术。这些菜单包括:西文 DOS 下的西文及汉字下拉式菜单设计;西文 DOS 下的汉字弹出式菜单设计,中文 DOS 下的汉字下拉式菜单设计;用鼠标选择立体按钮菜单,键盘和鼠标兼容的下拉式菜单设计,以及交替运行其它语言程序时的菜单设计。

## 6.1 西文 DOS 下的菜单设计

### 6.1.1 西文 DOS 下的西文下拉式菜单的设计

下拉菜单是一个典型的窗口菜单。所谓窗口是指在屏幕上划分的矩形区域,它可以从屏幕上消失,也可以重新显示在屏幕上。窗口之间也允许覆盖。下拉式菜单则是自上而下向屏幕弹出一个个窗口菜单供操作者选择或输入内容。Turbo C/Borland C 集成开发环境就是一个下拉式菜单的例子。下拉式菜单一般有一个主菜单,其中包括几个选择项,主菜单的每一项又可分为下一级菜单,这样逐级下分,用一个个窗口的形式弹出在屏幕上,一旦操作完毕又可从屏幕上消失,并恢复原来的屏幕状态。下拉式菜单几乎为所有商业化软件全部或部分采用。这种菜单有很多好处:首先是界面友好、直观,整体感强,使操作者一目了然,并能方便地在各功能间选择,灵活地随时转向另一功能;再次,下拉式菜单占用屏幕空间小,只占屏幕最上边一行,同时拉出一个子菜单栏,这样,余下的屏幕空间大,可用来显示要表达的各种信息,如计算过程和处理过程等各种图表和数字信息。

设计下拉式菜单一般有以下几个步骤:

- (1) 保存屏幕弹出前的矩形区域;
- (2) 显示菜单正文;
- (3) 产生光条;
- (4) 读取按键信息;
- (5) 处理用户按键及分支出其它菜单;
- (6) 恢复屏幕弹出前的原始状态。

1. 保存和恢复屏幕:

在文本方式下,保存屏幕可用 `gettext()` 函数,其格式如下:

```
int gettext(int x1,int y1,int x2,int y2,void *buffer);
```

该函数将屏幕上指定的矩形区域内的文本内容存入 `buffer` 指针指向的一个内存空间。内存大小用下式计算:

所用字节大小 = 行数  $\times$  列数  $\times 2$ ;

其中:行数 =  $y2 - y1 + 1$ ; 列数 =  $x2 - x1 + 1$ ;

恢复屏幕保存的内容可用函数 `puttext()`,其格式为:

```
int puttext(int x1,int y1,int x2,int y2,void *buffer);
```

将 `gettext()` 函数存入内存 `buffer` 中的文字内容拷贝到屏幕指定的位置。

## 2. 光条的产生方法

在文本方式下,最直接的方法是采用不同背景色和字符色重写应为光条的菜单项。

## 3. 读取按键信息

在进行菜单设计中很关键的一点是功能键的读取。由于键盘上的键有两种:ASCII 码键(即“普通键”)和特殊功能键,当用户进行选择菜单时,有可能按下“普通键”或特殊功能键,在许多资料介绍的程序中,大都是先读键,再判断是“普通键”还是特殊功能键,然后再进行分支判断,程序的设计较麻烦,可读性差。为此,作者在进行菜单设计中,编制了一个使用非常方便的函数 GetKey(),用来返回键值,如果是特殊功能键则返回一个大于 256 的整数;否则,返回的是 ASCII 码值。这样,在程序中两种类型的按键可以在一个循环中判断,不必再去关心它是哪一种键。GetKey()函数如下:

```
int GetKey()
{
    int Ch,Low,Hig;
    Ch = bioskey();           /* 等待按键 */
    Low = Ch & 0x00ff;        /* 取低 8 位 */
    Hig = (Ch & 0xff00) >> 8; /* 取高 8 位 */
    return (Low == 0 ? Hig+256 :Low); /* 如果是特殊功能键, */
}                             /* 返回一个大于 256 的整数 */
```

函数 GetKey()中的一个关键库函数是 bioskey(),它通过调用 BIOS 的 INT16H 中断对键盘进行管理,该中断包括两个功能:一是判断是否按了键并返回所按键的 ASCII 码(字符键)或扩充键(功能键)值,二是返回键盘的状态。

该函数的调用格式为:

bioskey(int cmd);

头文件为 conio.h,如果参数 cmd=0,则表示函数返回一个键盘输入的信息,该值为一个两字节 16 位二进制整数,如果没有键盘操作,将一直处于等待状态,若按下的是一般的 ASCII 码键(“普通键”),则返回的两字节整数中的低 8 位为所按字符的 ASCII 码值,若按下的是特殊功能键,则返回的两字节整数中的低 8 位为 0,高 8 位为所按键的扩充码。

表 6-1 键盘输入控制字符(00~31)表

ASCII 码	输入字符	作 用	ASCII 码	输入字符	作 用
00	Ctrl+Z		16	Ctrl+P	
01	Ctrl+A		17	Ctrl+Q	
02	Ctrl+B		18	Ctrl+R	
03	Ctrl+C		19	Ctrl+S	
04	Ctrl+D		20	Ctrl+T	
05	Ctrl+E		21	Ctrl+U	

续表

06	Ctrl+F		22	Ctrl+V	
07	Ctrl+G	喇叭发声	23	Ctrl+W	
08	BACKSPAC	回 空 格	24	Ctrl+X	
09	→	光标跳移	25	Ctrl+Y	
10	Ctrl+J	换 行	26	Ctrl+Z	
11	Ctrl+K	光标左上角	27	ESC	
12	Ctrl+L	换 页	28	Ctrl+\	光 标 右 移
13		回车换行	29	Ctrl+]	光 标 左 移
14	Ctrl+N		30	Ctrl+6	光 标 上 移
15	Ctrl+O		31	Ctrl+-	光 标 下 移

扩充 ASCII 码字符由两个代码组成:第一个代码是 0,第二个代码如表 6-2 所示。

表 6-2 扩充 ASCII 码表

第二代码	对应的 按 键	第二代码	对应的 按 键
15	←	82	Ins
16~25	Alt+Q,W,E,R,T,Y,U,I,O,P	82	Del
30~38	Alt+A,S,D,F,G,H,J,K,L	84~93	F11~F20(Shift+F1 到 Shift+F10)
44~50	Alt+Z,X,C,V,B,N,M	94~103	F21~F30(Ctrl+F1 到 Ctrl+F10)
59~68	功能键 F1~F10	104~113	F31~F40(Alt+F1 到 Alt+F10)
71	Home	114	Ctrl+PrtScrn
72	↑	115	Ctrl+←
73	PgUp	116	Ctrl+P→
75	←	117	Ctrl+End
77	→	118	Ctrl+PgDn
79	End	119	Ctrl+Home
80	↓	120~131	Alt+1,2,3,4,5,6,7,8,9,0,-,=
81	PgDn	132	Ctrl+PgUp

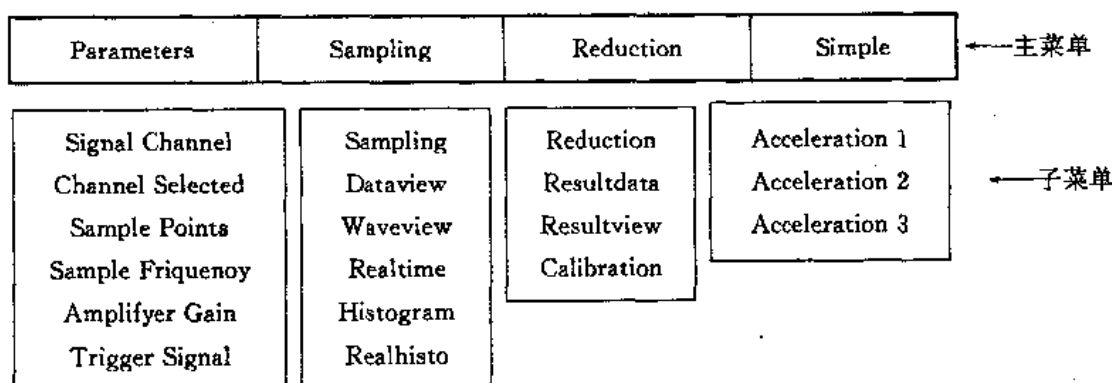
如果参数 cmd=1,bioskey()函数将用来查询是否按了键,是返回非 0,不是返回 0;

如果参数 cmd=2, bioskey() 函数将返回键盘上控制键的状态。

有了 GetKey() 函数, 只要进行一次判断就可以了, 结合宏定义 #define, 就可设计出可读性非常好的程序来, 下面是采用 GetKey() 函数读的常用功能键的宏定义:

```
#define LEFT      331      /* 左箭头键 */
#define RIGHT     333      /* 右箭头键 */
#define UPPER     328      /* 上箭头键 */
#define DOWN      336      /* 下箭头键 */
#define ALT-X     301      /* Alt+X 键 */
#define F1        315      /* 功能键 F1 */
#define F2        316      /* 功能键 F2 */
#define F3        317      /* 功能键 F3 */
#define ESC       27       /* ESC 键 */
#define ENTER     13       /* 回车键 */
```

设计下拉式菜单的另一个关键就是在弹出菜单窗口之前, 将要被该窗口占用的屏幕区域保存起来, 然后产生这一级窗口, 并可使用光标键选择菜单中的各项, 回车键认可选择。如果某一项还有下一级菜单, 则按同样的方法产生再下一级菜单窗口。另外, 在选择或输入结束后, 还要用 ESC 键向上逐级退回到主菜单, 也就是将保存的原始屏幕又向上级逐级释放出来。在文本方式时, 可用上面介绍的 gettext() 函数保存屏幕规定区域的内容, 当需要时又可用 puttext() 函数释放出来, 再加上 GetKey() 函数, 就可完成下拉式菜单的设计要求。下面通过一个例子说明设计下拉式菜单的方法, 本例设计了一个完整的下拉式菜单, 只要修改内容就可以应用于读者的程序中, 该菜单的结构如下:



主菜单共有 4 项, 子菜单分别有 6, 6, 4, 3 项, 对应于主菜单中的每一项, 称其为主菜单项。某一个主菜单项下对应的同级子菜单项, 显示在一个窗口之中, 称这个窗口为子菜单栏, 子菜单栏中的每一项称为子菜单项。

显示时, 在屏幕的最上边一行显示主菜单, 其中的一项为光条显示 (同其它项颜色不同, 一般是第一项), 以及该项对应的子菜单栏。当按下 ←、→ 键时, 主菜单项的光条作左、右移动, 相应的子菜单栏也随之移动, 原来的子菜单栏消失, 弹出新的子菜单栏; 当按下 ↑、↓ 键时, 子菜单栏不动, 光条在子项上作上下移动; 按回车键时, 执行相应的功能, 执行完毕, 又回到该菜单

界面。按 Alt+X 或 ESC 键,则退出程序。

下面是实现这些功能的源程序:

```
/*
    MENUE.C——西文 DOS 下的西文下拉式菜单
*/
#include "graphics.h"
#include "conio.h"
#include "bios.h"
#include "process.h"
#include "string.h"
#include "stdio.h"
#define LEFT    331  /* 功能键的宏定义 */
#define RIGHT   333
#define UPPER   328
#define DOWN    336
#define ALT _X 301
#define ESC     27
#define ENTER   13
void WrtMnMenu(void);
void WrtSbMenu(void);
void SlctMenu(void);
void SlctMnMenu(void);
void SlctSbMenu(void);
int GetKey(void);
void SbFunGo(void);
void Screen(void);
void Wind(int,int,int,int,int,int,int,int,int);
void Quit(void);
int Mm = 0;                /* 主菜单选项 */
int Smm[4];                /* 各项主菜单下的子菜单选项 */
int SbNum[4] = {6,6,4,3}; /* 各项主菜单的子菜单项数 */
int SbWid[4] = {20,10,13,16}; /* 各项子菜单的窗口宽度 */
int SbX[4] = {5,20,33,47}; /* 各项子菜单的 X 坐标 */
int Key = 0;               /* 选择的键 */
char Buf[1000];
char * Main[4] = {"Parameters","Sampling","Reduction","Simple"};
                                /* 主菜单项名 */
char * Sub[4][6] = {
    {"Signal Channel",          /* 子菜单项名 */
```

```

        "Channel Selected",
        "Sample Points",
        "Sample Friquency",
        "Amplifyer Gain",
        "Trigger signal"
    },
    {"Sampling ",
     "Dataview ",
     "Waveview ",
     "Realtime ",
     "Histogram",
     "Realhisto"
    },
    {"Reduction ",
     "Resultdata ",
     "Resultview ",
     "Calibration"
    },
    {"Acceleration 1",
     "Acceleration 2",
     "Acceleration 3"
    }
    });

int main(void)
{
    Screen();
    WrtMnMenu();
    WrtSbMenu();
    SlctMenu();
    Quit();
    return(0);
}

void WrtMnMenu(void)
{
    int i;
    window(1,1,80,25);          /* 显示主菜单 */
    textattr(0x3e);
    for (i = 0;i < 4;i++)
    {
        gotoxy(SbX[i],1);
    }
}

```

```

        cputs(Main[i]);
    }
    gotoxy(SbX[Mm],1);          /* 设置光条颜色 */
    textattr(0x4e);
    cputs(Main[Mm]);
}
void WrtSbMenu(void)
{
    int i;
    gettext(SbX[Mm]-2,2,SbX[Mm]+SbWid[Mm]+1,SbNum[Mm]+4,Buf);
    Wind(SbX[Mm]-2,2,SbX[Mm]+SbWid[Mm],SbNum[Mm]+3,1,1,3,15,1);
    textattr(0x3f);            /* 显示子菜单 */
    for (i=0;i < SbNum[Mm];i++)
    {
        gotoxy(2,1+i);
        cputs(Sub[Mm][i]);
    }
    textattr(0x1e);            /* 子菜单选项的颜色 */
    gotoxy(2,Smm[Mm]+1);
    cputs(Sub[Mm][Smm[Mm]]);
}
void SlctMenu(void)            /* 执行菜单选择功能 */
{
    while(Key != ALT_X && Key != ESC)
    {
        Key = GetKey();
        if (Key == LEFT || Key == RIGHT) SlctMnMenu();
        if (Key == UPPER || Key == DOWN) SlctSbMenu();
        if (Key == ENTER)      SbFunGo();
    }
    return;
}
void SlctMnMenu(void)
{
    window(1,1,80,25);
    textattr(0x3e);
    gotoxy(SbX[Mm],1);
    cputs(Main[Mm]);
    textattr(0x31);

```

```

    puttext(SbX[Mm]-2,2,SbX[Mm]+SbWid[Mm]+1,SbNum[Mm]+4,Buf);
    if (Key == LEFT) Mm = Mm == 0 ? 3 : Mm-1;
    if (Key == RIGHT) Mm = Mm == 3 ? 0 : Mm+1;
    textattr(0x4e);
    gotoxy(SbX[Mm],1);
    cputs(Main[Mm]);
    WrtSbMenu();
}

void SlctSbMenu(void)
{
    textattr(0x3f);
    gotoxy(2,1+Smm[Mm]);
    cputs(Sub[Mm][Smm[Mm]]);
    if (Key == UPPER) Smm[Mm] = Smm[Mm] == 0 ? SbNum[Mm]-1 :
        Smm[Mm]-1;
    if (Key == DOWN) Smm[Mm] = Smm[Mm] == SbNum[Mm]-1 ? 0 :
        Smm[Mm]+1;
    textattr(0x1e);
    gotoxy(2,Smm[Mm]+1);
    cputs(Sub[Mm][Smm[Mm]]);
}

int GetKey(void)
{
    int Ch,Low,Hig;
    Ch = bioskey(0);
    Low = Ch & 0x00ff;
    Hig = (Ch & 0xff00) >> 8;
    return (Low == 0 ? Hig+256 : Low);
}

void SbFunGo(void) /* 根据键选执行相应的功能 */
{
    switch(Mm)
    {
        case 0:
            switch(Smm[0])
            {
                case 0: break;
                case 1: break;
                case 2: break;
            }
    }
}

```



```

        case 3:    break;
        case 4:    break;
        case 5:    break;
        case 6:    break;
    } break;
case 1:
    switch(Smm[1])
    {
        case 0:    break;
        case 1:    break;
        case 2:    break;
        case 3:    break;
        case 4:    break;
        case 5:    break;
    } break;
case 2:
    switch(Smm[2])
    {
        case 0:    break;
        case 1:    break;
        case 2:    break;
        case 3:    break;
    } break;
case 3:
    switch(Smm[3])
    {
        case 0:    break;
        case 1:    break;
        case 2:    break;
    } break;
}
}
void Screen(void)
{
    window(1,1,80,25);
    textattr(0x17);
    clrscr();
    window(1,1,80,1);
    textattr(0x3e);

```

```

clrscr();
window(1,24,80,24);
textattr(0x74);
clrscr();
cputs("\x18 ");          /* ↑键 */
cputs("\x19 ");          /* ↓键 */
cputs("\x1a ");          /* →键 */
cputs("\x1b ");          /* ←键 */
textattr(0x7e); cputs("and ");
textattr(0x74); cputs("Enter ");
textattr(0x7e); cputs("to select menu. ");
textattr(0x74); cputs(" Alt _ X ");
textattr(0x7e); cputs(" or ");
textattr(0x74); cputs(" ESC ");
textattr(0x7e); cputs(" to quit");
window(1,25,80,25);
textattr(0x6A);
clrscr();
cputs("BEIJING AGRICLUTURAL ENGINEERING UNIVERSITY & SHAN DONG
      COLLGE \OF ENGINEERING");
Wind(1,2,80,23,2,0,1,15,1);
window(1,1,80,25);
}
/* 字符窗口函数:
   x1,y1:窗口左上角坐标;
   x2,y2:窗口右下角坐标;
   FrmTp: 窗口边框类型:0—无边框;1—单线边框;2—双线边框;
   IsBk:是否带阴影背景;
   BCl:字符颜色;
   TCl:背景颜色;
   BgCl:阴影背景的颜色.
*/
void Wind(int x1,int y1,int x2,int y2,int FrmTp,int IsBk,int BCl,int TCl,/int BgCl)
{
    int i;
    int c[2][6]={0xda,0xc4,0xbf,0xb3,0xc0,0xd9}, /* 单线字符 */
              {0xc9,0xcd,0xbb,0xba,0xc8,0xbc}}; /* 双线字符 */
    textcolor(TCl);
    textbackground(BCl);

```

```

window(x1,y1,x2,y2);
clrscr();
if (FrmTp)                                /* 有边框线 */
{
    window(1,1,80,25);
    gotoxy(x1,y1);
    putch(c[FrmTp-1][0]);
    for (i = x1+1;i < x2;i++)
        putch(c[FrmTp-1][1]);
    putch(c[FrmTp-1][2]);
    for (i = y1+1;i < y2;i++)
    {
        gotoxy(x1,i);
        putch(c[FrmTp-1][3]);
        gotoxy(x2,i);
        putch(c[FrmTp-1][3]);
    }
    gotoxy(x1,y2);
    putch(c[FrmTp-1][4]);
    for (i = x1+1;i < x2;i++)
        putch(c[FrmTp-1][1]);
    putch(c[FrmTp-1][5]);
}
if (IsBk)                                  /* 有阴影背景 */
{
    textcolor(BgCl);
    textbackground(0);
    for (i = y1+1;i < y2+1;i++)
    {
        gotoxy(x2+1,i);
        putch('/xb1');
    }
    for (i = x1+1;i < x2+2;i++)
    {
        gotoxy(i,y2+1);
        putch('/xb1');
    }
}
window(x1+1,y1+1,x2-1,y2-1);

```

```

}
void Quit(void)
{
    textbackground(0);
    textcolor(7);
    window(1,1,80,25);
    clrscr();
    exit(0);
}

```

本程序实际运行时,在屏幕上的显示如图 6-1。

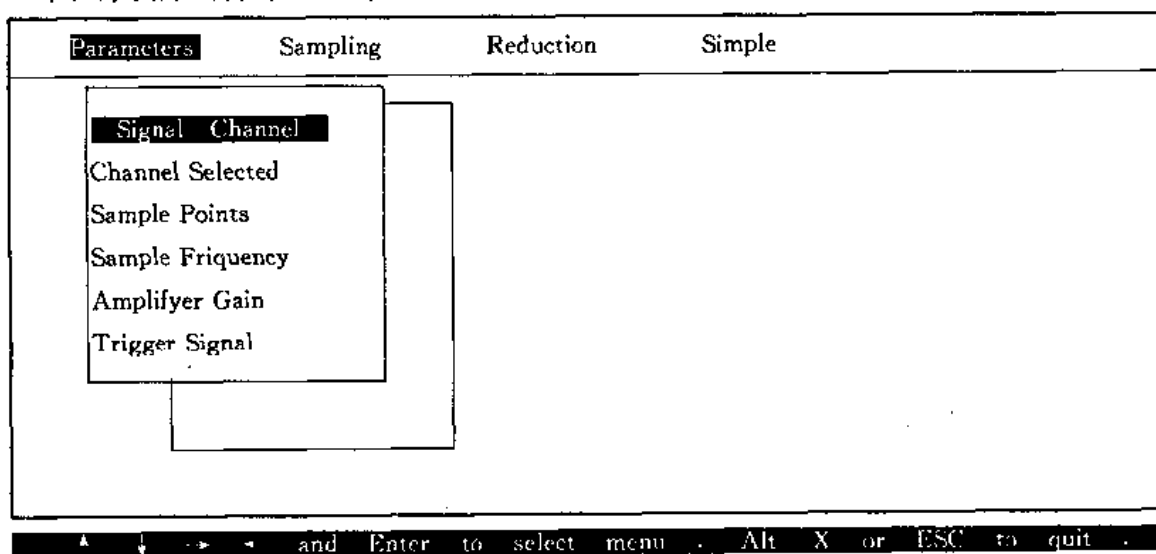


图 6-1

当按→←键时,光条将作左右移动,原来的光条项“Parameters”将变为正常显示,其对应的子菜单栏也作相应的变化,当前的菜单栏消失,弹出 Sampling(按→时)或 Simple(按←时)对应的子菜单栏;当按↑↓键时,光条将在当前的菜单栏内的菜单项上作上下移动,当前的菜单项“Signal Channel”变为正常显示,下一个菜单项“Channel Selected”(按↓时)或上一个菜单项“Trigger Signal”(按↑键时)变为光条显示,按 ENTER 键时,则执行当前菜单项对应的功能。

### 6.1.2 西文 DOS 下的汉字下拉式菜单设计

在 3.2 中介绍了西文 DOS 图方式下显示汉字的技术,本节介绍西文 DOS 图形方式下的汉字下拉式菜单设计。

在文本方式下保存和恢复屏幕可用函数 gettext()和 puttext(),在图形方式下保存和恢复屏幕不能用这两个函数,而只能用 getimage()和 putimage(),它们的格式如下:

```

void far getimage(int x1,int y1,int x2,int y2,void far *mapbuf);
void far putimage(int x1,int y1,*mapbuf,int op);

```

```
unsigned far imagesize(int x1,int y1,int x2,int y2);
```

这三个函数将屏幕上的图象复制到内存,然后再将内存中的图象回送到屏幕上。首先通过函数 `imagesize()` 测试要保存左上角为  $(x1,y1)$ , 右下角为  $(x2,y2)$  的图形屏幕区域内的全部内容需要多少个字节,然后再给 `mapbuf` 分配一个所测试数目的字节内存空间的指针,通过调用 `getimage()` 函数就可以将该区域内的图象保存在内存中,需要时可用 `putimage()` 函数将该图象输出到左上角为  $(x1,y1)$  的位置上,其中, `putimage()` 函数中的参数 `op` 规定如何释放内存中的图象。

表 6-3 `putimage()` 函数中的 `op` 值

符号常数	数 值	含 义
<code>COPY_PUT</code>	0	复 制
<code>XOR_PUT</code>	1	与屏幕图象异或后复制
<code>OR_PUT</code>	2	与屏幕图象或后复制
<code>AND_PUT</code>	3	与屏幕图象与后复制
<code>NOT_PUT</code>	4	复制反相的图形

注意: `imagesize()` 函数只能返回字节数小于 64K 字节的图象区域,否则将会出错,出错时返回 -1。

由此可以看出,用 `getimage()` 函数可保存的屏幕区域是比较小的。如果要保存字节大于 64K 的屏幕区域,则不能再使用该函数,在这种情况下,可用函数 `SavScr()` 和 `RestScr()` 函数来实现。下面是这两个函数及其应用的例子,在 VGA 640×480 的屏幕上画出图形,将整个屏幕保存然后再恢复,这两个函数将屏幕区域从上而下划分成 3 个小区域,并分别存放在对应的块中:

```
/*
```

SavRst.C——`SavScr()` 和 `RestScr()` 函数的应用例

```
*/
```

```
#include "stdio.h"
```

```
#include "alloc.h"
```

```
#include "graphics.h"
```

```
#include "conio.h"
```

```
#include "process.h"
```

```
void InitGra(void);
```

```
void DrawBar();
```

```
int SavScr(int x1,int y1,int x2,int y2,void far *Buf[],int *Yst);
```

```
void RestScr(int x1,int y1,int Block,int *Yst,void far *Buf[],int Op);
```

```
void Quit();
```

```

int main(void)
{
    int Block,Yst[4];
    void far *Buf[4];
    InitGra();
    DrawBar();
    if (! (Block = SavScr(0,0,639,479,Buf,Yst))) Quit();/* 保存整屏内容 */
    getch();
    setbkcolor(0);
    cleardevice();
    getch();
    RestScr(0,0,Block,Yst,Buf,COPY_PUT);/* 恢复整屏内容 */
    getch();
    Quit();
}

void InitGra(void)
{
    int GraphMode,DrpHDrive = DETECT;
    registerbgidriver(EGAVGA_driver);
    initgraph(&DrpHDrive,&GraphMode,"");
}

void DrawBar() /* 画 16 色矩形条 */
{
    int i;
    for (i = 0;i < 15;i++)
    {
        setfillstyle(1,i);
        bar(i * 40,0,(i+1) * 40,479);
    }
}

/* x1,y1:要保存区域的左上角坐标;
   x2,y2:要保存区域的右下角坐标;
   Buf:将矩形区域图形拷贝到以 Buf 为首指针的内存块中,当所需字节超界时,
   自动将欲存储的屏幕区域从上到下划分若干个小区域,并分别存放对应的块中;
   Yst:指向代表各小区域左上角 Y 坐标的数组;
*/

int SavScr(int x1,int y1,int x2,int y2,void far *Buf[],int *Yst)
{
    int Yend,Yinc,Dlty,i,Block = 1;

```

```

    unsigned size[4],Tsize;
    Dlty = y2-y1;
    Yinc = Dlty;
    while ((Tsize = imagesize(x1,0,x2,Yinc-1)) == 0)
    {
        Block ++;
        Yinc = (Dlty+Block-1)/Block;
    }
    Yst[0] = y1;
    Yend = y1+Yinc-1;
    for (i = 0;i < Block;i++)
    {
        if (i)
        {
            Yst[i] = Yend+1;
            Yend += Yinc;
            if (i == Block-1) Yend = y2;
        }
        size[i] = imagesize(x1,Yst[i],x2,Yend);
        if ((Buf[i] = farmalloc(size[i])) == NULL)
        {
            printf("内存不够!");
            exit (0);
        }
        getimage(x1,Yst[i],x2,Yend,Buf[i]);
    }
    return (Block);
}
/* x1,y1:图形重画到区域的左上角坐标;
   Block,Sav 分配的内存块数;
   Buf:将矩形区域图形拷贝到以 Buf 为首指针的内存块中;
   Yst:指向代表各小区域左上角 Y 坐标的数组;
   Op: 同 putimage()中的 op 含义相同.
*/
void RestScr(int x1,int y1,int Block,int * Yst,void far * Buf[],int Op)
{
    int i;
    for (i = 0;i < Block;i++) putimage(x1,Yst[i]+y1,Buf[i],Op);
    for (i = 0;i < Block;i++) farfree(Buf[i]);
}

```

```

}
void Quit()
{
    closegraph();
    exit (0);
}

```

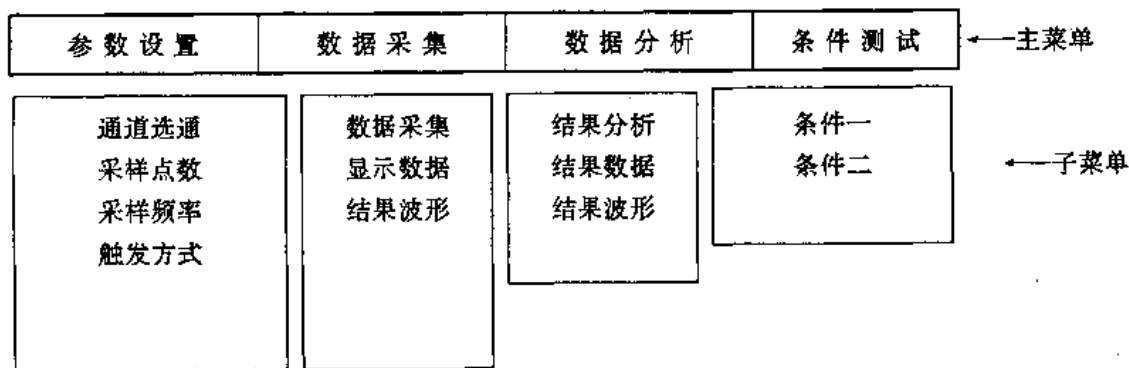
在图形方式下,光条的产生通常有以下两种方法:

(1) 采用不同的前景色和背景色重新显示一次;

(2) 采用图形屏幕操作函数 `getimage()` 和 `putimage()`, 先用 `getimage()` 函数存储光条大小的屏幕内容, 设置 `putimage()` 函数中的参数值使存储在内存中的图象反相释放在同一位置, 就可产生光条。

后一种方法产生的光条的颜色不能灵活控制, 它只能是原来颜色的反相, 再者, 它需要占用内存。因此, 本书采用前一种方法来产生光条。

在这一节中, 我们建立具有如下结构的汉字下拉式菜单:



下面是菜单的源程序:

```

/*
  MENU.C——西文 DOS 下的汉字下拉菜单
*/
#include "graphics.h"
#include "string.h"
#include "bios.h"
#include "conio.h"
#include "process.h"
#include "stdio.h"
#include "stdlib.h"
#define LEFT    331
#define RIGHT   333
#define UPPER   328
#define DOWN    336

```



```

#define ALT __X 301
#define ESC 27
#define ENTER 13
void WrtMnMenu(void);
void MnMenuColor(int n,int BCl,int TCl);
void WrtSbMenu(void);
void SbMenuColor(int n,int BCl,int TCl);
void SlctMenu(void);
void SlctMnMenu(void);
void SlctSbMenu(void);
int GetKey(void);
void SubFunGo(void);
void Quit(void);
void Screen(void);
void Wind(int,int,int,int,int,int,int,int,int);
void InitGr(void);
void PutCC16(int x,int y,int Wid,int TCl,char *Str);
void OpenLIB(void);
FILE *fp16;
int Hj = 20; /* 定义汉字显示的行距 */
int Mm = 0; /* 主菜单选项 */
int Smm[4]; /* 各子菜单选项 */
int SbNum[4] = {4,4,3,2}; /* 各子菜单的项数 */
int SbWid[4] = {93,93,93,90}; /* 各项子菜单的宽度 */
int SbX[4] = {20,140,270,400}; /* 各项子菜单水平方向的坐标值 */
int Key = 0; /* 选择键值 */
char *Main[4] = {"参数设置","数据采集","数据分析","条件测试"};
char *Sub[4][4] = {
    {"通道选通",
     "采样点数",
     "采样频率",
     "触发方式"},
    {"数据采集",
     "显示数据",
     "显示波形",
     "初始校准"},
    {"结果分析",
     "结果数据",
     "结果波形"},
    {"条件一",

```

"条件二" } };

```
int main(void)
```

```
{
    OpenLIB();
    InitGr();
    Screen();
    WrtMnMenu();
    WrtSbMenu();
    SlctMenu();
    Quit();
    return(0);
}
```

```
void WrtMnMenu(void)          /* 写主菜单 */
```

```
{
    int i;
    for(i = 0; i < 4; i++) MnMenuColor(i, 3, 14);
    MnMenuColor(Mm, 4, 14); /* 产生光条 */
}
```

/\* n: 菜单的第几项;

BCL: 背景颜色;

TCL: 前景颜色.

\*/

```
void MnMenuColor(int n, int BCL, int TCL)
```

```
{
    setfillstyle(1, BCL);
    bar(SbX[n], 1, SbX[n] + SbWid[Mm], 24);
    PutCC16(SbX[n] + 10, 5, 4, TCL, Main[n]);
}
```

```
void WrtSbMenu(void)
```

```
{
    int i;
    Wind(SbX[Mm], 26, SbX[Mm] + SbWid[Mm], SbNum[Mm] * Hj + 34, 1, 1, 3, 15, 8);
    for (i = 0; i < SbNum[Mm]; i++) SbMenuColor(i, 3, 15);
    SbMenuColor(Smm[Mm], 14, 4); /* 产生在菜单项的光条 */
}
```

/\* n: 菜单的第几项;

BCL: 背景颜色;

TCL: 前景颜色.

\*/

```

void SbMenuColor(int n,int BCl,int TCl)
{
    setfillstyle(1,BCl);
    bar(SbX[Mm]+3,30+n * Hj,SbX[Mm]+SbWid[Mm]-3,50+n * Hj);
    PutCC16(SbX[Mm]+10,32+n * Hj, 4,TCl,Sub[Mm][n]);
}

void SlctMenu(void)
{
    while(Key != ALT _X)
    {
        Key = GetKey();
        if (Key == LEFT || Key == RIGHT) SlctMnMenu();
        if (Key == UPPER || Key == DOWN) SlctSbMenu();
        if (Key == ENTER) SubFunGo();
    }
    return;
}

void SlctMnMenu(void)
{
    MnMenuColor(Mm,3,14);
    Wind (SbX[Mm],26,SbX[Mm]+SbWid[Mm]+10,SbNum[Mm] * Hj+44,0,0,1,1,
        8);
    if (Key == LEFT) Mm = Mm == 0 ? 3 : Mm-1;
    if (Key == RIGHT) Mm = Mm == 3 ? 0 : Mm+1;
    MnMenuColor(Mm,4,14);
    WrtSbMenu();
}

void SlctSbMenu(void)
{
    SbMenuColor(Smm[Mm],3,15);
    if (Key == UPPER) Smm[Mm] = Smm[Mm] == 0 ? SbNum[Mm]-1 :
        Smm[Mm]-1;
    if (Key == DOWN) Smm[Mm] = Smm[Mm] == SbNum[Mm]-1 ? 0 :
        Smm[Mm]+1;
    SbMenuColor(Smm[Mm],14,4);
}

int GetKey(void) /* 读取键值 */
{

```

```

int Ch,Low,Hig;
Ch = bioskey(0);
Low = Ch & 0x00ff;
Hig = (Ch & 0xff00) >> 8;
return (Low == 0 ? Hig+256 :Low);
}
void SubFunGo(void)
{
    switch(Mm)
    {
        case 0:
            switch(Smm[0])
            {
                /* 以下写入各功能对应的函数 */
                case 0: break;
                case 1: break;
                case 2: break;
                case 3: break;
                case 4: break;
            } break;
        case 1:
            switch(Smm[1])
            {
                case 0: break;
                case 1: break;
                case 2: break;
                case 3: break;
                case 4: break;
            } break;
        case 2:
            switch(Smm[2])
            {
                case 0: break;
                case 1: break;
                case 2: break;
            } break;
        case 3:
            switch(Smm[3])
            {
                case 0: break;
            }
    }
}

```

```

        case 1:    break;
    } break;
}
}

void Quit(void)
{
    closegraph();
    fcloseall();
    exit(0);
}

void Screen(void)
{
    char *Nam = "";
    char *Chs = "选择", *Ret = "返回", *Qut = "退出";
    char *Arr = "↑ ↓ ←→";
    setfillstyle(1,1);
    bar3d(0,0,639,479,0,0);
    setfillstyle(1,3);
    bar3d(0,0,639,25,0,0);
    bar3d(0,430,639,450,0,0);
    setfillstyle(1,6);
    bar3d(0,450,639,479,0,0); setcolor(RED);
    outtextxy(150,436,"Enter:");
    outtextxy(320,436,"ESC:");
    outtextxy(470,436,"Alt _ X");
    PutCC16(60, 432,4,4, Arr);
    PutCC16(220,432,4,14,Chs);
    PutCC16(370,432,4,14,Ret);
    PutCC16(530,432,4,14,Qut);/* PutCC24(100,453,4,2,Nam); */
}

/* x1,y1,x2,y2 分别为窗口左上角,右下角的坐标点;
   FrmTp: 边框类型: 0—不带边框, 1—单线边框, 2—双线边框;
   IsBk: 是否带阴影背景. 0—不带, 非 0—带阴影背景;
   BCl: 窗口背景颜色;
   TCl: 窗口文字颜色;
   BgCl: 背景阴影的颜色.
*/
void Wind(int x1,int y1,int x2,int y2,int FrmTp,int IsBk,int BCl,int TCl, \int BgCl)

```

```

{
    setfillstyle(1,BCl);
    bar(x1,y1,x2,y2);
    setcolor(TCl);
    if (FrmTp)                /* 边框线 */
    {
        rectangle(x1+2,y1+2,x2-2,y2-2);
        if (FrmTp == 2) rectangle(x1+4,y1+4,x2-4,y2-4);
    }
    if (IsBk)                  /* 阴影背景 */
    {
        setfillstyle(1,BgCl);
        bar(x1+7,y2+1,x2+7,y2+8);
        bar(x2+1,y1+7,x2+8,y2+8);
    }
}

void InitGr(void)
{
    int GraphDrive = DETECT,GraphMode;
    registerbgidriver(EGAVGA _driver);
    initgraph(&GraphDrive,&GraphMode,"");
}

void PutCC16(int x,int y,int Wid,int TCl,char * Str)
{
    unsigned Zcode,Bcode;      /* 区码,位码 */
    int i,j,k,Rec;
    long Len;
    char Buf[32];
    while (* Str)              /* 直到字符串显示完 */
    {
        if (( * Str & 0x80) && ( * (Str+1) & 0x80)) /* 是汉字 */
        {
            Zcode = ( * Str-0xa1) & 0x07f;          /* 区码 */
            Bcode = ( * (Str+1)-0xa1) & 0x07f;        /* 位码 */
            Rec = Zcode * 94+Bcode;                  /* 记录号 */
            Len = Rec * 32L;                          /* 在字库中位置 */
            fseek(fp16,Len,SEEK _SET);
            fread (Buf,1,32,fp16);                    /* 32 字节 */
            for (i = 0; i < 16; i++)                  /* 垂直方向 16 点 */

```

```

        for (j = 0; j < 2; j++)          /* 水平方向 2 字节 */
        for (k = 0; k < 8; k++)          /* 每字节 8 位 */
            if (Buf[i * 2 + j] >> (7 - k) & 1) /* 是 1 画点 */
                putpixel(x + j * 8 + k, y + i, TCL);
        x = x + 16 + Wid;
        Str += 2;
    }
}
return;
}
void OpenLIB(void)
{
    fp16 = fopen("c:/ucdos/hzkl6", "rb");
}

```

程序运行后屏幕显示如图 6-2 所示：

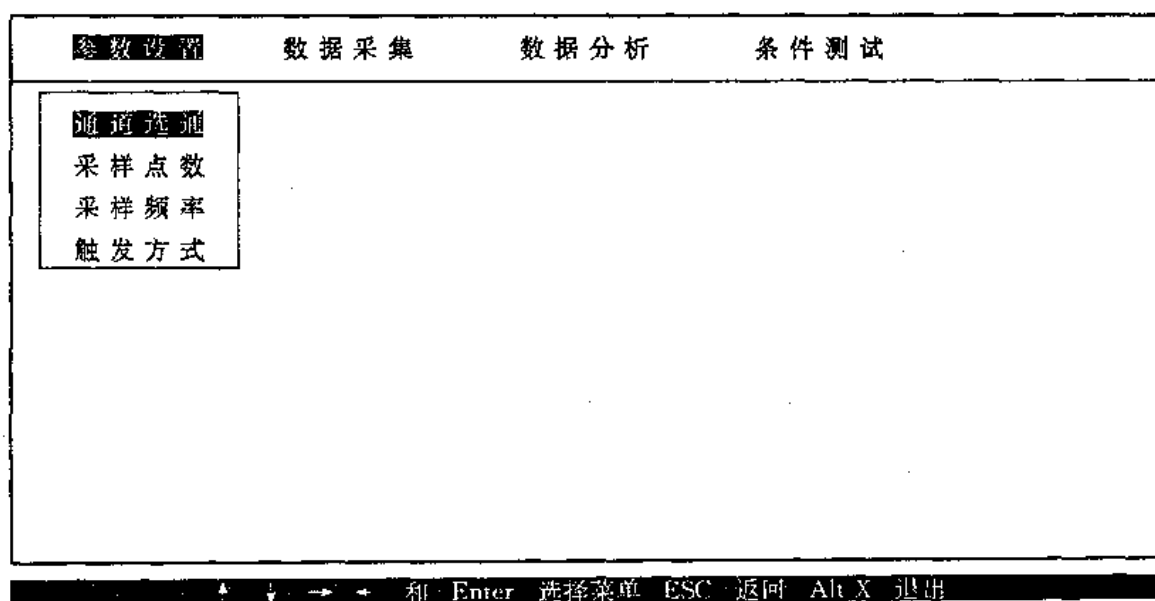


图 6-2

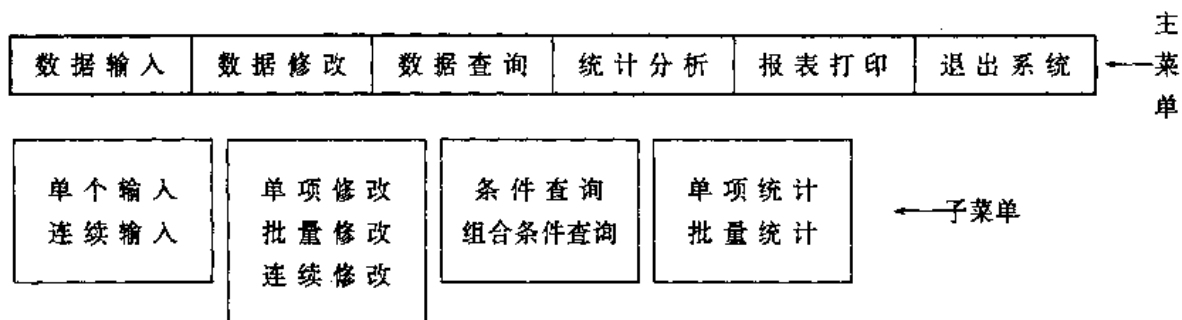
运行本程序可以看出，由于显示的每个汉字均是从字库中读取并显示的，所以在进行按键选择时速度显得有些慢，为了提高显示速度，可以采用建立小字库技术。

### 6.1.3 西文 DOS 下的汉字弹出式下拉菜单设计

本节介绍的也是一个下拉式菜单，跟上述两节介绍的不同之处在于，上两节所介绍的菜单的主菜单和子菜单栏同时显示，而本节介绍的菜单的子菜单栏可以根据需要弹出来，或者关闭。当子菜单栏没有弹出来时，可用 ENTER 键将其弹出（如果有子菜单）；而当子菜单栏弹出

时,又可用 ESC 键将其关闭。本节介绍的菜单中同时还采用了小字库技术,使得菜单在按键选择移动时速度明显得到提高。

本节菜单的结构如下:



源程序如下:

/\*

MENUP.C——弹出式下拉菜单

\*/

```

#define M_CC_CL    YELLOW    /* 主菜单汉字颜色 */
#define M_BK_CL    LIGHTBLUE /* 主菜单背景颜色 */
#define M_SL_CC_CL YELLOW    /* 主菜单选择光条汉字颜色 */
#define M_SL_BK_CL LIGHTRED  /* 主菜单选择光条背景颜色 */
#define S_CC_CL    YELLOW    /* 子菜单汉字颜色 */
#define S_BK_CL    CYAN      /* 子菜单背景颜色 */
#define S_SL_CC_CL YELLOW    /* 子菜单选择光条的汉字颜色 */
#define S_SL_BK_CL LIGHTRED  /* 子菜单选择光条的背景颜色 */
#define S_BORDLN_CL YELLOW   /* 子菜单边框线的颜色 */
#define FILLCL    LIGHTBLUE /* 主屏幕填充方式:密集点填充 */
#define LEFT      331
#define RIGHT     333
#define UPPER     328
#define DOWN      336
#define ALT_X     301
#define ESC       27
#define ENTER     13
#include "stdio.h"
#include "stdlib.h"
#include "alloc.h"
#include "graphics.h"
#include "process.h"
FILE *fp;
int Mn = 1;          /* 主菜单选项 */

```



```

int Key = 0;          /* 按键的键值 */
int Sub = 1;          /* 子菜单选项 */
int Lin = 0;          /* 各子菜单的菜单项数 */
int Num[6];           /* 各项子菜单的汉字个数 */
int Ast[6],y;
int St,St1;           /* 汉字在小字库中的起始位置 */
void InitGra();
void Start();
void MainMenu();
int GetKey();
void Quit();
void PutCC16(int x,int y,int St,int End,int Color);
void WrtMnMu();
void ClrWind(int x1,int y1,int x2,int y2);
void MnChis();
void SubChis(int x1,int x2);
void PopWin(int x1,int y1,int x2,int y2);
int main()
{
    InitGra();
    Start();
    MainMenu();
    Quit();
    return 0;
}
void InitGra()          /* 初始化屏幕为图形方式 */
{
    int GraphDrive = DETECT,GraphMode;
    registerbgidriver(EGAVGA_driver);
    initgraph(&GraphDrive,&GraphMode,"");
}
void Start()
{
    int i;
    setfillstyle(CLOSE_DOT_FILL,FILLCL);
    bar(0,0,639,449);
    setfillstyle(1,LIGHTBLUE);
    bar(2,1,636,32);
    setlinestyle(SOLID_LINE,0,THICK_WIDTH);

```

```

setcolor(LIGHTCYAN);
rectangle(1,0,637,449);
line(1,33,637,33);
for (i = 1;i < 7;i++) line(1+i * 106 ,0,1+i * 106,33);
fp = fopen ("menu.dat","rb");
}

void MainMenu()
{
    int i,Siz,IsESC;
    int x1,y1 = 35,x2,y2;
    WrtMnMu();
    for(;;)
    {
        IsESC = 0;
        while(Key != ENTER)
        {
            Key = GetKey();
            if (Key == LEFT || Key == RIGHT) MnChis();/* ←、→键选择主菜单 */
        }
        if (Mn == 6) Quit();
        if (Mn == 5) {      } /* 主菜单第5项 */
        for (;;)          /* 弹出子菜单 */
        {
            switch(Mn)
            {
                case 1:
                    x1 = 3;x2 = 114;y2 = 146;
                    Lin = 2;
                    St = 24;
                    Num[0] = Num[1] = 4;
                    break;
                case 2:
                    x1 = 108;x2 = 228;y2 = 186;
                    Lin = 3;
                    St = 32;
                    Num[0] = Num[1] = Num[2] = 4;
                    break;
                case 3:

```

```

        x1 = 214;x2 = 348;y2 = 146;
        Lin = 2;
        St = 44;
        Num[0] = 4;Num[1] = 6;
        break;
    case 4:
        x1 = 320;x2 = 438;y2 = 146;
        Lin = 2;
        St = 54;
        Num[0] = Num[1] = 4;break;
}
y = y1+26;
Sub = 1;
if (Mn != 5 && Mn != 6) PopWin(x1,y1,x2,y2);
/* 第5项和第6项 无子菜单 */

Key = 0;
y = y1+26;
while (Key != ENTER && Key != ESC)
{
    Key = GetKey();
    if (Key == RIGHT || Key == LEFT)
    {
        ClrWind(x1,y1,x2,y2);
        MnChis();
        break;
    }
    if (Key == UPPER || Key == DOWN) SubChis(x1,x2);
    if (Key == ENTER)
    {
        if (Mn == 6) Quit();          /* 根据选择执行相应功能 */
/*
        if (Mn == 5) Sub5();
        if (Mn == 1 && Sub == 1) Sub11();
        if (Mn == 1 && Sub == 2) Sub12();
        if (Mn == 2 && Sub == 1) Sub21();
        if (Mn == 2 && Sub == 2) Sub22();
        if (Mn == 2 && Sub == 3) Sub23();
        if (Mn == 3 && Sub == 1) Sub31();
        if (Mn == 3 && Sub == 2) Sub32();

```

```

        if (Mn == 4 && Sub == 1) Sub41();
        if (Mn == 4 && Sub == 2) Sub42(); /*
        Key = 0;
    }
    if (Key == ESC)
    {
        ClrWind(x1,y1,x2,y2);
        IsESC = 1;
    }
    if (IsESC) break;
}
if (IsESC) break;
}
}
}

```

```

int GetKey()

```

```

{
    int Ch,Low,Hig;
    Ch = bioskey(0);
    Low = Ch & 0x00ff;
    Hig = (Ch & 0xff00) >> 8;
    return (Low == 0 ? Hig+256 :Low);
}

```

```

void Quit()

```

```

{
    closegraph();
    fcloseall();
    exit(0);
}

```

```

/* x,y:要显示汉字的起点坐标;

```

```

    St:汉字在小字库中的起始位置;

```

```

    Ed:汉字在小字库中的终止位置;

```

```

    Color:汉字的颜色.

```

```

*/

```

```

void PutCC16(int x,int y,int St,int End,int Color)

```

```

{
    int i,i1,i2,i3;
    char by[32];
    for (i = St;i < End;i++)

```

```

    {
        fseek(fp, (long) (i * 32), SEEK_SET);
        fread(by, 1, 32, fp);
        for (i1 = 0; i1 < 16; i1++)
            for (i2 = 0; i2 < 2; i2++)
                for (i3 = 0; i3 < 8; i3++)
                    if (by[i1 * 2 + i2] >> (7 - i3) & 1)
                        putpixel(x + i2 * 8 + i3, y + i1, Color);
                        x = x + 17;
    }
}

void WrtMnMu()      /* 显示主菜单 */
{
    int i;
    for (i = 0; i < 6; i++) PutCC16(20 + i * 106, 8, i * 4, i * 4 + 4, M_CC_CL);
    setfillstyle(1, M_SL_BK_CL);
    bar(3, 2, 105, 31);
    PutCC16(20, 8, 0, 4, M_SL_CC_CL);
}

void ClrWind(int x1, int y1, int x2, int y2)
{
    setfillstyle(CLOSE_DOT_FILL, FILLCL);
    bar(x1, y1, x2, y2);
}

void MnChis()      /* 主菜单选择 */
{
    setfillstyle(1, M_BK_CL);
    bar((Mn - 1) * 106 + 3, 2, (Mn - 1) * 106 + 105, 31);
    PutCC16(20 + (Mn - 1) * 106, 8, (Mn - 1) * 4, (Mn - 1) * 4 + 4, M_CC_CL);
    if (Key == RIGHT) Mn = Mn == 6 ? 1 : Mn + 1;
    if (Key == LEFT) Mn = Mn == 1 ? 6 : Mn - 1;
    setfillstyle(1, M_SL_BK_CL);
    bar((Mn - 1) * 106 + 3, 2, (Mn - 1) * 106 + 105, 31);
    PutCC16(20 + (Mn - 1) * 106, 8, (Mn - 1) * 4, (Mn - 1) * 4 + 4, M_SL_CC_CL);
}

void SubChis(int x1, int x2)      /* 子菜单选择 */
{
    setfillstyle(1, S_BK_CL);
    bar(x1 + 16, y, x2 - 16, y + 20);
}

```

```

PutCC16(x1+18,y+2,Ast[Sub-1],Ast[Sub-1]+Num[Sub-1],S_CC_CL);
if (Key == UPPER) Sub = Sub == 1 ? Lin :Sub-1;
if (Key == DOWN) Sub = Sub == Lin ? 1 :Sub+1;
y = Sub * 40+20;
setfillstyle(1,S_SL_BK_CL);
bar(x1+16,y,x2-16,y+20);
PutCC16 (x1 + 18, y + 2, Ast [ Sub - 1 ], Ast [ Sub - 1 ] + Num [ Sub - 1 ],
        S_SL_CC_CL);
}

void PopWin(int x1,int y1,int x2,int y2)          /* 弹出窗口 */
{
    int i;
    setfillstyle(1,S_BK_CL);
    bar(x1,y1,x2,y2);
    St1 = St;
    setlinestyle(0,0,1);
    setcolor(S_BORDLN_CL);
    rectangle(x1+5,y1+5,x2-5,y2-5);
    for(i = 0;i < Lin;i++)
    {
        PutCC16(x1+18,i * 40+62,St1,St1+Num[i],14);
        Ast[i] = St1; St1 = St1+Num[i];
    }
    setfillstyle(1,S_SL_BK_CL);
    bar(x1+16,y,x2-16,y+20);
    PutCC16 (x1 + 18, y + 2, Ast [ Sub - 1 ], Ast [ Sub - 1 ] + Num [ Sub - 1 ],
            S_SL_CC_CL);
}

```

程序中的汉字是由小字库文件 MENU.DAT 中读出的,该小字库的建立是通过第三章中介绍的 CR16.C 来建立的,只要将 CR16.C 中的 Str 字符串改为:

\* Str = “数据输入数据修改数据查询统计分析报表打印退出系统单个输入连续输入单  
项修改批量修改连续修改条件查询组合条件查询单项统计批量统计”;

即可。

程序运行后屏幕上的显示如图 6-3。

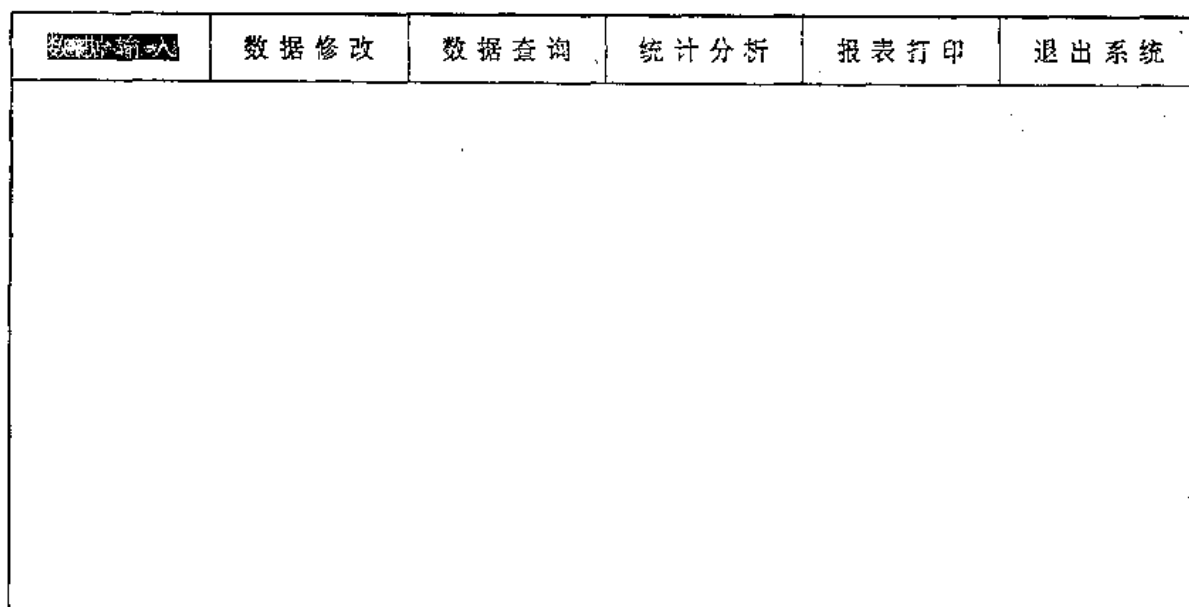


图 6-3

按→←键时,光条作右、左移动,当按下 ENTER 键时,则弹出该项对应的子菜单栏;假定光条在“数据修改”项上时,按 ENTER 键,则弹出子菜单栏后的屏幕显示如图 6-4。

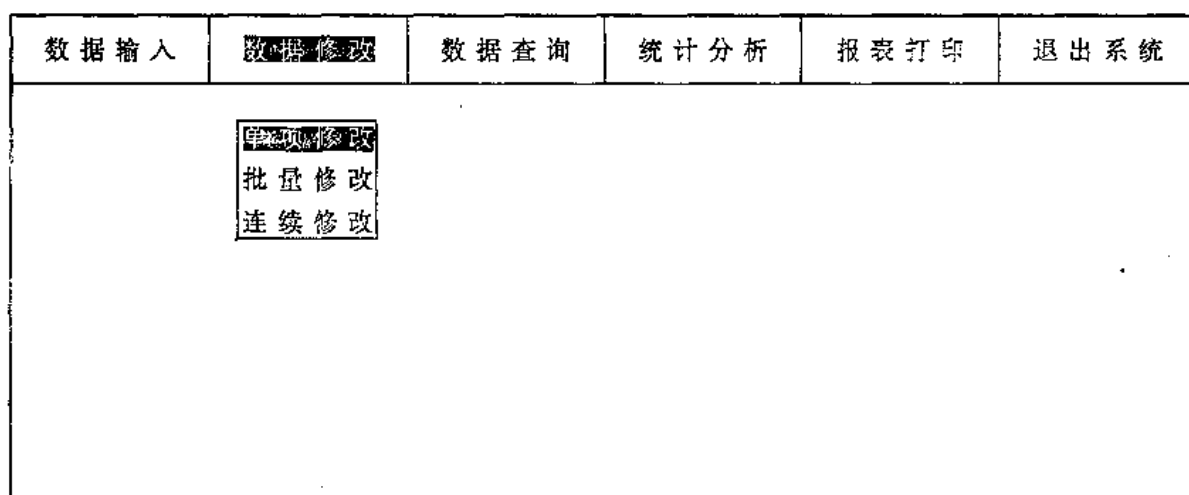


图 6-4

这时,按→←键,光条作右左移动,其对应的子菜单栏随之移动,按↑↓键时,光条在子菜单栏内的子菜单项上作上下移动,当按回车后,执行相应的功能,若按 ESC 键,则关闭子菜单栏,同上图相同。

## 6.2 中文 DOS 下的汉字下拉式菜单设计

以上介绍的都是西文 DOS 下的菜单设计,有时用户还希望在汉字 DOS 下开发软件,本节

我们介绍中文 DOS 下的下拉式菜单设计。

对 UC DOS 3.0 而言,在中文字符方式下,其显示方式同在西文字符方式是相同的。因此,只要将 MENU.C 中要显示的西文字串改为汉字即可,为帮助读者阅读方便起见,本节再将其列出,供读者参考:

/\*

MENU0.c——中文 DOS 下的汉字下拉式菜单设计

\*/

#include "graphics.h"

#include "conio.h"

#include "bios.h"

#include "process.h"

#include "string.h"

#include "stdio.h"

#define LEFT 331 /\* 功能键的宏定义 \*/

#define RIGHT 333

#define UPPER 328

#define DOWN 336

#define ALT \_X 301

#define ESC 27

#define ENTER 13

void WrtMnMenu(void);

void WrtSbMenu(void);

void SlctMenu(void);

void SlctMnMenu(void);

void SlctSbMenu(void);

int GetKey(void);

void SbFunGo(void);

void Screen(void);

void Wind(int,int,int,int,int,int,int,int,int);

void Quit(void);

int Mm = 0; /\* 主菜单选择序号 \*/

int Smm[4]; /\* 各子菜单选择序号 \*/

int SbNum[4] = {6,6,3,3}; /\* 各子菜单项数 \*/

int SbWid[4] = {9,9,9,11}; /\* 各子菜单栏宽度 \*/

int SbX[4] = {5,20,33,47}; /\* 各子菜单栏 X 坐标 \*/

int Key = 0; /\* 选择键 \*/

char Buf[1000];

char \* Main[4] = {"参数设置","数据采集","数据分析","条件测试"};

char \* Sub[4][6] = {{ "通道序号",



```

        "通道选通",
        "采样点数",
        "采样频率",
        "触发方式",
        "初始校准"},
    {"开始采样",
     "查看数据",
     "显示波形",
     "实时波形",
     "直方图形",
     "实时方图"
    },
    {"结果分析",
     "结果数据",
     "结果波形",
    },
    {"试验条件 1",
     "试验条件 2",
     "试验条件 3"
    }
    });

```

```
int main(void)
```

```
{
```

```
    Screen();
```

```
    WrtMnMenu();
```

```
    WrtSbMenu();
```

```
    SlctMenu();
```

```
    Quit();
```

```
    return(0);
```

```
}
```

```
void WrtMnMenu(void)
```

```
{
```

```
    int i;
```

```
    window(1,1,80,25);
```

```
    /* 显示主菜单 */
```

```
    textattr(0x3e);
```

```
    for (i = 0; i < 4; i++)
```

```
    {
```

```
        gotoxy(SbX[i],1);
```

```
        cputs(Main[i]);
```

```
    }
```

```

    gotoxy(SbX[Mm],1);          /* 设置光条颜色 */
    textattr(0x4e);
    cputs(Main[Mm]);
}
void WrtSbMenu(void)
{
    int i;
    gettext(SbX[Mm]-2,2,SbX[Mm]+SbWid[Mm]+1,SbNum[Mm]+4,Buf);
    Wind(SbX[Mm]-2,2,SbX[Mm]+SbWid[Mm],SbNum[Mm]+3,1,1,3,15,1);
    textattr(0x3f);             /* 显示子菜单 */
    for (i = 0;i < SbNum[Mm];i++)
    {
        gotoxy(2,1+i);
        cputs(Sub[Mm][i]);
    }
    textattr(0x1e);             /* 子菜单选项的颜色 */
    gotoxy(2,Smm[Mm]+1);
    cputs(Sub[Mm][Smm[Mm]]);
}

void SlctMenu(void)             /* 执行菜单选择功能 */
{
    while(Key != ALT _X && Key != ESC)
    {
        Key = GetKey();
        if (Key == LEFT || Key == RIGHT) SlctMnMenu();
        if (Key == UPPER || Key == DOWN) SlctSbMenu();
        if (Key == ENTER)      SbFunGo();
    }
    return;
}

void SlctMnMenu(void)
{
    window(1,1,80,25);
    textattr(0x3e);
    gotoxy(SbX[Mm],1);
    cputs(Main[Mm]);
    textattr(0x31);

```

```

    puttext(SbX[Mm]-2,2,SbX[Mm]+SbWid[Mm]+1,SbNum[Mm]+4,Buf);
    if (Key == LEFT) Mm = Mm == 0 ? 3 : Mm-1;
    if (Key == RIGHT) Mm = Mm == 3 ? 0 : Mm+1;
    textattr(0x4e);
    gotoxy(SbX[Mm],1);
    cputs(Main[Mm]);
    WrtSbMenu();
}

void SlctSbMenu(void)
{
    textattr(0x3f);
    gotoxy(2,1+Smm[Mm])(Key == DOWN) Smm[Mm] = Smm[Mm] == SbNum
        [Mm]-1 ? 0 : Smm[Mm]+1;
    textattr(0x1e);
    gotoxy(2,Smm[Mm]+1);
    cputs(Sub[Mm][Smm[Mm]]);
}

int GetKey(void)
{
    int Ch,Low,Hig;
    Ch = bioskey(0);
    Low = Ch & 0x00ff;
    Hig = (Ch & 0xff00) >> 8;
    return (Low == 0 ? Hig+256 : Low);
}

void SbFunGo(void) /* 根据键选执行相应的功能 */
{
    switch(Mm)
    {
        case 0:
            switch(Smm[0])
            {
                case 0: break;
                case 1: break;
                case 2: break;
                case 3: break;
                case 4: break;
                case 5: break;
                case 6: break;
            }
        case 1:
            switch(Smm[1])
            {
                case 0: break;
                case 1: break;
                case 2: break;
                case 3: break;
                case 4: break;
                case 5: break;
                case 6: break;
            }
        case 2:
            switch(Smm[2])
            {
                case 0: break;
                case 1: break;
                case 2: break;
                case 3: break;
                case 4: break;
                case 5: break;
                case 6: break;
            }
        case 3:
            switch(Smm[3])
            {
                case 0: break;
                case 1: break;
                case 2: break;
                case 3: break;
                case 4: break;
                case 5: break;
                case 6: break;
            }
    }
}

```

```

        } break;
    case 1:
        switch(Smm[1])
        {
            case 0:    break;
            case 1:    break;
            case 2:    break;
            case 3:    break;
            case 4:    break;
            case 5:    break;
        } break;
    case 2:
        switch(Smm[2])
        {
            case 0:    break;
            case 1:    break;
            case 2:    break;
            case 3:    break;
        } break;
    case 3:
        switch(Smm[3])
        {
            case 0:    break;
            case 1:    break;
            case 2:    break;
        } break;
    }
}

```

```

void Screen(void)

```

```

{
    window(1,1,80,25);
    textattr(0x17);
    clrscr();
    window(1,1,80,1);
    textattr(0x3e);
    clrscr();
    window(1,24,80,24);
    textattr(0x74);
    clrscr();

```

```

cputs(" ↑ ↓ ← →");
textattr(0x7e); cputs(" 和 ");
textattr(0x74); cputs("Enter ");
textattr(0x7e); cputs("选择菜单 ");
textattr(0x74); cputs("Alt _X ");
textattr(0x7e); cputs(" 或 ");
textattr(0x74); cputs(" ESC ");
textattr(0x7e); cputs("退出程序");
window(1,25,80,25);
textattr(0x6A);
clrscr();
cputs("BEIJING AGRICULTURAL ENGINEERING UNIVERSITY & SHAN DONG
      COLLEGE\OF ENGINEERING");
Wind(1,2,80,23,2,0,1,15,1);
window(1,1,80,25);
}
/* 字符窗口函数:
   x1,y1:窗口左上角坐标;
   x2,y2:窗口右下角坐标;
   FrmTp:窗口边框类型;0—无边框;1—单线边框;2—双线边框;
   IsBk:是否带阴影背景;
   BCl:字符颜色;
   TCl:背景颜色;
   BgCl:阴影背景的颜色.
*/
void Wind(int x1,int y1,int x2,int y2,int FrmTp,int IsBk,int BCl,int TCl,\int BgCl)
{
    int i;
    int c[2][6]={ {0xda,0xc4,0xbf,0xb3,0xc0,0xd9}, /* 单线字符 */
                  {0xc9,0xcd,0xbb,0xba,0xc8,0xbc}}; /* 双线字符 */
    textcolor(TCl);
    textbackground(BCl);
    window(x1,y1,x2,y2);
    clrscr();
    if (FrmTp) /* 有边框线 */
    {
        window(1,1,80,25);
        gotoxy(x1,y1);
        putch(c[FrmTp-1][0]);
    }
}

```

```

    for (i = x1+1; i < x2; i++)
        putchar(c[FrmTp-1][1]);
    putchar(c[FrmTp-1][2]);
    for (i = y1+1; i < y2; i++)
    {
        gotoxy(x1,i);
        putchar(c[FrmTp-1][3]);
        gotoxy(x2,i);
        putchar(c[FrmTp-1][3]);
    }
    gotoxy(x1,y2);
    putchar(c[FrmTp-1][4]);
    for (i = x1+1; i < x2; i++)
        putchar(c[FrmTp-1][1]);
    putchar(c[FrmTp-1][5]);
}
if (IsBk)          /* 有阴影背景 */
{
    textcolor(BgCl);
    textbackground(0);
    for (i = y1+1; i < y2+1; i++)
    {
        gotoxy(x2+1,i);
        putchar('\xb1');
    }
    for (i = x1+1; i < x2+2; i++)
    {
        gotoxy(i,y2+1);
        putchar('\xb1');
    }
}
window(x1+1,y1+1,x2-1,y2-1);
}

void Quit(void)
{
    textbackground(0);
    textcolor(7);
    window(1,1,80,25);
    clrscr();
}
• 150 •

```

```

    exit(0);
}

```

## 6.3 用鼠标选择立体按钮菜单

在当今流行的许多软件如 WINDOWS 等,都采用了立体按钮菜单,感觉非常形象,令人耳目一新,本书在第二章中介绍了立体块的设计,结合第五章中介绍的关于鼠标的函数,本节将介绍利用鼠标进行立体按钮菜单的设计。

在第二章中已介绍,立体块的设计是通过同一颜色在不同亮度上的差别来实现的,假设中间大面积区域的颜色为 ColorM,左边和上边的颜色为高亮度 ColorH,右侧和下侧的低亮度颜色为 ColorL,为了实现逼真的按钮动感设计,可以按下面的过程进行:

当鼠标选中某项功能后,

- (1) 保存整个立体块;
- (2) 以 ColorM 填充整个矩形区域;
- (3) 延时 100 毫秒;
- (4) 恢复保存的立体块;
- (5) 延时 100 毫秒。

具体实现中,可用 line()函数,也可用 getimage()和 putimage()函数,本例中采用的是后者,参看程序 MsMenu.C 中的 PrsBox()函数,比较简单,读者可直接采用。

```

/*
  MsMenu.c——用鼠标选择立体按钮菜单
*/
#include "stdio.h"
#include "stdlib.h"
#include "graphics.h"
#include "dos.h"
#define LEFTBTN 1
FILE *fp;
void Quit();
void MainMenu();
void InitGra(void);
void PutCC16(int x,int y,int,int Color,char * Str);
void PrsBox(int x,int y,int l,int h);
void Box(int x,int y,int l,int h);
int InitMouse();
void ErrMsg();
void ShowMouse();
void HideMouse();

```

```

void ReadMouse(int *f,int *x,int *y);
int MsInBox(int x1,int y1,int x2,int y2,int x,int y);
void OpenLIB();
int main()
{
    InitGra();
    OpenLIB();
    MainMenu();
    Quit();
    return 0;
}
void Quit()
{
    closegraph();
    fcloseall();
    exit(0);
}
void MainMenu()
{
    int x1,y1,l,h,x2;
    int Button,x,y,Choice;
    char *Tit = "用鼠标选择菜单";
    char *Mmu[] = {"绘制新图","编辑旧图",
                   "图形输出","退出系统"};

    x1 = 190;
    y1 = 240;
    x2 = 350;
    l = 100;
    h = 32;
    setfillstyle(1,7);
    bar(140,30,500,460);
    setcolor(14);
    rectangle(140,30,500,460);
    setfillstyle(1,9);
    bar(215,100,430,147);
    PutCC16(250,116,4,15,Tit);          /* 画出菜单画面 */
    Box(x1,y1,l,h);
    PutCC16(x1+15,y1+8,2,1,Mmu[0]);    /* 显示菜单的项 */
    Box(x2,y1,l,h);

```



```

PutCC16(x2+15,y1+8,2,1,Mmu[1]);
Box(x1,y1+100,l,h);
PutCC16(x1+15,y1+108,2,1,Mmu[2]);
Box(x2,y1+100,l,h);
PutCC16(x2+15,y1+108,2,1,Mmu[3]);
if (! InitMouse()) ErrMsg();          /* 如果没安装鼠标,则退出 */
ShowMouse();
for (;;)
{
    Button = 0;
    while (Button != LEFTBTN) ReadMouse(&Button,&x,&y);/* 等待鼠标按键 */
    HideMouse();
    if (MsInBox(x1,y1,x1+l,y1+h,x,y)) /* 判断选择的是哪一个功能 */
    {
        PrsBox(x1,y1,l,h);
        Choice = 1;break;
    }
    if (MsInBox(x2,y1,x2+l,y1+h,x,y))
    {
        PrsBox(x2,y1,l,h);
        Choice = 2;break;
    }
    if (MsInBox(x1,y1+100,x1+l,y1+100+h,x,y))
    {
        PrsBox(x1,y1+100,l,h);
        Choice = 3;break;
    }
    if (MsInBox(x2,y1+100,x2+l,y1+100+h,x,y))
    {
        PrsBox(x2,y1+100,l,h);
        Choice = 4;break;
    }
}
switch(Choice)                        /* 根据选择执行相应的功能 */
{
    case 1;break;
    case 2;break;
    case 3;break;
    case 4:Quit();
}

```

```

    }
}

void InitGra(void) /* 初始化屏幕为图形方式 */
{
    int GraphDrive = DETECT, GraphMode;
    registerbgidriver(EGAVGA_driver);
    initgraph(&GraphDrive, &GraphMode, "");
}

void PutCC16(int x, int y, int z, int Color, char *Str) /* 显示 16 点阵汉字 */
{
    unsigned Zcode, Bcode; /* 区码, 位码 */
    int i, j, k, Rec;
    long Len;
    char Buf[32];
    while (*Str) /* 直到字符串显示完 */
    {
        if ((*Str & 0x80) && (*(Str+1) & 0x80)) /* 是汉字 */
        {
            Zcode = (*Str - 0xa1) & 0x07f; /* 区码 */
            Bcode = (*(Str+1) - 0xa1) & 0x07f; /* 位码 */
            Rec = Zcode * 94 + Bcode; /* 记录号 */
            Len = Rec * 32L; /* 在字库中位置 */
            fseek(fp, Len, SEEK_SET);
            fread(Buf, 1, 32, fp); /* 32 字节 */
            for (i = 0; i < 16; i++)
                for (j = 0; j < 2; j++)
                    for (k = 0; k < 8; k++)
                        if (Buf[i * 2 + j] >> (7 - k) & 1)
                            putpixel(x + j * 8 + k, y + i, Color);
            x = x + 16 + z;
            Str += 2;
        }
    }
    return;
}

void PrsBox(int x, int y, int l, int h) /* 立体块"按下", 并"弹起" */
{
    void *Buf1, *Buf2;
    Buf1 = malloc(imagesize(x, y, x + l, y + h));
}

```

```

    Buf2 = malloc(imagesize(x+4,y+4,x+l-4,y+h-4));
    getimage(x,y,x+l,y+h,Buf1);
    getimage(x+4,y+4,x+l-4,y+h-4,Buf2);
    setfillstyle(1,7);
    bar(x,y,x+l,y+h);
    putimage(x+4,y+4,Buf2,COPY_PUT);
    delay(100);
    putimage(x,y,Buf1,COPY_PUT);
    delay(100);
    free(Buf1);
    free(Buf2);
}

void Box(int x,int y,int l,int h)    /* 画一立体块 */
{
    setcolor(0);
    rectangle(x-1,y-1,x+l+1,y+h+1);
    setfillstyle(1,8);
    bar(x,y,x+l,y+h);
    setfillstyle(1,7);
    bar(x+2,y+2,x+l-2,y+h-2);
    setcolor(15);
    line(x,y,x+l,y);
    line(x,y,x,y+h);
    line(x+1,y+1,x+l-1,y+1);
    line(x+1,y+1,x+1,y+h-1);
}

int InitMouse()
{
    union REGS Inr,Outr;
    Inr.x.ax = 0;
    int86(0x33,&Inr,&Outr);
    return Outr.x.ax;
}

void ShowMouse()                    /* 显示鼠标的光标 */
{
    union REGS Inr;
    Inr.x.ax = 1;
    int86(0x33,&Inr,&Inr);
}

```

```

void HideMouse()          /* 隐去鼠标的光标,以便在屏幕上输出信息 */
{
    union REGS Inr;
    Inr.x.ax = 2;
    int86(0x33,&Inr,&Inr);
}

void ReadMouse(int *f,int *x,int *y)
    /* 读取鼠标的按键信息,并返回按键时鼠标光标的屏幕坐标 */
{
    union REGS Inr,Outr;
    Inr.x.ax = 3;
    int86(0x33,&Inr,&Outr);
    *f = Outr.x.bx;
    *x = Outr.x.cx;
    *y = Outr.x.dx;
}

/*
    判断坐标(x,y)是否在以(x1,y1)为左上角,以(x2,y2)为右下角的矩形区域之中
*/
int MsInBox(int x1,int y1,int x2,int y2,int x,int y)
{
    return((x >= x1 && x <= x2 && y >= y1 && y <= y2) ? 1 : 0);
}

void OpenLIB()            /* 打开 16 点阵汉字库 */
{
    fp = fopen("c:\\ucdos\\hzk16", "
rb");
}

void ErrMsg()
{
    printf("No Mouse Error!");
    getch();
    Quit();
}

```

程序运行后的屏幕如图 6-5 所示。

当移动鼠标时,光标箭头作相应的移动,当光标在某一个菜单项上时,按鼠标左键即执行相应的功能。

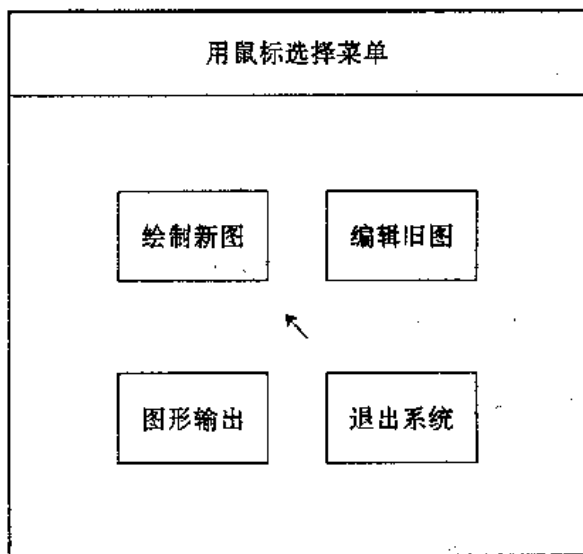


图 6-5

## 6.4 鼠标和键盘兼容的下拉式菜单设计

在 6.1 和 6.2 中我们介绍了用键盘选择菜单的程序设计,在 6.3 中又介绍了用鼠标选择按钮菜单。在本节中,我们介绍的是如何将键盘和鼠标融和在一起,来设计二者兼容的下拉式菜单。要使二者兼容,最关键的一点是如何能同时读取键盘和鼠标的键值。以 MENU.C 为例,函数 SlctMenu()是读取功能键的,只要将该函数中加入判断所按的是哪一类的键即可,对键盘上的按键可用 kbhit()函数来检测,当其为 TRUE 时表示有键按下了,否则是有鼠标的键按下了,如果没有任何键按下,则一直在等待。修改后的 SlctMenu()函数如下:

```
void SlctMenu(void)
{
    int Button = 0,mx,my;
    for (;;)
    {
        if (kbhit()) KeySlct();          /* 如果是键盘的键按下,执行键盘选择功能 */
        ReadMouse(&Button,&mx,&my); /* 否则,读取鼠标的按键信息 */
        if (Button == LEFTBTN)
            MouseSlct(mx,my);           /* 鼠标的左键按下,执行鼠标选择功能 */
        if (Button == RIGHTBTN) Quit();  /* 鼠标右键按下,退出程序系统 */
    }                                     /* 如果什么键也没有按下,则一直循环等待 */
}
```

另外,还要增加一个 MouseSlct()函数,用来处理鼠标的选择。函数有两个入口参数,分别是鼠标左键被按下时光标的屏幕行坐标和列坐标。该函数要做两个方面的处理:如果光标是在主菜单区域内,则判断光标在哪一项主菜单内,然后主菜单项和其子菜单栏将移到该项;如果光标是在子菜单区,则判断所选定的是哪一个主菜单项对应的哪一条子菜单项,转向执行相应的功能,执行完后返回。MouseSlct()函数如下:

```
void MouseSlct(int mx,int my)
{
    int i;
    HideMouse();
    if (my < 25)          /* 主菜单区域 */
    {
        MinMenuColor(Mm,3,14);
        Wind (SubX[Mm],26,SubX[Mm]+SubWid[Mm]+10,SubNum[Mm]*Hj+44,0,
              0,1,1);
        for (i = 0;i < 4;i++)
            if (MsInBox(SubX[i],1,SubX[i]+SubWid[Mm],24,mx,my)) Mm = i;
        MinMenuColor(Mm,4,14);
    }
```

```

        WrtSubMenu();
    }
    else /* 子菜单区域 */
    {
        SubMenuColor(Smm[Mm],3,15);
        for (i = 0;i < SubNum[Mm];i++)
            if (MsInBox(SubX[Mm]+3,30+i * Hj,SubX[Mm]+SubWid[Mm]-3,50+i *
                Hj,\mx,my))
                Smm[Mm] = i;
        SubMenuColor(Smm[Mm],14,4);
        SubFunGo();
    }
    ShowMouse();
    return;
}

```

下面是完整的键盘和鼠标兼容的下拉式菜单的源程序:

```

/*
    MsKyMu.C——鼠标和键盘兼容的西文DOS下的汉字下拉菜单
*/
#include "graphics.h"
#include "string.h"
#include "bios.h"
#include "dos.h"
#include "conio.h"
#include "process.h"
#include "stdio.h"
#include "stdlib.h"
#define LEFT      331
#define RIGHT     333
#define UPPER     328
#define DOWN      336
#define ALT _X    301
#define ESC       27
#define ENTER     13
#define LEFTBTN   1
#define RIGHTBTN  2
void WrtMnMenu(void);
void MinMenuColor(int n,int BkColor,int ForColor);
void WrtSbMenu(void);

```

```

void SbMenuColor(int n,int BkColor,int ForColor);
void SlctMenu(void);
void KeySlct();
void MouseSlct(int,int);
void SlctMnMenu(void);
void SlctSbMenu(void);
int GetKey(void);
void SbFunGo(void);
void Quit(void);
void Screen(void);
void Wind(int,int,int,int,int,int,int,int);
void InitGraph(void);
void PutCC16(int,int,int,int,char * Str);
void OpenLIB(void);
int InitMouse();
void ShowMouse();
void HideMouse();
void ReadMouse(int *f,int *x,int *y);
int MsInBox(int x1,int y1,int x2,int y2,int x,int y);
FILE *fp16;
int Hj = 20;
int Mm = 0,Smm[4];
int SbNum[4] = {4,4,3,2};
int SbWid[4] = {93,93,93,90};
int SbX[4] = {20,140,270,400};
int Key = 0;
char *Main[4] = {"参数设置","数据采集","数据分析","条件测试"};
char *Sub[4][4] = {"通道选通",
                  "采样点数",
                  "采样频率",
                  "触发方式"},
{"数据采集",
"显示数据",
"显示波形",
"初始校准"},
{"结果分析",
"结果数据",
"结果波形"},
{"条件一",

```

“条件二”} );

```
int main(void)
```

```
{
    OpenLIB();
    InitGraph();
    Screen();
    WrtMnMenu();
    WrtSbMenu();
    InitMouse();
    ShowMouse();
    SlctMenu();
    Quit();
    return(0);
}
```

```
void WrtMnMenu(void)
```

```
{
    int i;
    for(i = 0; i < 4; i++) MinMenuColor(i, 3, 14);
    MinMenuColor(Mm, 4, 14);
}
```

```
void MinMenuColor(int n, int BkColor, int ForColor)
```

```
{
    setfillstyle(1, BkColor);
    bar(SbX[n], 1, SbX[n] + SbWid[Mm], 24);
    PutCC16(SbX[n] + 10, 5, 4, ForColor, Main[n]);
}
```

```
void WrtSbMenu(void)
```

```
{
    int i;
    Wind(SbX[Mm], 26, SbX[Mm] + SbWid[Mm], SbNum[Mm] * Hj + 34, 1, 1, 3, 15);
    for (i = 0; i < SbNum[Mm]; i++) SbMenuColor(i, 3, 15);
    SbMenuColor(Smm[Mm], 14, 4);
}
```

```
void SbMenuColor(int n, int BkColor, int ForColor)
```

```
{
    setfillstyle(1, BkColor);
    bar(SbX[Mm] + 3, 30 + n * Hj, SbX[Mm] + SbWid[Mm] - 3, 50 + n * Hj);
    PutCC16(SbX[Mm] + 10, 32 + n * Hj, 4, ForColor, Sub[Mm][n]);
}
```



```

void SlctMenu(void)
{
    int Button = 0, mx, my;
    for (;;)
    {
        if (kbhit()) KeySlct();          /* 如果是键盘的键按下 */
        ReadMouse(&Button, &mx, &my); /* 读取鼠标的按键信息 */
        if (Button == LEFTBTN) MouseSlct(mx, my);
        if (Button == RIGHTBTN) Quit(); /* 鼠标右键按下, 退出 */
    }
}

void KeySlct()
{
    Key = GetKey();
    if (Key == LEFT || Key == RIGHT) SlctMnMenu();
    if (Key == DOWN || Key == UPPER) SlctSbMenu();
    if (Key == ENTER) SbFunGo();
    if (Key == ALT _X) Quit();
}

void MouseSlct(int mx, int my)
{
    int i;
    HideMouse();
    if (my < 25) /* 主菜单区域 */
    {
        MinMenuColor(Mm, 3, 14);
        Wind (SbX[Mm], 26, SbX[Mm] + SbWid[Mm] + 10, SbNum[Mm] * Hj + 44, 0, 0, 1,
              1);
        for (i = 0; i < 4; i++)
            if (MsInBox(SbX[i], 1, SbX[i] + SbWid[Mm], 24, mx, my)) Mm = i;
        MinMenuColor(Mm, 4, 14);
        WrtSbMenu();
    }
    else /* 子菜单区域 */
    {
        SbMenuColor(Smm[Mm], 3, 15);
        for (i = 0; i < SbNum[Mm]; i++)
            if (MsInBox(SbX[Mm] + 3, 30 + i * Hj, SbX[Mm] + SbWid[Mm] - 3, 50 + i * Hj, mx,

```

```

        my))
        Smm[Mm] = i;
        SbMenuColor(Smm[Mm],14,4);
        SbFunGo();
    }
    ShowMouse();
    return;
}

void SlctMnMenu(void)
{
    HideMouse();
    MinMenuColor(Mm,3,14);
    Wind(SbX[Mm],26,SbX[Mm]+SbWid[Mm]+10,SbNum[Mm]*Hj+44,0,0,1,1);
    if (Key == LEFT) Mm = Mm == 0 ? 3 : Mm-1;
    if (Key == RIGHT) Mm = Mm == 3 ? 0 : Mm+1;
    MinMenuColor(Mm,4,14);
    WrtSbMenu();
    ShowMouse();
}

void SlctSbMenu(void)
{
    HideMouse();
    SbMenuColor(Smm[Mm],3,15);
    if (Key == UPPER) Smm[Mm] = Smm[Mm] == 0 ? SbNum[Mm]-1 :
        Smm[Mm]-1;
    if (Key == DOWN) Smm[Mm] = Smm[Mm] == SbNum[Mm]-1 ? 0 :
        Smm[Mm]+1;
    SbMenuColor(Smm[Mm],14,4);
    ShowMouse();
}

int GetKey(void)          /* 读取键值 */
{
    int Ch,Low,Hig;
    Ch = bioskey(0);
    Low = Ch & 0x00ff;
    Hig = (Ch & 0xff00) >> 8;
    return (Low == 0 ? Hig+256 : Low);
}

void SbFunGo(void)

```

```

{
    switch(Mm)
    {
        case 0:
            switch(Smm[0])
            {
                /* 以下写入个功能对应的函数 */
                case 0: break;
                case 1: break;
                case 2: break;
                case 3: break;
                case 4: break;
            } break;
        case 1:
            switch(Smm[1])
            {
                case 0: break;
                case 1: break;
                case 2: break;
                case 3: break;
                case 4: break;
            } break;
        case 2:
            switch(Smm[2])
            {
                case 0: break;
                case 1: break;
                case 2: break;
            } break;
        case 3:
            switch(Smm[3])
            {
                case 0: break;
                case 1: break;
            } break;
    }
}

void Quit(void)
{
    closegraph();
}

```

```

    fcloseall();
    exit(0);
}

void Screen(void)
{
    char * Chs = "选择", * Ret = "返回", * Qut = "退出";
    char * Or1 = "或鼠标左键", * Or2 = "或鼠标右键";
    char * Arr = "↑ ↓ ← →";
    setfillstyle(1,1);
    bar3d(0,0,639,479,0,0);
    setfillstyle(1,3);
    bar3d(0,0,639,25,0,0);
    bar3d(0,430,639,450,0,0);
    setfillstyle(1,6);
    bar3d(0,450,639,479,0,0);
    setcolor(RED);
    outtextxy(125,436,"Enter");
    outtextxy(340,436,"ESC");
    outtextxy(430,436,"Alt _ X");
    PutCC16(40, 432,4,4, Arr);
    PutCC16(170,432,2,4, Or1);
    PutCC16(270,432,2,14,Chs);
    PutCC16(370,432,2,14,Ret);
    PutCC16(480,432,2,4, Or2);
    PutCC16(580,432,2,14,Qut);
}

void Wind(int x1,int y1,int x2,int y2,int BrdTp,int IsBk,int BkColor,\    int TxtColor)
{
    int i;
    setfillstyle(1,BkColor);
    setcolor(BkColor);
    bar3d(x1,y1,x2,y2,0,1);
    setcolor(TxtColor);
    if (BrdTp) rectangle(x1+2,y1+2,x2-2,y2-2);
    if (IsBk)
    {
        setfillstyle(1,8);setcolor(8);
        bar3d(x1+7,y2+1,x2+7,y2+8,0,1);
    }
}

```

```

        bar3d(x2+1,y1+7,x2+8,y2+8,0,1);
    }
}

void InitGraph(void)
{
    int GraphDrive = DETECT,GraphMode;
    registerbgidriver(EGAVGA __driver);
    initgraph(&GraphDrive,&GraphMode,"");
}

void PutCC16(int x,int y,int Wid,int ForColor,char * Str)
{
    unsigned Zcode,Bcode;          /* 区码,位码 */
    int i,j,k,Rec;
    long Len;
    char Buf[32];
    while (* Str)                  /* 直到字串显示完 */
    {
        if ((* Str & 0x80) && (*(Str+1) & 0x80))    /* 是汉字 */
        {
            Zcode = (* Str-0xa1) & 0x07f;          /* 区码 */
            Bcode = (*(Str+1)-0xa1) & 0x07f;        /* 位码 */
            Rec = Zcode * 94 + Bcode;                /* 记录号 */
            Len = Rec * 32L;                          /* 在字库中位置 */
            fseek(fp16,Len,SEEK_SET);
            fread (Buf,1,32,fp16);                  /* 32 字节 */
            for (i = 0;i < 16;i++)                  /* 垂直方向 16 点 */
                for (j = 0;j < 2;j++)                /* 水平方向 2 字节 */
                    for (k = 0;k < 8;k++)            /* 每字节 8 位 */
                        if (Buf[i * 2 + j] >> (7-k) & 1) /* 是 1 画点 */
                            putpixel(x+j * 8+k,y+i,ForColor);
            x = x+16+Wid;
            Str += 2;
        }
    }
    return;
}

void OpenLIB(void)
{

```

```

    fp16 = fopen("c:\\ucdos\\hzk16", "rb");
}
int InitMouse()
{
    union REGS Inr, Outr;
    Inr.x.ax = 0;
    int86(0x33, &Inr, &Outr);
    return (Outr.x.ax);
}
void ShowMouse()
{
    union REGS Inr;
    Inr.x.ax = 1;
    int86(0x33, &Inr, &Inr);
}
void HideMouse()
{
    union REGS Inr;
    Inr.x.ax = 2;
    int86(0x33, &Inr, &Inr);
}
void ReadMouse(int *f, int *x, int *y)
{
    union REGS Inr, Outr;
    Inr.x.ax = 3;
    int86(0x33, &Inr, &Outr);
    *f = Outr.x.bx;
    *x = Outr.x.cx;
    *y = Outr.x.dx;
}
/* MsInBox():
   判断一个点(x,y) 是否在一矩形框内
   入口参数: x1, y1: 矩形框左上角坐标
             x2, y2: 矩形框右下角坐标
   出口参数: 真: 1
             假: 0 */
int MsInBox(int x1, int y1, int x2, int y2, int x, int y)
{
    return((x >= x1 && x <= x2 && y >= y1 && y <= y2) ? 1 : 0);
}

```

上例运行后的屏幕形式如图 6-6 所示。

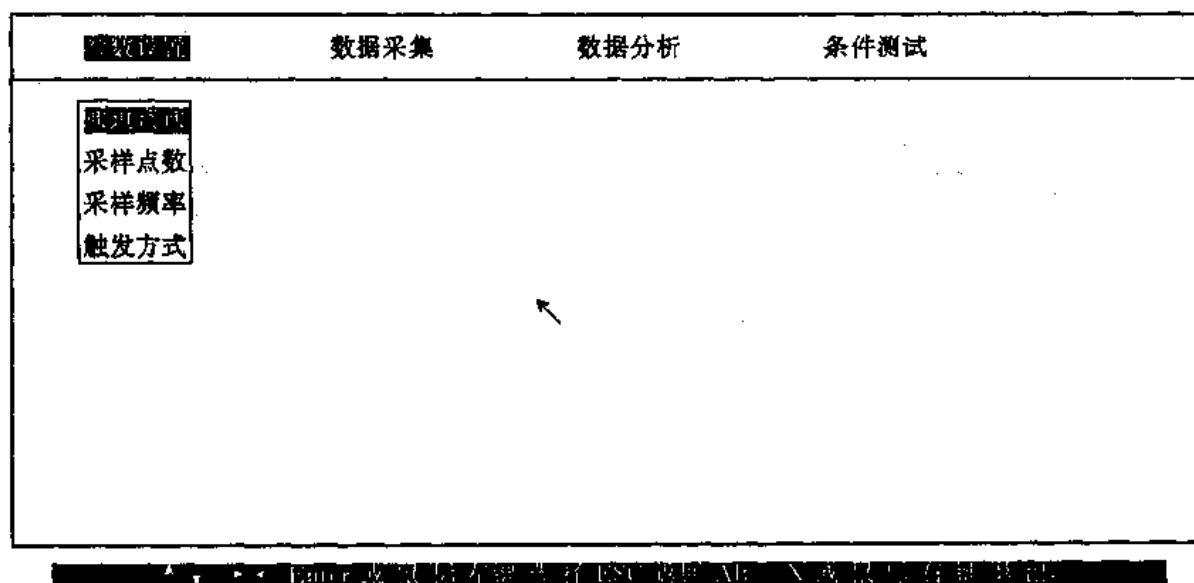


图 6-6

如果没有安装鼠标或没有运行驱动程序,则其只能用键盘上的键进行选择菜单,这时其功能跟 MENU.C 完全相同。当安装有鼠标时可用箭头键选择菜单,也可用鼠标进行选择。用鼠标选择时,如果鼠标器箭头在“数据分析”项上按了鼠标左键,则屏幕内容变为如图 6-7 所示。

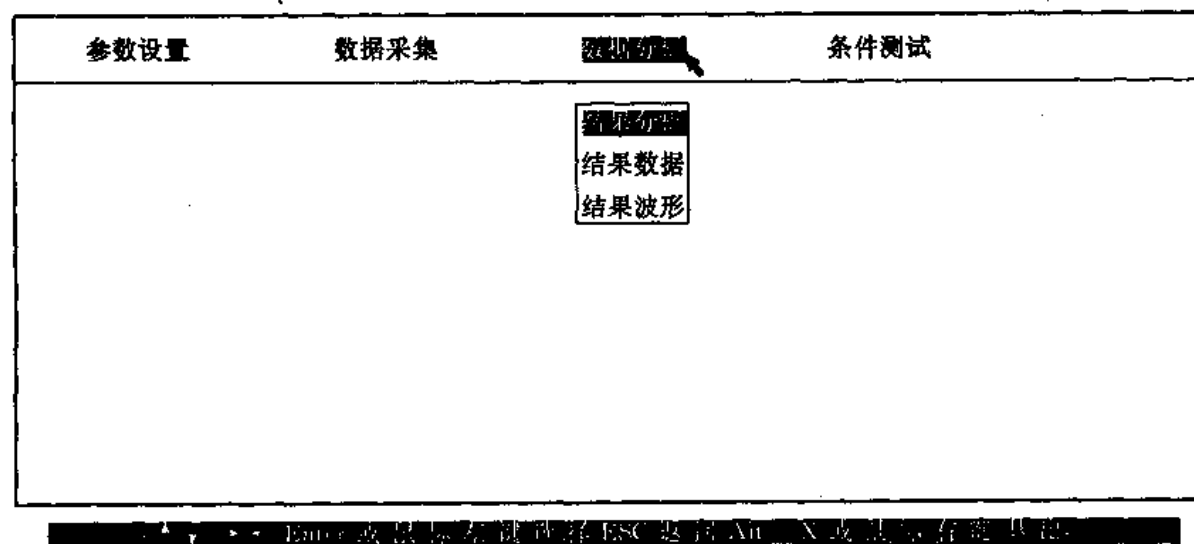


图 6-7

而不是像按键盘上的键时一项一项地移动,从而使效率提高,操作更方便。

## 6.5 交替运行其它语言程序时的菜单设计

用 C 语言编写菜单界面漂亮美观,但有时在某个应用软件中可能用其它语言编写了子程序,比如:用 FoxBASE+ 编制了命令文件进行数据库管理,用 FORTRAN 语言编写了分析计算程序等。虽然在 C 语言中可以通过 system() 函数和 spawnl() 函数调用外部的可执行文件,但当 C 程序较大时,再调用这些程序,可能由于内存不够而无法实现,这些情况也是经常存在的。

在此结合菜单技术,介绍一种较实用的巧用 DOS 批处理命令解决这一问题的方法。

假定用户编制了一个数据库管理系统,命令文件名为 MIS. PRG,用 FORTRAN 语言编制了分析计算程序,名为 CAL. EXE,用 C 语言编写了读取数据库中的数据并画直方图的程序 DRW. EXE。现可以编写一个主菜单 MENU. C,分别调用这个系统。MENU. C 是一个汉字窗口式菜单,为了减少占用内存空间,我们采用西文 DOS 下显示汉字的技术,光条的产生以及按键的读取同 6.1.2 介绍的方法。菜单程序如下:

/\*

MENU. C——交替运行其它语言程序的汉字菜单

\*/

#include "graphics. h"

#include "string. h"

#include "bios. h"

#include "conio. h"

#include "process. h"

#include "stdio. h"

#include "stdlib. h"

#define UPPER 328

#define DOWN 336

#define ESC 27

#define ENTER 13

void WrtMenu(void);

void MenuColor(int n,int BCl,int TCl);

void SlctMenu(void);

int GetKey(void);

void FunGo(void);

void Quit(void);

void Screen(void);

void Wind(int,int,int,int,int,int,int,int,int);

void InitGra(void);

void PutCC16(int x,int y,int Wid,int TCl,char \* Str);



```

void OpenLIB(void);
FILE *fp16, *fp;
int Hj = 40;           /* 定义汉字显示的行距 */
int MuX = 110;         /* 光条的 X 位置 */
int Mm = 0;           /* 菜单选项 */
int Wid = 180;         /* 光条的长度 */
int Key = 0;          /* 选择键值 */
char *Tit="系统主菜单";
char *Mu[4] = {"数据库管理",
               "分析及计算",
               "绘制直方图",
               "退出系统"};

int main(void)
{
    OpenLIB();
    InitGra();
    Screen();
    WrtMenu();
    SlctMenu();
    Quit();
    return(0);
}

void WrtMenu(void)      /* 写主菜单 */
{
    int i;
    for(i = 0; i < 4; i++) MenuColor(i,1,14);
    MenuColor(Mm,4,14); /* 产生光条 */
}

/*  n:菜单的第几项;
   BCl:背景颜色;
   TCl:前景颜色.
*/
void MenuColor(int n,int BCl,int TCl)
{
    setfillstyle(1,BCl);
    bar(MuX,n * Hj+180,MuX+Wid,n * Hj+200);
    PutCC16(MuX+50,n * Hj+182,2,TCl,Mu[n]);
}

void SlctMenu(void)

```

```

{
while(Key != ESC)
{
    Key = GetKey();
    if (Key == UPPER || Key == DOWN)
    {
        MenuColor(Mm,1,14);
        if (Key == UPPER) Mm = Mm == 0 ? 3 : Mm-1;
        if (Key == DOWN) Mm = Mm == 3 ? 0 : Mm+1;
        MenuColor(Mm,4,14);
    }
    if (Key == ENTER) FunGo();
}
}

int GetKey(void)          /* 读取键值 */
{
    int Ch,Low,Hig;
    Ch = bioskey(0);
    Low = Ch & 0x00ff;
    Hig = (Ch & 0xff00) >> 8;
    return (Low == 0 ? Hig+256 : Low);
}

void FunGo(void)
{
    switch(Mm)
    {
        case 0:
            fputs("FOXBASE MIS. PRG\n",fp);
            fputs("MENU. EXE\n",fp);
            break;
        case 1:
            fputs("CAL. EXE\n",fp);
            fputs("MENU. EXE\n",fp);
            break;
        case 2:
            fputs("DRW. EXE\n",fp);
            fputs("MENU. EXE\n",fp);
            break;
    }
}

```

```

        case 3;
            fputs("ECHO ON\n",fp);
        }
        Quit();
    }
void Quit(void)
{
    closegraph();
    fcloseall();
    exit(0);
}
void Screen(void)
{
    char *Chs = "选择", *Qut = "退出";
    char *Arr = "↑↓";
    Wind(100,100,300,360,2,1,1,15,8);
    setcolor(15);
    line(104,150,296,150);
    line(104,152,296,152);
    PutCC16(160,120,2,14,Tit);
    setcolor(RED);
    setfillstyle(1,7);
    bar(0,430,500,450);
    outtextxy(120,436,"Enter:");
    outtextxy(290,436,"ESC:");
    PutCC16(60,432,4,4,Arr);
    PutCC16(190,432,4,14,Chs);
    PutCC16(350,432,4,14,Qut);
}
/* x1,y1,x2,y2 分别为窗口左上角,右下角坐标
   FrmTp:边框类型:0—不带边框,1—单线边框,2—双线边框;
   IsBk:是否带阴影背景,0—不带,非0—带阴影背景;
   BCl:窗口背景颜色;
   TCl:窗口文字颜色;
   BgCl:背景阴影的颜色.
*/
void Wind(int x1,int y1,int x2,int y2,int FrmTp,int IsBk,int BCl,int TCl,\int BgCl)
{
    int i;

```

```

setfillstyle(1,BCl);
bar(x1,y1,x2,y2);
setcolor(TCl);
if (FrmTp)          /* 边框线 */
{
    rectangle(x1+2,y1+2,x2-2,y2-2);
    if (FrmTp == 2) rectangle(x1+4,y1+4,x2-4,y2-4);
}
if (IsBk)           /* 阴影背景 */
{
    setfillstyle(1,BgCl);
    bar(x1+7,y2+1,x2+7,y2+8);
    bar(x2+1,y1+7,x2+8,y2+8);
}
}
void InitGra(void)
{
    int GraphDrive = DETECT,GraphMode;
    registerbgidriver(EGAVGA _driver);
    initgraph(&GraphDrive,&GraphMode,"");
}
void PutCC16(int x,int y,int Wid,int TCl,char *Str)
{
    unsigned Zcode,Bcode;          /* 区码,位码 */
    int i,j,k,Rec;
    long Len;
    char Buf[32];
    while (*Str)                   /* 直到字符串显示完 */
    {
        if ((*Str & 0x80) && (*(Str+1) & 0x80)) /* 是汉字 */
        {
            Zcode = (*Str-0xa1) & 0x07f;      /* 区码 */
            Bcode = (*(Str+1)-0xa1) & 0x07f;    /* 位码 */
            Rec = Zcode * 94 + Bcode;           /* 记录号 */
            Len = Rec * 32L;                    /* 在字库中位置 */
            fseek(fp16,Len,SEEK _SET);
            fread (Buf,1,32,fp16);              /* 32 字节 */
            for (i = 0; i < 16; i++)            /* 垂直方向 16 点 */
                for (j = 0; j < 2; j++)         /* 水平方向 2 字节 */

```

```

        for (k = 0; k < 8; k++)
            if (Buf[i * 2 + j] >> (7 - k) & 1)
                putpixel(x + j * 8 + k, y + i, TCL);
        x = x + 16 + Wid;
        Str += 2;
    }
}
return;
}
void OpenLIB(void)
{
    fp16 = fopen("c:\\ucdos\\hzk16", "rb");
    fp = fopen("CHS.BAT", "w");
}

```

/\* 每字节 8 位 \*/  
/\* 是 1 画点 \*/

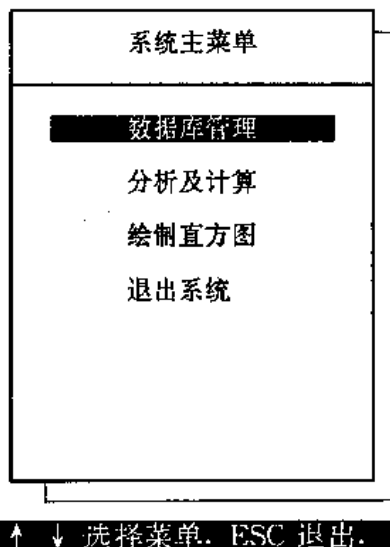


图 6-8

程序运行时, 屏幕上的显示如图 6-8 所示。

当选择某一功能时, 由程序自动向批处理文件 CHS.BAT 中写入批命令, MENU.C 编译后文件名为 MENU.EXE, 建立一个 MAIN.BAT 的批处理文件, 内容如下:

```

ECHO OFF
MENU.EXE
CHS.BAT

```

要运行整个系统时, 只要键入 MAIN, 将首先调用菜单程序 MENU, 由用户选择功能, 当选择数据库管理时, 写入 CHS.BAT 文件中的内容为:

```

FOXBASE MIS.PRG
MENU.EXE

```

接着退出菜单程序并按 CHS.BAT 文件的内容去执行 FoxBASE+ 命令文件 MIS.PRG, 运行完毕又调用 MENU.EXE。

选择“分析及计算”功能时, CHS.BAT 的内容如下:

```

CAL.EXE
MENU.EXE

```

选择“绘制直方图”功能时, CHS.BAT 的内容如下:

```

DRW.EXE
MENU.EXE

```

当选择退出系统功能时, 则 CHS.BAT 文件的内容为 ECHO ON, 从而退出整个程序。

通过以上方法, 彻底消除了运行大程序时内存中有 C 程序而使内存不够的问题, 在操作者来看, 整个系统似乎是用一种语言编写的, 增强了整体效果。但有一点应注意: 在设计的 FoxBASE+ 命令文件结束时, 应使用 QUIT 命令退出 (如果没有编译成可执行文件运行时)。

# 数据输入/输出程序设计

第

7

章

本章内容:

## 7.1 标准输入/输出函数

### 7.1.1 格式化输入输出函数

### 7.1.2 非格式化输入输出函数

## 7.2 文件的输入/输出函数

### 7.2.1 标准文件函数

### 7.2.2 非标准文件函数

## 7.3 可编辑的数据输入程序设计

## 7.4 图形方式下的数据输入

## 7.5 图形方式下的文本输出

### 7.5.1 文本输出函数

### 7.5.2 有关文本字体、字型和输出方式的设置

### 7.5.3 用户对文本字符大小的设置

## 7.6 格式化数据输出

### 7.6.1 格式化文本输出

### 7.6.2 文本覆盖

### 7.6.3 突出显示文本

## 7.7 西文字符的放大、旋转与倾斜显示

### 7.7.1 矢量字库\*.CHR 的结构分析

### 7.7.2 放大、旋转与倾斜

在人机界面设计中,经常要输入一些数据,本章主要介绍 Turbo C/Borland C 输入函数,文件的输入输出函数,可编辑的数据输入程序设计,图形光标的建立及数据输入,格式化数据输出及文字的任意放大、旋转倾斜显示。

## 7.1 标准输入/输出函数

### 7.1.1 格式化输入输出函数

Turbo C/Borland C 标准库提供了两个控制台格式化输入输出函数 `printf()` 和 `scanf()`, 这两个函数可以在标准输入输出设备上以各种不同的格式读写数据。`printf()` 函数用来向标准输出设备(屏幕)写数据; `scanf()` 函数用来从标准输入设备(键盘)上读入数据。

#### 1. `printf()` 函数

`printf()` 函数是格式化输出函数,一般用于向标准输出设备按规定格式输出信息。在编写程序时经常会用到此函数,其调用格式如下:

`printf("<格式化字符串>", <参量表>);`

其中,格式化字符串包括两部分内容:一部分是正常的字符,这些字符将按原样输出;另一部分是格式化规定字符,以“%”开始,后跟一个或几个规定字符,用来确定输出内容格式。

参量表是需要输出的一系列参数表,其个数必须与格式化字符串所说明的输出个数一样多,各参量之间用“,”分开,且顺序一一对应,否则将会出现意想不到的错误。

#### (1) 格式化规定符

Turbo C/Borland C 提供的格式化规定符如表 7-1 所示:

表 7-1 格式化规定符

符号	作用
%d	十进制有符号整数
%u	十进制无符号整数
%f	浮点数
%s	字符串
%c	单个字符
%p	指针的值
%e	指数形式的浮点数
%x 或 %X	无符号以 16 进制表示的整数
%o	无符号的以 8 进制表示的整数

说明:可以在“%”和字母之间插进数字表示最大场宽。

例如：%3d 表示输出 3 位整型数,不够 3 位时,右对齐。

%9.2f 表示输出场宽为 9 的浮点数,其中小数 2 位,整数 6 位,小数点占 1 位。

%8s 表示输出 8 个字符的字符串,不够 8 个字符右对齐。

如果字符串的长度或整型数的位数超过说明的场宽,将按其实际长度输出,但对于浮点数,若整数部分位数超过了说明的整数位宽度,将按实际整数位输出;若小数部分位数超过了说明的小数位宽度,则按说明的宽度以四舍五入输出。

另外,若想在输出值前加一些 0,就应在场宽项前加个 0;可以在“%”和字母之间加一小写字母 l,表示输出的是长整型数;可以控制输出左对齐或右对齐,即在“%”和字母之间加入一个“-”号可以说明输出为左对齐,否则为右对齐。

## (2) 一些特殊规定字符

表 7-2 特殊规定符

符号	作 用
\n	换行
\f	清屏并换页
\r	回车
\t	Tab 符
\xhh	表示一个 ASCII 码用 16 进制表示,其中 hh 是 1—2 个 16 进制整数 16 进制数

有关这些格式的用法请看下例:

```
#include "stdio.h"
#include "string.h"
int main()
{
    char c,s[20],*p;
    int a = 1234,*i;
    float f = 3.141592653589;
    double x = 0.12345678987654321;
    p = "Where are you going?";
    strcpy(s,"Hello,Mr.wang");
    *i = 12;
    c = '\x41';
    printf("a=%d\n",a);        /* 输出十进制整数 */
    printf("a=%6d\n",a);       /* 输出 6 位十进制整数 */
    printf("a=%06d\n",a);      /* 输出 6 位十进制整数,不够 6 位前补 0 */
    printf("a=%2d\n",a);       /* 超过 2 位按实际值输出 */
}
```



```

printf(" * i=%4d\n", * i); /* 输出 4 位十进制整数 */
printf(" * i=%-4d\n", * i); /* 输出左对齐 4 位十进制整数 */
printf("i=%p\n", i); /* 输出地址 */
printf("f=%f\n", f); /* 输出浮点数 */
printf("f=%6.4f\n", f); /* 输出 6 位其中小数 4 位的浮点数 */
printf("x=%lf\n", x); /* 输出长浮点数 */
printf("x=%18.16lf\n", x); /* 输出 18 位其中小数 16 位的长浮点数 */
printf("c=%c\n", c); /* 输出字符 */
printf("c=%x\n", c); /* 输出字符的 ASCII 码值 */
printf("s[]=%s\n", s); /* 输出数组字符串 */
printf("s[]=%9s\n", s); /* 输出最多为 9 个字符的字符串 */
printf("s=%p\n", s); /* 输出数组字符串字符地址 */
printf(" * p=%s\n", p); /* 输出指针字符串 */
printf("p=%p\n", p); /* 输出指针的值 */
return (0);
}

```

输出结果:

```

a=1234
a=1234
a=001234
a=1234
 * i=12
 * i=12
i=0716
f=3.141593
f=3.1416
x=0.123457
x=0.1234567898765432
c=A
c=41
s[]=Hello,Mr. wang
s[]=Hello,Mr
s=FFBC
 * p=Where are you going?
p=0194

```

## 2. scanf() 函数

scanf() 函数是格式化输入函数, 它从标准输入设备(键盘)读取输入的信息。其调用格式为:

```
scanf("<格式化字符串>", <地址表>);
```

格式化字符串包括以下三类不同的字符：

- (1) 格式化说明符：格式化说明符与 printf() 函数中的格式说明符基本相同；
- (2) 空白字符：空白字符使 scanf() 函数在读操作中略去输入中的一个或多个空白字符；
- (3) 非空白字符：一个非空白字符会使 scanf() 函数在读入时，剔除掉与这个非空白字符相同的字符。

例如：scanf("%d,%d",&i,&j);

先读入一个整数，然后把接着输入的逗号剔除掉，最后读入另一个整型数。

地址表是需要读入的所有变量的地址，而不是变量本身。这与 printf() 函数完全不同，各变量的地址之间用“,”号分开。

说明：

- (1) 对于字符串数组或字符串指针变量，由于数组名和指针变量名本身就是地址，因此使用 scanf() 函数时，不需要在它们前面加上“&”操作符；

```
#include "stdio.h"
```

```
int main()
```

```
{
```

```
    char *p,str[20];
```

```
    scanf("%s",p);
```

```
    scanf("%s",str);
```

```
    printf("%s\n",p);
```

```
    printf("%s\n",str);
```

```
    return (0);
```

```
}
```

- (2) 可以在格式化字符串中的“%”和格式化规定符之间加入一个整型数，表示任何读操作中的最大位数。

上例中若规定只输入 10 个字符给字符串指针 p，则第一条 scanf() 函数语句变为：

```
scanf("%10s",p);
```

程序运行时，一旦输入字符个数大于 10，p 就不再继续读入了，而后面一个读入函数，即 scanf("%s",str) 就会从第 11 个字符开始读入。

实际使用 scanf() 函数时存在一个问题，下面进行说明。

当使用多个 scanf() 函数连续给多个字符变量输入时，例如：

```
#include "stdio.h"
```

```
int main()
```

```
{
```

```
    char c1,c2;
```

```
    scanf("%c",&c1);
```

```
    scanf("%c",&c2);
```

```
    printf("c1 is %c,c2 is %c",c1,c2);
```

```
    return (0);
```

```
}
```

运行该程序,输入一个字符 A 后回车(要完成输入必须回车),在执行 scanf("%c",&c1) 时,给变量 c1 赋值“A”,但回车符仍然留在缓冲区内,接着执行语句 scanf("%c",&c2)时,变量 c2 将接收一个回车符而使输入结束。输出的结果中变量 c2 没有赋值,c2 的输出是一个空行,如果输入 AB 后回车,则输出结果为: c1 is A,c2 is B。

要解决以上问题,可以在输出函数前加入清除函数 fflush(),修改以上程序变成:

```
#include "stdio.h"
int main()
{
    char c1,c2;
    scanf("%c",&c1);
    fflush(stdin);
    scanf("%c",&c2);
    printf("c1 is %c,c2 is %c",c1,c2);
    return (0);
}
```

### 7.1.2 非格式化输入输出函数

非格式化输入输出函数可以由上面讲述的标准格式化输入输出函数代替,不过非标准函数编译后代码少,相对占用内存也小,从而提高了速度,同时使用也比较方便。

#### 1. puts()和 gets()函数

##### (1) puts()函数

puts()函数用来向标准输出设备(屏幕)写字符串并换行,其调用格式为:

```
puts(s);
```

其中 s 为字符串变量(字符串数组名或字符串指针)。

puts()函数的作用与语句 printf("%s\n",s)相同。

```
#include "stdio.h"
int main()
{
    char s[20], *f;
    strcpy(s,"Hello! Mr. Wang?");
    f = "Fine, Thank you, and you?";
    puts(s);
    puts(f);
    return (0);
}
```

说明:

- puts()只能输出字符串,不能输出数值或进行格式变换;
- 可以将字符串直接写入 puts()函数中,如:

```
puts("Hello!");
```

## (2) gets()函数

gets()函数用来从标准的输入设备(键盘)读取字符串直到回车结束,回车符不属于该字符串,其调用格式为:

```
gets(s);
```

其中s为字符串变量(字符串数组名或字符串指针)。

gets(s)函数与scanf("%s",s)相似,但不完全相同,使用scanf("%s",s)函数输入字符串时,存在一个问题,就是如果输入了空格会被认为输入字符串结束,空格后的字符将作为下一个输入项处理,但gets()函数将接收输入的整个字符串直到回车为止。

```
#include "stdio.h"
```

```
int main()
```

```
{
```

```
    char s[20], *f;
```

```
    printf("What's your name? \n");
```

```
    gets(s);          /* 等待输入字符串直到回车结束 */
```

```
    puts(s);          /* 将输入的字符串输出 */
```

```
    puts("How old are you?");
```

```
    gets(s);
```

```
    puts(s);
```

```
    return (0);
```

```
}
```

说明:

●gets(s)函数中的变量s为一个字符串。如果为单个字符,编译时不会有错误,但运行后会出现“Null pointer assignment”的错误。

## 2. putchar(), getch()和 getchar()函数

(1) putchar()函数是向标准输出设备输出一个字符,其调用格式为:

```
putchar(ch);
```

其中,ch为一个字符变量或常数。

putchar()函数的作用等同于printf("%c",ch)。

```
#include "stdio.h"
```

```
int main()
```

```
{
```

```
    char c;
```

```
    c = 'A';          /* 给字符变量赋初值 */
```

```
    putchar(c);        /* 输出该字符 */
```

```
    putchar('A');      /* 直接输出字符“A” */
```

```
    putchar('\x41');   /* 输出字母“A” */
```

```
    putchar(0x41);     /* 直接用ASCII码值输出字母“A” */
```

```
    return (0);
```

}

## (2) getch()和 getche()函数

getch()和 getche()都是从键盘上读入一个字符,其调用格式为:

```
getch();
```

```
getche();
```

两者的区别是: getch()函数不把读入的字符回显到屏幕上,而 getche()函数却将读入的字符回显到屏幕上。

```
#include "stdio.h"
```

```
int main()
```

```
{
```

```
    char c,ch;
```

```
    c = getch();          /* 从键盘上读入一个字符不回显送给字符变量 c */
```

```
    putchar(c);          /* 输出该字符 */
```

```
    ch = getche();        /* 从键盘上带回显的输入一个字符送给字符变量 ch */
```

```
    putchar(ch);
```

```
    return (0);
```

```
}
```

这两个函数经常用于交互输入的过程中完成暂停等功能。

```
#include "stdio.h"
```

```
int main()
```

```
{
```

```
    char c,s[20];
```

```
    printf("Name: ");
```

```
    gets(s);
```

```
    printf("按任意一个键继续...");
```

```
    getch();
```

```
    return (0);
```

```
}
```

## (3) getchar()函数

getchar()函数也是从键盘上读入一个字符,并带回显。它与前面的两个函数的区别在于: getchar()函数等待输入直到按回车才结束,回车前的所有输入字符都会逐个显示在屏幕上。但只有第一个字符作为函数的返回值。

getchar()函数的调用格式为:

```
getchar();
```

```
#include "stdio.h"
```

```
int main()
```

```
{
```

```
    char c;
```

```
    c = getchar();        /* 从键盘上读入字符直到回车结束 */
```

```

    putchar(c);          /* 显示输入的第一个字符 */
    getch();             /* 等待按任意一键 */
    return (0);
}

```

## 7.2 文件的输入/输出函数

Turbo C/Borland C 提供了两类关于文件的函数：一类称做标准文件函数，也称缓冲型文件函数，这是 ANSI 标准定义的函数；另一类称做非标准文件函数，也称做非缓冲型文件函数。

### 7.2.1 标准文件函数

标准文件函数主要包括文件的打开、关闭、读和写等函数。

#### 1. 文件的打开关闭

任何一个文件在使用之前都必须打开，使用之后关闭，这是因为操作系统对于同时打开的文件数目是有限制的。DOS 操作系统中，可以在 DEVICE.SYS 中定义允许同时打开的文件数  $n$  (用 FILES =  $n$  定义)。其中  $n$  为可同时打开的文件数，一般  $n \leq 20$ 。因此在使用文件前应打开文件，才可对其中的信息进行存取，用完后需要关闭，否则将会出现一些意想不到的错误。Turbo C/Borland C 提供了打开和关闭文件的函数。

##### (1) fopen() 函数

fopen() 函数用于打开文件，其调用格式为：

FILE \* fopen(char \* filename, char \* type);

文件指针 FILE 是一个包括了文件管理有关信息的数据类型。

filename 为文件名，可以包括路径和文件名两部分。如：

"D:\UCDOS\HZK16"

type 表示打开文件的类型。关于文件的规定参见表 7-3。

表 7-3 文件操作类型

字符	含 义
"r"	打开文字文件只读
"w"	创建文字文件只写
"a"	增补，如果文件不存在则创建一个
"r+"	打开一个文字文件读/写
"w+"	创建一个文字文件读/写
"a+"	打开或创建一个文件增补
"b"	二进制文件(可以和上面每一项合用)
"t"	文字文件(默认值)

如果要打开一个 UC DOS 子目录下的文件 HZK16 读数据,可写成:

```
fopen("UCDOS\\HZK16","rb");
```

打开文件成功时,fopen()函数返回文件指针,否则返回空指针(NULL)。

## (2) fclose()函数

fclose()函数用来关闭一个由 fopen()函数打开的文件,其调用格式为:

```
int fclose(FILE * stream);
```

该函数将返回一个整数值,当文件关闭成功时,返回 0,否则返回一个非 0 值。可以根据函数的返回值判断文件是否关闭成功。

```
#include "stdio.h"
```

```
int main()
```

```
{
```

```
FILE * fp; /* 定义文件指针 */
```

```
int i;
```

```
fp = fopen("HZK16","rb"); /* 打开当前目录中名为 HZK16 的文件只读 */
```

```
if (fp == NULL) /* 判断文件打开是否成功 */
```

```
puts("文件打不开!");
```

```
i = fclose(fp); /* 关闭打开的文件 */
```

```
if (i) /* 判断文件关闭是否成功 */
```

```
printf("文件关闭错误!");
```

```
else
```

```
printf("O. K!");
```

```
return (0);
```

```
}
```

## 2. 有关文件操作的函数

本节所述的文件读写函数均是指顺序读写,即读了一条信息后,指针自动加 1。

### (1) 文件的顺序写函数

fprintf(). fputs()和 fputc()函数

函数 fprintf(). fputs()和 fputc()函数均为文件的顺序写操作函数,其调用格式如下:

```
int fprintf(FILE * stream, char * format, <variable list>);
```

```
int fputs(char * string, FILE * stream);
```

```
int fputc(int ch, FILE * stream);
```

上述三函数的返回值均为整型量。fprintf()函数的返回值为实际写入文件中的字符个数(字节数)。如果写错误,则返回一个负数,fputs()函数返回 0 时表明将 string 指针所指的字符串写入文件的操作成功,返回非 0,表明写操作失败。fputc()函数返回一个向文件所写字符的值,此时写操作成功,否则返回 EOF() (在 stdio.h 中定义,值为 -1)表示写操作错误。

fprintf()函数中格式化的规定与 printf()函数相同,所不同的是 fprintf()函数是向文件中写入,而 printf()是向屏幕输出。

下面介绍一个例子,运行后产生一个 TEST.DAT 的文件:

```
#include "stdio.h"
```

```

int main()
{
    FILE *fp;
    int i = 620;
    char *s = "That's good news";
    fp = fopen("TEST.DAT","w");           /* 建立一个文字文件只写 */
    fputs("Your score of TOEFL is:",fp);   /* 向所建文件写入一串字符 */
    fprintf(fp,"%d\n",i);                   /* 向所建文件写入一整型数 */
    fprintf(fp,"%s",s);                     /* 向所建文件写入一字符串 */
    fclose(fp);
    return (0);
}

```

用 DOS 的 TYPE 命令显示 TEST.DAT 的内容如下所示:

Your score of TOEFL is: 620

That's good news

## (2) 文件的顺序读操作函数

fscanf()、fgets()和 fgetc()函数

这三个函数均为文件的顺序读操作函数,其调用格式为:

```

int fscanf(FILE *stream, char *format, <address list>);
char fgets(char *string, int n, FILE *stream);
int fgetc(FILE *stream);

```

fscanf()函数的用法与 scanf()函数相似,只是它是从文件中读取信息。fscanf()函数的返回值为 EOF(-1)时,表示读错误,否则读数据成功。fgets()函数从文件中读取至多 n-1 个字符(n 用来指定字符个数),并把它们放入 string 指向的字符串中,在读入之后自动向字符串末尾加一个空格字符,读成功返回 string 指针,失败返回一个空指针。fgetc()函数返回文件当前位置的一个字符,读错误时返回 EOF。

下面的程序读取上例产生的 TEST.DAT 文件,并将读出结果显示在屏幕上:

```

#include "stdio.h"
#include "conio.h"
int main()
{
    FILE *fp;
    int i;
    char *s,m[20];
    fp = fopen("TEST.DAT","r");           /* 打开文字文件只读 */
    fgets(s,24,fp);                        /* 从文件中读取 24 个字符 */
    printf("%s",s);                        /* 输出所读的字符串 */
    fscanf(fp,"%d",&i);                    /* 读取整型数 */
    printf("%d",i);                        /* 输出所读整型数 */
}

```



```

    putchar(fgetc(fp));          /* 读取一个字符同时输出 */
    fgets(m,17,fp);              /* 读取 17 个字符 */
    puts(m);                     /* 输出所读字符串 */
    fclose(fp);                  /* 关闭文件 */
    getch();
    return (0);
}

```

说明:

- 只要是读文字文件,则不论是字符还是数字,均按其 ASCII 码值处理;
- fscanf()函数读到空白符时,便自动结束,在使用时要特别注意。

### (3) 文件的随机读写

有时用户想直接读取文件中间某处的信息,若用文件的顺序读写必须从文件头开始直到要求的文件位置再读,这显然不方便。Turbo C/Borland C 提供了一组文件的随机读写函数,即将文件的位置指针定位在所要求读写的地方直接读写,文件的随机读写函数如下:

```

int fseek(FILE *stream,long offset,int fromwhere);
int fread(void *buf,int size,int count,FILE *stream);
int fwrite(void *buf,int size,int count,FILE *stream);
long ftell(FILE *stream);

```

fseek()函数的作用是将文件的位置指针设置到从 fromwhere 开始的第 offset 字节的位置上,其中 fromwhere 是表 7-4 中所下列几个宏定义之一:

表 7-4 文件位置指针起始计算位置 fromwhere

符号常数	数 值	含 义
SEEK__SET	0	从文件开头
SEEK__CUR	1	从文件指针的现行位置
SEEK__END	2	从文件末尾

offset 是指文件位置指针从指定开始位置(fromwhere 指出的位置)跳过的字节数。它是一个长整型量,以支持大于 64K 字节的文件。fseek()函数一般用于对二进制文件进行操作。

当 fseek()函数返回 0 时表示操作成功,返回非 0 表示失败。

下面的程序从二进制文件 TESTB.DAT 中读取第 8 个字节:

```

#include "stdio.h"
int main()
{
    FILE *fp;
    if ((fp == fopen("TESTB.DAT","rb")) == NULL)
        /* 打开二进制文件只读,并判断打开是否成功 */

```

```

{
    printf("文件打不开!");
    exit(1);
}
fseek(fp,8,SEEK_SET);          /* 文件指针定位 */
fgetc(fp);                     /* 读取一个字符 */
fclose(fp);                    /* 关闭文件 */
return (0);
}

```

fread()函数是从文件中读 count 个字段,每个字段长度为 size 个字节,并把它们存放到 buf 指针所指的缓冲器中。

fwrite()函数是把 buf 指针所指的缓冲中,长度为 size 个字节的 count 个字段写到 stream 指向的文件中去。

随着读和写字节数的增大,文件位置指示器也增大,读多少个字节,文件指示器相应也跳过多少个字节。读写完毕,函数返回所读和写的字段个数。

ftell()函数返回文件位置指示器的当前值,这个值是指示器从文件头开始计算起的字节数,返回的数为长整型数,当返回-1时,表明出现错误。

下面的程序把一个浮点数组以二进制方式写入文件 TESTB.DAT 中:

```

#include "stdio.h"
int main()
{
    FILE *fp;
    float f[6] = {3.2,-4.34,25.04,0.1,50.5,80.6};
    int i;
    fp = fopen("TESTB.DAT","wb");    /* 创建一个二进制文件只写 */
    fwrite(f,sizeof(float),6,fp);    /* 将6个浮点数写入文件中 */
    fclose(fp);
    return (0);
}

```

下面的例子从一个叫 TESTB.DAT 的文件中读取 100 个整数,并把它们放到 dat 数组中:

```

#include "stdio.h"
int main()
{
    FILE *fp;
    int dat[100];
    int i;
    fp = fopen("TESTB.DAT","rb");    /* 打开一个二进制文件只读 */
    if (fread(dat,sizeof(int),100,fp) != 100) /* 判断是否读了100个数 */

```

```

{
    if (feof(fp))
        printf("文件结束");
    else
        printf("读数据错误!");
}
fclose(fp);
return (0);
}

```

说明:

当用标准文件函数对文件进行读写操作时,首先将所读写的内容放进缓冲区,即写函数只对输出缓冲区进行操作,读函数只对输入缓冲区操作。例如向一个文件写入内容,所写的内容将首先放在输出缓冲区中,直到输出缓冲区存满或使用 `fclose()` 函数关闭文件时,缓冲区的内容才会写入文件中。若无 `fclose()` 函数,则不会向文件中存入所写的内容或所写的内容不全。有一个对缓冲区进行刷新的函数,即 `fflush()`,其调用格式为:

```
int fflush (FILE * stream);
```

该函数将输出缓冲区中的内容实际写入文件中,而将输入缓冲取的内容清除掉。

(4) `feof()` 和 `rewind()` 函数

这两个函数的调用格式为:

```
int feof(FILE * stream);
```

```
int rewind(FILE * stream);
```

`feof()` 函数检测文件位置指示器是否到达了文件结尾,若是则返回一个非 0 值,否则返回 0。这个函数对二进制文件操作特别有用,因为二进制文件中,文件结束标志 EOF 也是一个合法的二进制数,只简单地检查读入字符的值来进行判断文件是否结束是不行的。如果那样的话,可能会造成文件未结尾而被认为结尾,所以就必须用 `feof()` 函数。

`rewind()` 函数用于把文件位置指示器移到起点处,成功时,返回 0,否则返回非 0 值。

## 7.2.2 非标准文件函数

这类函数最早用于 UNIX 操作系统,ANSI 标准未定义,但有时也经常用到,DOS3.0 以上版本支持这些函数,它们的头文件为 `io.h`。

### 1. 文件的打开和关闭

#### (1) `open()` 函数

`open()` 函数的作用是打开文件,其调用格式为:

```
int open(char * filename, int access);
```

该函数表示按 `access` 的要求打开文件名为 `filename` 的文件,返回值为文件描述字,其中 `access` 有两部分内容:基本模式和修饰符,两者用“ ”或“|”方式连接。修饰符可以有多个,但基本模式只能有一个。`access` 的规定如下表 7-5 所示。

表 7-5

access 的规定

基本模式	含义	修饰符	含 义
O__RDONLY	只读	O__APPEND	文件指针指向末尾
O__WRONLY	只写	O__CREAT	文件不存在时,创建文件, 属性按基本模式属性
O__RDWR	读写	O__TRUNC	若文件存在,将其长度 缩为 0,属性不变
		O__BINARY	打开一个 二进制文件
		O__TEXT	打开一个 文本文件

open()函数打开成功,返回值就是文件描述的值(非负数),否则返回-1。

## (2) close()函数

close()函数的作用是关闭由 open()函数打开的文件,其调用格式为:

```
int close(int handle);
```

该函数关闭与文件描述字 handle 相连的文件。

## 2. 读写函数

### (1) read()函数

read()函数的调用格式为:

```
int read(int handle,void *buf,int count);
```

read()函数从 handle(文件描述字)相连的文件中,读取 count 个字节放到 buf 所指的缓冲区中,返回值为实际所读字节数,返回-1 表示出错,返回 0 表示文件结束。

### (2) write()函数

write()函数的调用格式为:

```
int write(int handle,void *buf,int count);
```

该函数把 count 个字节从 buf 指向的缓冲区写入与 handle 相连的文件中,返回值为实际写入的字节数。

## 3. 随机定位函数

### (1) lseek()函数

lseek()函数的调用格式为:

```
int lseek(int handle,long offset,int fromwhere);
```

该函数对与 handle 相连的文件位置指针进行定位,功能和用法与 fseek()函数相同。

### (2) tell()函数

tell()函数的调用格式为:

```
long tell(int handle);
```

该函数返回与 handle 相连的文件现行位置指针,功能和用法与 ftell()函数相同。

### 7.3 可编辑的数据输入程序设计

C 语言提供的这些标准输入函数并不很完善,表现在无编辑能力,控制能力弱,界面差。为此本节将提供中西文字符方式以及西文图形方式下的数据输入方法。

针对 C 语言标准输入函数的不足,我们编制了一个函数 GetD(),能按指定的属性和字段长度进行输入。具有仿全屏幕编辑的功能,能够越界报警,能够在汉字系统下随意输入。删除、插入汉字,并具有消除半个汉字的功能,键入回车键认可输入,ESC 键保留默认的字串,函数的返回值为中断编辑的键值。

/\*

GETD.C—— 中西文 DOS 下可编辑的数据输入程序

\*/

#include "conio.h"

#include "bios.h"

#include "mem.h"

#include "alloc.h"

#include "string.h"

#include "stdio.h"

#include "stdlib.h"

#include "graphics.h"

#define LEFT 331 /\* 功能键的宏定义 \*/

#define RIGHT 333

#define HOME 327

#define INS 338

#define END 335

#define DEL 339

#define ESC 27

#define BACK 8

#define ENTER 13

#define TRUE 1

#define FALSE 0

void GetD(int,int,int,int,int,unsigned char \*OldStr);

int GetKey(void);

int main(void)

{

unsigned char Str[30]; /\* 定义字符串 \*/

int i;

directvideo = 0;

```

clrscr();
strcpy(Str,"计算机"); /* 赋初值 */
GetD(10,10,WHITE,BLUE,30,Str);
textbackground(0);
clrscr();
printf("%s\n",Str); /* 显示输入的数据 */
getch();
return 0;
}
/*
TCl: 文字颜色;
BCl: 背景色;
Len: 字符串长度;
OldStr: 初始字符串。
*/
void GetD(int x,int y,int TCl,int BCl,int Len,unsigned char * OldStr)
{
    unsigned char * NewStr;
    unsigned char cc,uu;
    int OldLen,Loop,Key,Pos,Attr,Chis,Ins,i,j,k;
    Loop = TRUE; Chis = FALSE; Ins = TRUE;
    NewStr = (unsigned char *) malloc(Len+1);
    OldLen = strlen(OldStr);
    memcpy(NewStr,OldStr,OldLen); /* 复制缺省串入编辑串 */
    memset(NewStr+OldLen,' ',Len-OldLen);
    *(NewStr+Len) = '\0';
    Attr = (BCl << 4) + TCl;
    textattr(Attr);
    gotoxy(x,y); cprintf("%s",NewStr);
    if (OldLen) Pos = OldLen; /* 确定编辑光标位置 */
    else Pos = 0;
    gotoxy(x+Pos,y);
    do
    {
        Key = GetKey(); /* 读取按键 */
        switch(Key)
        {
            case HOME;
                Pos = 0; gotoxy(x,y); break; /* 光标移至首位 */

```

```

case END:
    Pos = Len-1; gotoxy(x+Len,y); break;          /* 光标移至尾位 */
case LEFT:
    if (Pos <= 0) cprintf("\007");                /* 越界报警 */
    else
    {
        Pos--;
        if (Pos && * (NewStr+Pos) > 0xa0) Pos--;    /* 汉字左移两位 */
        gotoxy(x+Pos,y);
    }
    break;
case RIGHT:
    if (Pos >= Len-1) cprintf("\007");
    else
    {
        Pos++;
        if (Pos != Len-1 && * (NewStr+Pos) > 0xa0) Pos++;
        gotoxy(x+Pos,y);                          /* 汉字右移两位 */
    }
    break;
case BACK:
    if (Pos <= 0) cprintf("\007");                /* 越界报警 */
    else
    {
        i = 1;
        if ((Pos-i) && * (NewStr+Pos-i) > 0xa0) i++; /* 汉字退两格 */
        memmove(NewStr+Pos-i, NewStr+Pos, Len-Pos);
        memset(NewStr+Len-i, ' ', i);
        Pos -= i;
        gotoxy(x,y); cprintf("%s", NewStr);
        gotoxy(x+Pos,y);
    }
    break;
case DEL:
    if (Pos > Len-1) cprintf("\007");             /* 越界报警 */
    else
    {
        if (Pos == Len-1) * (NewStr+Len-1) = ' ';
        else

```

```

    {
        i = 1;
        if ((Pos-i) && * (NewStr+Pos) > 0xa0) i++; /* 汉字删两位 */
        memmove(NewStr+Pos, NewStr+Pos+i, Len-Pos);
        memset(NewStr+Len-i, ' ', i);
    }
    gotoxy(x, y);
    cprintf("%s", NewStr);
    gotoxy(x+Pos, y);
}
break;
case INS:
    Ins = ! Ins; /* 插入键切换 */
    break;
case ESC:
case ENTER: /* 回车键退出循环 */
    Loop = FALSE; break;
default:
    if (Key > 255) continue;
    i = 1;
    uu = (unsigned char) Key;
    if (Key > 0xa0) /* 如果是汉字 */
    {
        cc = (unsigned char) GetKey(); /* 读取另一内码 */
        i++;
        Chis = TRUE;
        if (! Ins && * (NewStr+Pos) < 0xa0 && * (NewStr+Pos+1) > 0xa0)
            memmove(NewStr+Pos+2, NewStr+Pos+1, 1);
    }
    else if (! Ins && * (NewStr+Pos) > 0xa0) {
        memcpy(NewStr+Pos+1, NewStr+Pos+2, Len-Pos);
        * (NewStr+Len-1) = '\0';
    }
    if (Ins) memmove(NewStr+Pos+i, NewStr+Pos, Len-Pos);
    * (NewStr+Len) = '\0';
    * (NewStr+Pos) = uu;
    if (i > 1) * (NewStr+Pos+1) = cc;
    j = 0;
    for (k = Pos; k < Len; k++)

```



```

        if ((*(NewStr+k)) > 0xa0) j++;
        if (j % 2) *(NewStr+Len-1) = ' '; /* 删除半个汉字 */
        gotoxy(x,y); cprintf("%s",NewStr);
        if (Chis)
        {
            gotoxy(x+Pos+2,y);
            Chis = FALSE;
        }
        else gotoxy(x+Pos+1,y);
        Pos += i;
        if (Pos >= Len)
        {
            cprintf("\007"); Loop = FALSE;
        }
        break;
    }
} while (Loop);
if (Key == ENTER) strcpy(OldStr,NewStr);
free(NewStr);
return;
}

int GetKey(void)
{
    int Ch,Low,Hig;
    Ch = bioskey(0);
    Low = Ch & 0x00ff;
    Hig = (Ch & 0xff00) >> 8;
    return (Low == 0 ? Hig+256 : Low);
}

```

## 7.4 图形方式下的数据输入

在图形方式下,数据的输入可用前面介绍的标准函数,但没有光标,使用非常不便。因此,解决的关键是如何建立图形光标。建立光标的方法有多种,本章介绍的是采用异或(XOR)方式实现画线的。首先利用三点宽直线画两条高为9个象素单位的线段,形成一个6×9的矩形块光标,当有合法的键按下时,先在原来光标位置重画一次光标。然后在下一个字符位置再画出光标,由于采用了异或(XOR)方式,因此原先的光标消失,新位置光标出现,就相当于光标移动了。源程序如下:

```
/*  
    GGetD.c : 图形方式下的数据输入——图形光标的建立
```

```
*/
```

```
#include "stdio.h"
```

```
#include "conio.h"
```

```
#include "stdlib.h"
```

```
#include "graphics.h"
```

```
#define ESC      27
```

```
#define ENTER    13
```

```
#define BACKSPACE 8
```

```
void InitGra(void);
```

```
void Quit(void);
```

```
void GGetD(int x,int y,char * Str);
```

```
void Curs(int x,int y);
```

```
int main(void)
```

```
{
```

```
    char Str[30];
```

```
    InitGra();
```

```
    GGetD(100,100,Str);
```

```
    printf("%s\n",Str);
```

```
    getch();
```

```
    Quit();
```

```
    return(0);
```

```
}
```

```
void InitGra(void)
```

```
{
```

```
    int GraphDrive = DETECT,GraphMode;
```

```
    registerbgidriver(EGAVGA-driver);
```

```
    initgraph(&GraphDrive,&GraphMode,"");
```

```
}
```

```
void Quit(void)
```

```
{
```

```
    closegraph();
```

```
    exit(0);
```

```
}
```

```
/*
```

```
入口参数:x,y: 提示信息的坐标;
```

```
        Str: 定义的串指针;
```

```
出口参数:Str: 输入的串;
```

```

* /
void GGetD(int x,int y,char * Str)
{
    int i = 0,j,k,Len;
    int Wid,ii;
    char ch = 0,s[2];
    s[1] = '\0';
    Len = 1; Wid = 160;
    setfillstyle(1,1);
    setcolor(15);
    bar(Len * 8+x,y,x+Wid+Len * 8,y+16);
    setwritemode(XOR __PUT);
    for (; )
    {
        j = (Len+i) * 8;
        Curs(x+j,y+6);
        ch = getch();
        Curs(x+j,y+6);
        if (ch == ESC)                /* ESC 键取消 */
        {
            i = 0;
            break;
        }
        if (ch == ENTER) break;      /* ENTER 键,退出循环 */
        if (ch == BACKSPACE)         /* BACKSPACE 键,光标左移 */
        {
            if (--i < 0) i = 0;
            k = (Len+i) * 8;
            bar(x+k,y,x+k+8,y+16);
        }
        else
        {
            Str[i] = s[0] = ch;
            outtextxy(x+j,y+6,s);
            i++;
        }
    }
    Str[i] = '\0';
    setwritemode(COPY __PUT);
}

```

```

    return;
}
/*
    用于显示一矩形光标
*/
void Curs(int x,int y)
{
    int OldColor;
    OldColor = getcolor();
    setlinestyle(0,0,3);
    setcolor(15);
    line(x+2,y-2,x+2,y+7);
    line(x+5,y-2,x+5,y+7);
    setcolor(OldColor);
    setlinestyle(0,0,1);
}

```

## 7.5 图形方式下的文本输出

在图形模式下,只能用标准输出函数,如 `printf()`,`puts()`,`putchar()` 函数输出文本到屏幕。除此之外,其它输出函数(如窗口输出函数)不能使用,即使是可以输出的标准函数,也只能以前景为白色,按 80 列,25 行的文本方式输出。

Turbo C/Borland C 也提供了一些专门用于在图形显示模式下的文本输出函数。下面予以介绍:

### 7.5.1 文本输出函数

```
void far outtext(char far *txtstring);
```

该函数将输出字符串指针 `txtstring` 所指的文本在现行位置。

```
void far outtextxy(int x,int y,char far *txtstring);
```

该函数输出字符串指针 `txtstring` 所指的文本在规定的  $(x,y)$  位置。其中,  $x$  和  $y$  为象元坐标。

说明:

这两个函数都是输出字符串,但经常会遇到输出数值或其它类型的数据,此时就必须使用格式化输出函数 `sprintf()`:

```
int sprintf(char *str,char *format,varlist);
```

它与 `printf()` 函数的不同之处是将按格式化规定的内容写入 `str` 指向的字符串中,返回值等于写入的字符个数。

### 7.5.2 有关文本字体、字型和输出方式的设置

有关图形方式下的文本输出函数,可以通过 `setcolor()` 函数设置输出的文本颜色。另外,也可以改变文本字体大小以及选择水平方向输出还是垂直方向输出。

```
void far settextjustify(int horiz,int vert);
```

该函数用于定位输出字符串。

对使用 `outtextxy()` 函数所输出的字符串,其中哪个点对应于定位坐标  $(x,y)$  在 TurboC/Borland C 中是有规定的。如果把一个字符串看成一个长方形的图形,在水平方向显示时,字符串长方形按垂直方向可分为顶部、中部和低部三个部分,水平方向可分为左中右三个位置,两者结合就有 9 个位置:

`settextjustify()` 中的第一个参数 `horiz` 指出水平方向三个位置中的一个,第二个参数 `vert` 指出垂直方向三个位置中的一个,二者就确定了其中的一个位置。当规定了这个位置后,用 `outtextxy()` 函数输出字符串时,字符串长方形的这个规定位置就对准函数中的  $(x,y)$  位置。而对用 `outtext()` 函数输出字符串时,这个规定的位置就位于现行游标的位置。有关参数 `horiz` 和 `vert` 的取值参见表 7-6。

表 7-6 参数 `horiz` 和 `vert` 的取值

符号常数	数 值	用 于
LEFT __ TEXT	0	水平
RIGHT __ TEXT	2	水平
BOTTOM __ TEXT	0	垂直
TOP __ TEXT	2	垂直
CENTER __ TEXT	1	水平或垂直

```
void far settextstyle(int font,int direction,int charsize);
```

带函数用来设置输出字符的字型(由 `font` 确定)。输出方向(由 `direction` 确定)和字符大小(由 `charsize` 确定)等特性,各参数的规定如表 7-7,表 7-8,表 7-9 所示。

表 7-7 font 的取值

符号常数	数 值	含 义
DEFAULT __ FONT	0	8×8 点阵字(缺省值)
TRIPLEX __ FONT	1	三倍笔划字体
SMALL __ FONT	2	小号笔划字体
SANSERIF __ FONT	3	无衬线笔划字体
GOTHIC __ FONT	4	黑体笔划字

表 7-8 direction 的取值

符号常数	数 值	含 义
HORIZ __ DIR	0	从左到右
VERT __ DIR	1	从底到顶

表 7-9 charsize 的取值

符号常数或数值	含 义
1	8×8 点阵
2	16×16 点阵
3	24×24 点阵
4	32×32 点阵
5	40×40 点阵
6	48×48 点阵
7	56×56 点阵
8	64×64 点阵
9	72×72 点阵
10	80×80 点阵
USER __ CHAR __ SIZE=0	用户定义的字符大小

有关图形屏幕下文本输出和字体字型设置函数的用法请看下例：

```

/*
  OUTXT1.C——图形方式下文本输出和字型字体设置之一
*/
#include "graphics.h"
#include "conio.h"
void InitGra(void);
void OuTxt();
int main()
{
    InitGra();
    OuTxt();
    getch();
    closegraph();
    return 0;
}
void InitGra(void)
{
    int GraphDrive = DETECT, GraphMode;    /* 定义为 DETECT 方式 */
    registerbgidriver(EGAVGA __ driver);    /* 登记的图形驱动名 */
    registerbgifont(triplex __ font);    /* 使用三倍笔划字体 */
}

```

```

    registerbgifont(small _ font);          /* 使用小号笔划字体 */
    registerbgifont(sansserif _ font);      /* 使用无衬线笔划字体 */
    registerbgifont(gothic _ font);        /* 使用黑体笔划字体 */
    initgraph(&GraphDrive,&GraphMode,"");
}
void OutTxt()
{
    setcolor(YELLOW);
    rectangle(0,0,439,279);                /* 画一个黄色的矩形框 */
    setcolor(12);
    settextstyle(1, 0, 8);    /* TRIPLEX _ FONT 字,水平输出放大 8 倍 */
    outtextxy(20,20,"Good News");          /* 在指定位置输出文本 */
    setcolor(15);
    settextstyle(3,0,5);    * SANSERIF _ FONT 字,水平输出放大 8 倍 */
    outtextxy(120,120,"Good News");
    setcolor(2);
    settextstyle(2,0,8);          /* SMALL _ FONT 字,水平输出放大 8 倍 */
    outtextxy(180,200,"GOOD");
    setcolor(14);
    settextstyle(4,0,3); /* 设置 GOTHIC-FONT 字,水平输出放大 3 倍 */
    outtextxy(50,240,"WELCOME TO USE THIS BOOK");
}

```

### 7.5.3 用户对文本字符大小的设置

前面介绍的 `settextstyle()` 函数,可以设置图形方式下输出文本字符的字体和大小,但对于笔划型字体(除  $8 \times 8$  点阵字符以外),只能在水平和垂直方向以相同的放大倍数放大。为此,Turbo C/Borland C 又提供了另外一个函数 `setusercharsize()`,对笔划字体可以分别设置水平和垂直方向的放大倍数。其调用格式如下:

```
void far setusercharsize(int mulx,int divx,int muly,int divy);
```

该函数用来设置笔划型字的放大系数,它只有在 `settextstyle()` 函数中的 `charsize` 为 0 (`USER _ CHAR _ SIZE`)时才起作用,并且字体为函数 `settextstyle()` 规定的字体。调用函数 `setusercharsize()`后,每个显示在屏幕上的字符都以其缺省大小乘以 `mulx/divx` 为输出宽度,乘以 `muly/divy` 为输出字符高。该函数的用法见下例:

```
/*
```

```
    OUTXT2.C—— 图形方式下文本输出和字型字体设置之二
```

```
*/
```

```
#include "graphics.h"
```

```
#include "conio.h"
```

```

void InitGra(void);
void OuTxt();
int main()
{
    InitGra();
    OuTxt();
    getch();
    closegraph();
    return 0;
}

void InitGra(void)
{
    int GraphDrive = DETECT, GraphMode; /* 定义为 DETECT 方式 */
    registerbgidriver(EGAVGA_driver); /* 登记的图形驱动名 */
    registerbgifont(triplex_font); /* 使用三倍笔划字体 */
    registerbgifont(small_font); /* 使用小号笔划字体 */
    registerbgifont(sansserif_font); /* 使用无衬线笔划字体 */
    registerbgifont(gothic_font); /* 使用黑体笔划字体 */
    initgraph(&GraphDrive, &GraphMode, "");
}

void OuTxt()
{
    setcolor(YELLOW);
    setfillstyle(1, 2);
    rectangle(100, 100, 540, 380); /* 画一黄色矩形框 */
    setcolor(12);
    settextstyle(1, 0, 8); /* 三重笔划字, 放大 8 倍 */
    outtextxy(120, 110, "Good News");
    setusercharsize(2, 1, 4, 1); /* 水平放大 2 倍, 垂直放大 4 倍 */
    settextstyle(1, 0, 0); /* 选字体和用户定义倍数 */
    setcolor(10);
    outtextxy(150, 210, "GOOD NEWS");
    setcolor(13);
    settextstyle(3, 0, 5); /* 用无衬线笔划写字, 放大 5 倍 */
    outtextxy(220, 190, "Good News");
    setusercharsize(4, 1, 1, 1); /* 水平放大 4 倍, 垂直放大 1 倍 */
    setcolor(14);
    settextstyle(3, 0, 0); /* 选字体和用户定义倍数 */
    outtextxy(180, 330, "GOOD");
}

```



```
}
```

作为图形状态下文字输出的例子,SYSDAT.C 是以日历牌的形式显示系统日期的程序,首先用 BIOS 功能调用读出系统日期,然后显示,同时也结合了西文 DOS 下的汉字显示技术。

```
/*
```

SYSDAT.C——以日历牌的形式显示系统日期

```
*/
```

```
#include "bios.h"
```

```
#include "stdio.h"
```

```
#include "stdlib.h"
```

```
#include "graphics.h"
```

```
#include "conio.h"
```

```
#include "dos.h"
```

```
FILE *fp16;
```

```
void InitGra();
```

```
void Quit();
```

```
void OpenLIB();
```

```
void PutCC16(int,int,int,int,char * Str);
```

```
void SysDate();
```

```
int main()
```

```
{
```

```
    InitGra();
```

```
    OpenLIB();
```

```
    SysDate();
```

```
    getch();
```

```
    Quit();
```

```
    return 0;
```

```
}
```

```
void InitGra()
```

```
{
```

```
    int GraphDrive = DETECT,GraphMode;
```

```
    registerbgidriver(EGAVGA _driver);    /* 登记的图形驱动名 */
```

```
    registerbgifont(triplex _font);        /* 使用三倍笔划字体 */
```

```
    registerbgifont(small _font);          /* 使用小号笔划字体 */
```

```
    registerbgifont(sansserif _font);      /* 使用无衬线笔划字体 */
```

```
    registerbgifont(gothic _font);         /* 使用黑体笔划字体 */
```

```
    initgraph(&GraphDrive,&GraphMode,"");
```

```
}
```

```
void Quit()
```

```
{
```

```

    fclose(fp16);
    closegraph();
    exit(0);
}

void OpenLIB()
{
    fp16 = fopen("c:\\ucdos\\hzk16", "rb");
}

void PutCC16(int x,int y,int Wid,int Color,char * Str)
{
    unsigned i,Code1,Code2;          /* 区码,位码 */
    int i1,i2,i3,Rec;
    long Len;
    char Buf[32],OldColor;
    OldColor = getcolor();
    while (* Str)                    /* 直到字符串显示完 */
    {
        if ((* Str & 0x80) && (*( Str+1) & 0x80)) /* 是汉字 */
        {
            Code1 = (* Str-0xa1) & 0x07f;      /* 区码 */
            Code2 = (*( Str+1)-0xa1) & 0x07f; /* 位码 */
            Rec = Code1 * 94 + Code2;          /* 记录号 */
            Len = Rec * 32L;                   /* 在字库中位置 */
            fseek(fp16,Len,SEEK _SET);
            fread (Buf,1,32,fp16);             /* 32 字节 */
            for (i1 = 0; i1<16; i1++)          /* 垂直方向 16 点 */
                for (i2 = 0; i2<2; i2++)        /* 水平方向 2 字节 */
                    for (i3 = 0; i3<8; i3++)    /* 每字节 8 位 */
                        if (Buf[i1 * 2+i2]>> (7-i3) & 1) /* 是 1 画点 */
                            putpixel(x+i2 * 8+i3,y+i1,Color);
            x = x+16+Wid;
            Str += 2;
        }
    }
    setcolor(OldColor);
    return;
}

void SysDate()
{

```

```

char *Wek0 = "星期";
char *Mon0 = "月";
char *Week[] = {"日","一","二","三","四","五","六"};
char Year[5],Month[3],Date[3];
union REGS In,Out;
int x = 300,y = 150;
int l = 88,h = 120;
setfillstyle(1,7);
bar(x,y,x+l,y+h);
setcolor(15);
rectangle(x+l,y+l,x+l-1,y+h-1);
rectangle(x+3,y+3,x+l-3,y+h-3);
In.h.ah = 0x2a;
int86(0x21,&In,&Out);
sprintf(Month,"%2d",Out.h.dh);
sprintf(Year,"%4d",Out.x.cx);
sprintf(Date,"%2d",Out.h.dl);
setcolor(14);
settextstyle(1,0,1);
outtextxy(x+27,y+6,Month);
PutCC16(x+48,y+9,0,14,Mon0);
settextstyle(1,0,4);
setcolor(15);
outtextxy(x+12,y+24,Year);
setcolor(9);
outtextxy(x+27,y+57,Date);
PutCC16(x+21,y+95,1,11,Wek0);
PutCC16(x+56,y+95,1,11,Week[Out.h.al]);
}

```

## 7.6 格式化数据输出

图形状态下,在输出文本信息时,Turbo C/Borland C 所提供的文本输出库函数还存在着几个不足之处:

- (1) 未提供格式化输出函数;
- (2) 覆盖已在屏幕上的文本时,不擦除旧文本;
- (3) 不支持突出显示文本。

针对这些不足,本节通过讨论,提供几种解决办法:

### 7.6.1 格式化文本输出

Turbo C/Borland C 提供了图形状态下的文本输出库函数 `outtext()` 和 `outtextxy()`，但不能进行格式化输出。但 Turbo C/Borland C 还提供函数 `vsprintf()`，其功能是将格式化文本信息输出到字符串中。因此，我们可以利用这两种函数编写格式化的输出函数 `OutFmTxt()` 进行文本格式输出。

```
void OutFmTxt(int x,int y,char *fmt,...) /* 格式化文本输出 */
{
    va _list argptr;
    char Txt[255];
    va _start(argptr,fmt);
    vsprintf(Txt,fmt,argptr);
    va _end(argptr);
    outtextxy(x,y,Txt);
}
```

### 7.6.2 文本覆盖

Turbo C/Borland C 提供的图形方式下的文本输出函数只是作了一种 OR 运算，这样会使新文本受到破坏，解决这个问题的较好的方法是利用库函数 `bar()` 画适当大小的实心矩形，并用背景色填充，然后输出新文本。

### 7.6.3 突出显示文本

突出显示文本的方法主要有：反相显示、闪烁显示等，可参看下面程序中介绍及实现方法。反相显示可用 `bar()` 函数先画出一背景色区域，然后再写文字；闪烁显示提供的方式是：先输出文本→保存文本信息→`bar()` 函数清除掉文本→延时 400 毫秒→等待按键，若无键按下，则继续闪烁显示。

```
/*
    FMTOUT.C—— 图形方式文本格式输出
*/
#include "conio.h"
#include "string.h"
#include "stdlib.h"
#include "stdarg.h"
#include "graphics.h"
#include "stdio.h"
```

```

#include "dos.h"
#include "bios.h"
void OutFmTxt(int x,int y,char *fmt,...) ;
void Revs(int x,int y,char *Txt,int BkColor,int TColor) ;
void InitGra(void) ;
int main(void)
{
    char Msg1[] = "Hello!";
    char Msg2[] = "How do you do?";
    char Msg3[] = "Good Morning!";
    int Age = 30,Wid,Hit;
    struct textsettingstype TxtInf;
    InitGra();
    OutFmTxt(10,10,"I am %d years old.",Age); /* 格式化文本输出 */
    getch();
    settextstyle(DEFAULT __FONT,HORIZ __DIR,1);
    outtextxy(10,50,Msg1);
    getch();
    Hit = textheight(Msg1);
    Wid = textwidth(Msg1);
    gettextsettings(&TxtInf);
    setfillstyle(SOLID __FILL,0); /* 文本覆盖 */
    if (TxtInf.font) bar(10,50+Hit/4,10+Wid,50+Hit+Hit/4);
    else bar(10,50,10+Wid,50+Hit);
    outtextxy(10,50,Msg2);
    getch();
    settextstyle(DEFAULT __FONT,HORIZ __DIR,1);
    Revs(10,100,Msg3,WHITE,BLACK); /* 反相显示 */
    getch();
    settextstyle(DEFAULT __FONT,HORIZ __DIR,1);
    Hit = textheight(Msg1);
    Wid = textwidth(Msg2);
    setcolor(WHITE);
    setfillstyle(SOLID __FILL,BLACK);
    for (;) /* 闪烁显示 */
    {
        outtextxy(10,150,Msg1);
        delay(400);
        gettextsettings(&TxtInf);
    }
}

```

```

        if (TxtInf.font) bar(10,150+Hit/4,10+Wid,150+Hit+Hit/4);
        else bar(10,150,10+Wid,150+Hit);
        delay(400);
        if (bioskey(1)) break;
        else continue;
    }
    getch();
    closegraph();
}

void OutFmTxt(int x,int y,char *fmt,...) /* 格式化文本输出 */
{
    va __list argptr;
    char Txt[255];
    va __start(argptr,fmt);
    vsprintf(Txt,fmt,argptr);
    va __end(argptr);
    outtextxy(x,y,Txt);
}

void Revs(int x,int y,char *Txt,int BkColor,int TColor) /* 反相显示 */
{
    struct textsettingstype TxtInf;
    int Wid,Hit;
    gettextsettings(&TxtInf);
    Hit = textheight(Txt);
    Wid = textwidth(Txt);
    if (TxtInf.font) Hit++;
    setfillstyle(1,BkColor);
    bar(x,y,x+Wid-1,y+Hit-1);
    setcolor(TColor);
    if (TxtInf.font) outtextxy(x,y-Hit/4,Txt);
    else outtextxy(x,y,Txt);
}

void InitGra(void)
{
    int GraphDrive = DETECT,GraphMode; /* 定义为 DETECT 方式 */
    registerbgidriver(EGAVGA_driver); /* 登记的图形驱动名 */
    registerbgifont(triplex_font); /* 使用三倍笔划字体 */
    registerbgifont(small_font); /* 使用小号笔划字体 */
    registerbgifont(sansserif_font); /* 使用无衬线笔划字体 */
}

```

```

    registerbgifont(gothic __font);          /* 使用黑体笔划字体 */
    initgraph(&GraphDrive,&GraphMode,"");
}
void OutTxt()
{
    setcolor(YELLOW);
    setfillstyle(1,2);
    rectangle(100,100,540,380);             /* 画一黄色矩形框 */
    setcolor(12);
    settextstyle(1,0,8);                    /* 三重笔划字,放大 8 倍 */
    outtextxy(120,110,"Good News");
    setusercharsize(2,1,4,1);              /* 水平放大 2 倍,垂直放大 4 倍 */
    settextstyle(1,0,0);                    /* 选字体和用户定义倍数 */
    setcolor(10);
    outtextxy(150,210,"GOOD NEWS");
    setcolor(13);
    settextstyle(3,0,5);                    /* 用无衬线笔划写字,放大 5 倍 */
    outtextxy(220,190,"Good News");
    setusercharsize(4,1,1,1);              /* 水平放大 4 倍,垂直放大 1 倍 */
    setcolor(14);
    settextstyle(3,0,0);                    /* 选字体和用户定义倍数 */
    outtextxy(180,330,"GOOD");
}

```

## 7.7 西文字符的放大、旋转与倾斜显示

在 Turbo C/Borland C 中,可用函数 `outtext()`,`outtextxy()` 和 `settextstyle()` 来在图形方式下输出任意大小、水平或垂直两个方向、任意位置、5 种字体的字符串。但它有一个很大的缺点,就是不能在任意方向上以任意间距显示字符串,只能在纵横两个方向显示,再者,字符也不能倾斜显示。而我们在编制界面程序时,又常常可能要在任意方向上以任意大小、任意字符间距、任意倾斜角度来显示字符串。为此,可以直接读取 TurboC/BorlandC 提供的矢量字库 \*.CHR 来进行显示。

### 7.7.1 矢量字库 \*.CHR 的结构分析

BGI 矢量字体文件 \*.CHR 分为文件头和数据区两部分,文件头长 80H 字节。

表 7-10

CHR 文件的结构

偏 移	内 容
00H _ 58H	一些文字信息
59H _ 5AH	文件头长度
5BH _ 5EH	字库文件名
5FH _ 60H	数据长度
80H	字体文件类型标识
81H _ 82H	文件中定义的字符个数
84H	第一个字符的 ASCII 码值,一般为 20H
85H _ 86H	笔划定义区的相对位移,是相对于 80H 的
88H	字符在基线上的高度
8AH	字符在基线以下的高度,为一负数

字符的矢量数据按字符的 ASCII 码值由小到大的顺序排列。假定  $m$  为定义的字符数,则从偏移 90H 开始即为长  $2m$  字节的索引表,每两个字节给出了相对于矢量定义区首到对应字符矢量定义区的位移。该表之后是长为  $m$  个字节的字符宽度表,记录每个字符的宽度,宽度表之后是矢量定义区。

BGI 矢量字库中每个字符的定义由若干字组成,每个字表示一笔,第 1 字节和第 2 字节的最高位分别为操作码  $oc1$  和  $oc2$ ,低 7 位分别为 X 坐标和 Y 坐标(带符号位),操作码的含义如下:

- (1) 当  $oc1 = 0, oc2 = 0$  时,表示字符矢量定义结束;
- (2) 当  $oc1 = 1, oc2 = 0$  时,表示抬笔移至  $(x, y)$  处;
- (3) 当  $oc1 = 1, oc2 = 1$  时,表示落笔从当前位置画至  $(x, y)$  处;
- (4) 当  $y < 128$  时,  $y = y - 128$ ;

### 7.7.2 放大、旋转与倾斜

字符的放大可在  $x$  方向和  $y$  方向上进行,只要将坐标乘以放大系数即可实现。

对于旋转和倾斜,只要按 3.2.1.3 中的方法乘以旋转矩阵和错切矩阵即可,为了读者阅读方便,在此不妨再列出这两个矩阵:

旋转矩阵:  $R = \begin{pmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{pmatrix}$   $\alpha$  为旋转角。

即:  $(x_2 \ y_2) \times R = (x_1 \cos\alpha - y_1 \sin\alpha \ x_1 \sin\alpha + y_1 \cos\alpha)$

错切矩阵:  $T = \begin{pmatrix} 1 & 0 \\ \tan\theta & 1 \end{pmatrix}$  其中,  $\theta$  为倾斜角。

即:  $(x_2 \ y_2) \times T = (x_1 + y_1 \cdot \tan\theta \ y_1)$ ,



在显示过程中,由于屏幕的左上角为原点,所以,显示出来的字符为倒置,为了将其正置,需要作如下处理:

以字符 A 为例:若以屏幕左上角的 0 为坐标原点,则显示的字符为倒置;若以左下角 01 为原点,则显示的字符就是正置。字符上的某一点 P,在 X0Y 坐标系中,其坐标为(x,y),那么,在 X101Y1 坐标系中为(x1,y1),其中:  $x1 = x$ ;  $y1 = Hid - y$  (Hid 为字符高度)。

下面是源程序,处理了 4 种字体的任意放大、旋转、倾斜显示:

```
/*
ROUTXT.C——图形方式 ASCII 字符的放大旋转倾斜显示
*/
#include "stdio.h"
#include "graphics.h"
#include "math.h"
#include "string.h"
#include "conio.h"
#include "process.h"
#define PI 3.14159
void InitGra(void);
void Quit(void);
void Write(int,int,char *ch,float,float,int,int,int,char *Name);
void Conv(int,int,int,int,int,int,int *x1,int *y1);
int main(void)
{
    InitGra();
    Write(100,0,"ABCDEFGH",1.0,1.0,-36,20,10,"trip.chr");
    Write(200,450,"ABCDEFGH",1.5,1.5,36,20,10,"sans.chr");
    Write(200,230,"ABCDEFGH",2.0,2.0,0,30,10,"goth.chr");
    Write(100,160,"ABCDEFGH",5.0,3.0,0,25,10,"litt.chr");
    getch();
    Quit();
    return 0;
}
void InitGra(void)
{
    int GraphDrive = DETECT,GraphMode;
    registerbgidriver(EGAVGA__driver);
    initgraph(&GraphDrive,&GraphMode,"");
}
void Quit(void)
{

```

```

    closegraph();
    fcloseall();
    exit (0);
}
/*
    x0,y0: 字符的起始位置;
    Str: 要显示的字符串;
    Xt,Yt: X,Y 方向的放大倍数;
    Af: 旋转角;
    Bt: 倾斜角;
    Zj: 字符间距;
    Name: 矢量字库文件名.
*/
void Write(int x0,int y0,char * Str,float Xt,float Yt,int Af,int Bt,int Zj,char * Name)
{
    FILE *fp;
    unsigned char c,c1,c2,UpBs,Hit,Wid,FstAsc,Num;
    int i,Stx = 0,x1,y1,x2,y2,x11,y11,x22,y22,wy;
    char BlBs;
    fp = fopen(Name,"rb");
    fseek(fp,0x81L,SEEK_SET);
    Num = getw(fp);          /* 字体文件中的字符数 */
    fseek(fp,0x84L,SEEK_SET);
    FstAsc = getc(fp);       /* 第一个字符的 ASCII 值 */
    fseek(fp,0x88L,SEEK_SET);
    UpBs = getc(fp);         /* 字符在基线以上的高度 */
    fseek(fp,0x8aL,SEEK_SET);
    BlBs = getc(fp);         /* 字符在基线以下的高度 */
    Hit = UpBs - BlBs;
    for(i = 0; i < strlen(Str); i++) /* 字符的高度 */
    {
        c = *(Str+i);
        fseek(fp,0x90+(c-FstAsc)*2,SEEK_SET);
        wy = getw(fp); /* 相对于矢量定义区首,到对应字符矢量定义区的位移 */
        fseek(fp,0x90+Num*2+c-FstAsc,SEEK_SET);
        Wid = getc(fp);    /* 字符宽度 */
        fseek(fp,0x90+Num*3+wy,SEEK_SET);
        c1 = getc(fp);     /* 操作码 */
        c2 = getc(fp);
    }
}

```

```

x1 = c1 & 0x07f;          /* 矢量数据 */
y1 = c2 & 0x07f;
if ((y1-128) >= BlBs && (y1-128) < 0) y1 -= 128;
for(;;)
{
    c1 =getc(fp);          /* 操作码 */
    c2 =getc(fp);
    x2 = c1 & 0x07f;        /* 矢量数据 */
    y2 = c2 & 0x07f;
    if ((y2-128) >= BlBs && (y2-128) < 0) y2 -= 128;
    c1 = c1 >> 7;
    c2 = c2 >> 7;
    if (c1 == 0 && c2 == 0) break;
    if (c1 == 1 && c2 == 0)
    {
        x1 = x2;
        y1 = y2;
        continue;
    }
    Conv(x0,y0,Af,Bt,Stx+x1 * Xt,Hit * Yt-y1 * Yt,&x11,&y11);
    Conv(x0,y0,Af,Bt,Stx+x2 * Xt,Hit * Yt-y2 * Yt,&x22,&y22);
    line(x11,y11,x22,y22);
    x1 = x2;
    y1 = y2;
}
Stx = Stx+Wid * Xt+Zj;
}
fclose(fp);
}

void Conv(int g,int h,int Af,int Bt,int x,int y,int *x1,int *y1)
{
    float Af1,Bt1;
    Af1 = -PI * Af/180;
    Bt1 = -PI * Bt/180;
    *x1 = g+x * cos(Af1)-y * sin(Af1);
    *y1 = h+x * sin(Af1)+y * cos(Af1);    /* 旋转 */
    *x1 = *x1+*y1 * tan(Bt1);              /* 倾斜 */
}

```

# 图符及图符菜单

## 第 8 章

本章内容：

8.1 创建图符

8.2 图符菜单设计

图符(ICON)式界面越来越受到用户的欢迎和程序员的重视,它用一个个形象的小图符来代替文字叙述,表达系统的功能,用户界面非常直观和易于使用。图符界面是否友好,关键在于所创建的图符是否形象化地表达了该选择项所对应的功能。因此,图符的创建是非常重要的。

## 8.1 创建图符

在图形方式下,图符的创建比较简单,只要用图形函数将其画出即可。在文本方式下却不能采用这种方法,不过我们可以编制一个图符编辑程序,先将一个个图符编辑好,保存在文件中,然后再从文件中读出,即可显示出相应的图符。

图符编辑器在工作时,首先在屏幕上显示一个放大的图符基座,用户使用光标键,在该图符基座上绘制所需要的图符。同时,正常大小的图符也显示在屏幕上(它随放大的图符的变化而变化),每一个图符作为一种图形映象出现在屏幕上,它跟其结构相关联,该结构也包含了和图符有关的数据信息。每一个图符的数据信息由三个部分构成:图符的图形映象(保存在二维数组中);当图符被选中时;它应调用的函数指针;图符在屏幕上的位置坐标。

```
struct IconType {                                /* 图符结构定义 */
    unsigned char Image[XDIM][YDIM];           /* 图符的图形映象 */
    void (* func)();                             /* 当图符被选中时应调用的函数指针 */
    int x,y;                                     /* 图符在屏幕上的位置坐标 */
} Icon;
```

图符编辑器由以下几个部分构成:初始化图符、显示正常的图符、显示放大的图符、光标键绘制图符、保存图符到文件、从文件中装载图符。

下面是完整的图符编辑器的源程序清单:

```
/*
    CrtIcon.C——创建图符程序(图符编辑器)
*/
#include "stdio.h"
#include "dos.h"
#include "conio.h"
#include "bios.h"
#include "string.h"
#define IX 240                                    /* 编辑图符的位置 */
#define IY 120
#define XDIM 20                                  /* 正常图符的大小 */
#define YDIM 16
#define XPAND 3                                  /* 放大编辑图符的放大倍数 */
#define BCOLOR 9                                /* 图符的背景色 */
#define FCOLOR 15                                /* 图符的前景色 */
```

```

#define LIGHT 13
#define LEFT 331
#define RIGHT 333
#define UPPER 328
#define DOWN 336
#define HOME 327
#define END 335
#define PGUP 329
#define PGDN 337
#define ESC 27
#define F1 315
#define F2 316
#define F3 317
#define F4 318
#define F5 319
void DispGrid();
void EditIcon();
void DispIcon(int,int);
void InitIcon();
int SavIcon();
int LoadIcon();
int GetKey(void);
char GetPoint(int x,int y);
void PutPoint(int x,int y,int color,int how);
struct IconType {
    unsigned char Image[XDIM][YDIM]; /* 图符的图形映象 */
    void (* func)(); /* 当图符被选中时应调用的函数指针 */
    int x,y; /* 图符在屏幕上的位置坐标 */
} Icon;
int main()
{
    clrscr();
    textattr(0x0e);
    cprintf("F1-画笔打开 F2-关闭画笔 F3-保存图符");
    cprintf("F4-装载图符 F5-删处图符 ESC-退出编辑");
    InitIcon(); /* 初始化图符 */
    EditIcon(); /* 编辑图符 */
    clrscr();
    return 0;
}

```

```

}
void DispGrid()                /* 显示放大的图符 */
{
    int x,y;
    for (y = IY; y < IY+YDIM; y++)
        for (x = IX; x < IX+XDIM; x++)
            PutPoint(x+((x-IX)*XPAND),y+((y-IY)*XPAND),Icon. Image[x-IX]
                [y-IY],0);
}

void EditIcon()                /* 图符交互编辑 */
{
    int x,y,Key;
    char Pen,Temp;
    x = IX; y = IY;
    Pen = FCOLOR;
    DispIcon(0,IY);
    DispGrid();
    do
    {
        PutPoint(x+((x-IX)*XPAND),y+((y-IY)*XPAND),Pen,0);
                                /* 栅格内写点 */

        Icon. Image[x-IX][y-IY] = Pen;
        PutPoint(x-IX,y,Icon. Image[x-IX][y-IY],0);
        Temp = GetPoint(x+((x-IX)*XPAND),y+((y-IY)*XPAND));
        PutPoint(x+((x-IX)*XPAND),y+((y-IY)*XPAND),LIGHT,0);
                                /* 图符内写点 */

        Key = GetKey();        /* 读取功能键 */
        PutPoint(x+((x-IX)*XPAND),y+((y-IY)*XPAND),Temp,0);
        switch(Key)
        {
            case LEFT: x--; break;
            case RIGHT: x++; break;
            case UPPER: y--; break;
            case DOWN: y++; break;
            case HOME: x--, y--; break;
            case PGUP: x++, y--; break;
            case END: x--, y++; break;
            case PGDN: x++, y++; break;
            case F1: Pen = FCOLOR; break;

```

```

    case F2: Pen = BCOLOR; break;
    case F3: SavIcon(); break;
    case F4:
        LoadIcon();
        DispIcon(0,IY);
        DispGrid();
        break;
    case F5:
        InitIcon();
        DispIcon(0,IY);
        DispGrid();
        break;
}
if (x < IX) x++;
if (x > IX+XDIM-1) x--;
if (y < IY) y++;
if (y > IY+YDIM-1) y--;
} while (Key != ESC);          /* ESC 键退出编辑 */
}

void DispIcon(int x0,int y0) /* 显示原始图符 */
{
    int x,y;
    for (y = y0; y < y0+YDIM; y++)
        for (x = x0; x < x0+XDIM; x++)
            PutPoint(x,y,Icon.Image[x-x0][y-y0],0);
}

void InitIcon()                /* 图符初始化 */
{
    int x,y;
    for (x = 0; x < XDIM; x++)
        for (y = 0; y < YDIM; y++)
            Icon.Image[x][y] = BCOLOR;
}

int SavIcon()                  /* 保存图符文件 */
{
    FILE *fp;
    char FilNam[80];
    int Res;
    gotoxy(10,22);

```



```

    cprintf("保存图符到文件:");
    gets(FilNam);
    if ((fp = fopen(FilNam,"wb")) == NULL)
    {
        printf("文件打不开\n");
        return 0;
    }
    fwrite(&Icon,sizeof(Icon),1,fp); /* 写图符文件 */
    if (ferror(fp)) Res = 0;
    else Res = 1;
    fclose(fp);
    gotoxy(1,22);
    clrcl();
    return Res;
}

int LoadIcon() /* 读取图符文件 */
{
    FILE *fp;
    char FilNam[80];
    int Res;
    gotoxy(10,22);
    cprintf("装载图符从文件:");
    gets(FilNam);
    if ((fp = fopen(FilNam,"rb")) == NULL)
    {
        printf("文件打不开");
        return 0;
    }
    fread(&Icon,sizeof(Icon),1,fp);
    if (ferror(fp)) Res = 0;
    else Res = 1;
    fclose(fp);
    gotoxy(1,22);
    clrcl();
    return Res;
}

int GetKey(void) /* 读取功能键 */
{
    int Ch,Low,Hig;

```

```

    Ch = bioskey(0);
    Low = Ch & 0x00ff;
    Hig = (Ch & 0xff00) >> 8;
    return (Low == 0 ? Hig + 256 : Low);
}

char GetPoint(int x,int y)
{
    union REGS r;
    r.h.ah = 13;
    r.x.dx = y;
    r.x.cx = x;
    int86(0x10,&r,&r);
    return r.h.al;
}

void PutPoint(int x,int y,int color,int how)
{
    union REGS r;
    if (how == 0x18) color = color | 128;

    r.h.ah = 12;           /* 功能号 */
    r.x.dx = y;           /* Y坐标 */
    r.x.cx = x;           /* X坐标 */
    r.h.al = color;       /* 象素值 */
    int86(0x10,&r,&r);
}

```

## 8.2 图符菜单设计

基于图符的菜单仍是菜单。因此,它具有菜单的最基本的功能,如显示当前活动的图符,等待用户选择,并在选择后能采用适当的操作。

在基于图符的界面中,最重要的函数是获取用户输入的函数。它完成三个主要功能:显示形成图符菜单的图符;允许用户选择一个图符;把用户选择返回到调用例程。

下面是一个完整的图符菜单程序,它可以通过光标的移动完成对图符的选择,然后执行相应的功能(各子功能都很简单,为节省篇幅,例程中没有写出,读者可采用进程控制函数 `system()` 或 `spawnl()` 自行编制)。程序运行前,应首先用 `CRTICON.EXE` 创建下列的图符文件,并确保在当前子目录下:

```

copy.ico
del.ico

```

```

    dir.ico
    type.ico
    run.ico
    chkdsk.ico
    ver.ico
    end.ico
/*
    IconMu.C——图符菜单例子
*/
#include "stdio.h"
#include "dos.h"
#include "conio.h"
#include "bios.h"
#include "stdlib.h"
#include "process.h"
#include "ctype.h"
#include "string.h"
#define IX 200          /* 图符的显示位置 */
#define IY 100
#define XDIM 20         /* 图符 X 方向的大小 */
#define YDIM 16         /* 图符 Y 方向的大小 */
#define ICONNUM 10      /* 图符的个数 */
#define NUM 4           /* 每行图符的个数 */
#define ICONXS 50       /* 图符 X 方向间距 */
#define ICONYS 50       /* 图符 Y 方向间距 */
#define BCOLOR 9        /* 图符背景颜色 */
#define FCOLOR 15       /* 图符前景颜色 */
#define LIGHT 13        /* 高亮度颜色 */
#define LEFT 331        /* 以下是功能键定义 */
#define RIGHT 333
#define UPPER 328
#define DOWN 336
#define ESC 27
#define ENTER 13
struct IconType {        /* 图符结构定义 */
    unsigned char Image[XDIM][YDIM]; /* 图符的图形映象 */
    void (* func)();      /* 当图符被选中时应调用的函数指针 */
    int x,y;             /* 图符在屏幕上的位置坐标 */
} Icons[ICONNUM];

```

```

int IconMenu(struct IconType *Icon,int Num);
void RevIcon(struct IconType Icon);
void DispIcon(struct IconType Icon);
void DispIcons(struct IconType *Icon,int Num);
int LoadIcon();
void Dir();
void Copy();
void Del();
void Run();
void Type();
void ChkDsk();
void Ver();
void End();
int GetKey(void);
void PutPoint(int x,int y,int color,int how);
char * IconFile[]={"Copy.ico","Del.ico","Dir.ico","Type.ico",/* 图符文件名 */
    "Run.ico","ChkDsk.ico","Ver.ico","End.ico",""};
void (* SysFun[])()={Copy,Del,Dir,Type,Run,ChkDsk,Ver,End};
    /* 子功能函数名字 */

int NumIcon;
int main()
{
    int Sel,i,j;
    clrscr();
    if ((NumIcon = LoadIcon()) == 0) exit (1);
    for (i = 0,j = 0; i < NumIcon; i++,j++) /* 显示出各功能图符 */
    {
        if (! (j % NUM)) j = 0;
        Icons[i].x = j * ICONXS+IX;
        Icons[i].y = IY+(ICONYS * (i/NUM));
    }
    for (i = 0; i < NumIcon; i++) Icons[i].func = SysFun[i];
    for (;;)
    {
        textbackground(0);
        textcolor(15);
        Sel = IconMenu(Icons,NumIcon); /* 图符菜单中等待用户选择 */
        (* Icons[Sel].func)(); /* 根据用户选择执行相应功能 */
        textcolor(0);
    }
}

```

```

        clrscr();
    }
    return 0;
}

int IconMenu(struct IconType *Icons,int Num) /* 图符菜单选择 */
{
    int Sel,Key;
    DispIcons(Icons,Num);
    gotoxy(25,15);
    textattr(0x0e);
    cprintf("箭头键选择,ENTER 键执行");
    Sel = 0;
    for (; )
    {
        RevIcon(Icons[Sel]);          /* 反相显示 */
        Key = GetKey();                /* 等待用户按键 */
        DispIcon(Icons[Sel]);          /* 正常显示 */
        switch(Key)
        {
            case LEFT: Sel --; break;
            case RIGHT: Sel ++; break;
            case UPPER: Sel -= NUM; break;
            case DOWN: Sel += NUM; break;
            case ENTER:
                gotoxy(2,20);
                clrscr();
                return Sel;
            case ESC:
                End();
        }
        if (Sel < 0) Sel = NumIcon-1;
        if (Sel > NumIcon-1) Sel = 0;
    }
}

void RevIcon(struct IconType Icon) /* 反相显示图符 */
{
    int x,y;
    for (y = Icon.y; y < Icon.y+YDIM; y++)
        for (x = Icon.x; x < Icon.x+XDIM; x++)

```

```

        if (Icon.Image[x-Icon.x][y-Icon.y] == BCOLOR) PutPoint(x,y,FCOLOR,
            0);
        else PutPoint(x,y,BCOLOR,0);
    }
}

void DispIcon(struct IconType Icon) /* 显示一个图符 */
{
    int x,y;
    for (y = Icon.y; y < Icon.y+YDIM; y++)
        for (x = Icon.x; x < Icon.x+XDIM; x++)
            PutPoint(x,y,Icon.Image[x-Icon.x][y-Icon.y],0);
}

void DispIcons(struct IconType *Icons,int Num) /* 显示所有图符 */
{
    int i;
    for (i = 0; i < Num; i++) DispIcon(Icons[i]);
}

int LoadIcon() /* 装载图符文件 */
{
    FILE *fp;
    int i;
    for (i = 0; (i < ICONNUM) && *IconFile[i] != '\0'; i++)
    {
        if ((fp = fopen(IconFile[i],"rb")) == NULL)
        {
            fprintf("文件 %s 打不开!",IconFile[i]);
            return 0;
        }
        fread(&Icons[i],sizeof(struct IconType),1,fp);
        if (ferror(fp)) break;
        fclose(fp);
    }
    return i;
}

int GetKey(void)
{
    int Ch,Low,Hig;
    Ch = bioskey(0);
    Low = Ch & 0x00ff;
    Hig = (Ch & 0xff00) >> 8;
}

```

```

    return (Low == 0 ? Hig+256 : Low);
}
void PutPoint(int x,int y,int color,int how)
{
    union REGS r;
    if (how == 0x18) color = color|128;
    r.h.ah = 12;
    r.x.dx = y;
    r.x.cx = x;
    r.h.al = color;
    int86(0x10,&r,&r);
}
void Copy(){
}
void Del(){
}
void Dir(){
}
void Run(){
}
void Type(){
}
void ChkDsk(){
}
void Ver()
{
    gotoxy(0,22);
    system("ver");
    getch()
}
void End()
{
    textbackground(0);
    clrscr();
    exit (0);
}

```

# 屏幕特技设计

## 第 9 章

本章内容：

- 9.1 特技消屏
- 9.2 整屏弹出特技
- 9.3 连续颜色变换技术
- 9.4 屏幕的淡入淡出技术
- 9.5 屏幕图案设计



在本章中,我们介绍屏幕特技清屏、整屏弹出特技、连续颜色的变换技术、屏幕的淡入淡出技术以及屏幕图案设计。

## 9.1 特技清屏

在 Turbo C/Borland C 中,能够直接清屏的库函数有:在字符方式下,可用 `clrscr()`;在图形方式下,可用 `cleardevice()` 函数。在图形方式下,我们还可以采用画图的方式将屏幕信息清掉,与此有关的常用函数有:

```
void far bar(int x1,int y1,int x2,int y2);
```

确定一个以  $(x1,y1)$  为左上角,  $(x2,y2)$  为右下角的矩形窗口,并按规定的图模和颜色填充。颜色及填充方式的设定由函数 `setfillstyle()` 指定。

```
void far line(int x0,int y0,int x1,int y1);
```

画一条从点  $(x0,y0)$  到点  $(x1,y1)$  的直线,画线的颜色由 `setcolor()` 设定。

```
void far floodfill(int x,int y,int border);
```

对任意封闭图形填充。其中:  $x,y$  为封闭图形内的任意一点, `border` 为边界颜色。

清屏的方法有多种多样,完全取决于程序开发者的创造和想象。最常见的清屏方法有:

拉幕式(自左至右,自右而左,中间向两侧,两侧向中间)、栅格式、网格式、菱形式,或者是从一个角向另一个对角,等等。CLRTEC.C 列出了这些方式的实现方法,读者可以自己去领略体会,程序非常简单:

```
/*  
    CLRTEC.C——清屏特技  
*/  
#include "stdio.h"  
#include "conio.h"  
#include "dos.h"  
#include "graphics.h"  
void InitGra(void);  
void FillScr(void);  
void ClrScr1(void);  
void ClrScr2(void);  
void ClrScr3(void);  
void ClrScr4(void);  
void ClrScr5(void);  
void ClrScr6(void);  
void ClrScr7(void);  
void ClrScr8(void);  
int main(void)  
{
```

```

    InitGra();
    FillScr();
    ClrScr1();
    delay(100);
    FillScr();
    ClrScr2();
    delay(100);
    FillScr();
    ClrScr3();
    delay(100);
    FillScr();
    ClrScr4();
    delay(100);
    FillScr();
    ClrScr5();
    delay(100);
    FillScr();
    ClrScr6();
    delay(100);
    FillScr();
    ClrScr7();
    delay(100);
    FillScr();
    ClrScr8();
    closegraph();
    return(0);
}

void InitGra(void)
{
    int GraphMode, DraphDrive = DETECT;
    registerbgidriver(EGAVGA_driver);
    initgraph(&DraphDrive, &GraphMode, "");
}

void FillScr(void)
{
    setfillstyle(1, 1);
    bar(0, 0, 639, 479);
}

void ClrScr1(void)

```

```

{
    int i;
    setcolor(0);
    for (i = 0; i < 640; i++)
        line(i, 0, i, 479);
}

void ClrScr2(void)
{
    int i;
    for (i = 639; i >= 0; i--)
        line(i, 0, i, 479);
}

void ClrScr3(void)
{
    int i;
    for (i = 0; i < 321; i++)
    {
        line(i, 0, i, 479);
        line(640 - i, 0, 640 - i, 479);
    }
}

void ClrScr4(void)
{
    int i;
    for (i = 320; i >= 0; i--)
    {
        line(i, 0, i, 479);
        line(640 - i, 0, 640 - i, 479);
    }
}

void ClrScr5(void)
{
    int i, j;
    for (i = 0; i < 80; i++)
        for (j = 0; j < 8; j++)
            line(i + 80 * j, 0, i + 80 * j, 479);
}

void ClrScr6(void)

```

```

    int i,j;
    for (i = 0;i < 60;i++)
        for (j = 0;j < 8;j++)
            line(0,i+60*j,639,i+60*j);
}

void ClrScr7(void)
{
    int i,j;
    for (i = 0;i < 60;i++)
        for (j = 0;j < 8;j++)
        {
            line(i+80*j,0,i+80*j,479);
            line(0,i+60*j,639,i+60*j);
        }
}

void ClrScr8(void)
{
    int i,x,y,xy[8];
    x = 320;
    y = 240;
    setfillstyle(1,0);
    for (i = 0;i < 55;i++)
    {
        xy[0] = x;
        xy[1] = y-i*9;
        xy[2] = x+i*12;
        xy[3] = y;
        xy[4] = x;
        xy[5] = y+i*9;
        xy[6] = x-i*12;
        xy[7] = y;
        fillpoly(4,xy);
    }
}

```

## 9.2 整屏弹出特技

整屏弹出的技术可以通过屏幕的打开与关闭来实现,当从一个比较大的文件中读入数据

来生成或画一个比较复杂的图形时,由于计算机的速度不够快,给人的感觉图形是一部分一部分生成的,或是慢慢地画出的。若采用整屏弹出技术,在图形生成之前,首先关闭屏幕,屏幕上不显示任何内容,待图形生成完毕后,再打开屏幕,给人的感觉整个屏幕一下子弹出来了。

在 2.2 中已介绍,VGA 的操作定序器(输入/输出地址为 3C4H,3C5H)的时钟模式寄存器(索引号为 1)的第 5 位提供了开关显示器的功能:当它为 1 时,决定屏幕上不显示任何内容;当其为 0 时,则开始显示内容。因此,只要用 inportb()函数读出 3C5H 的字节值,采用“与”、“或”操作使其第 5 位为 0 或 1,然后再用 outportb()函数写入 3C5H,即可完成屏幕上的显示内容或不显示内容。下面的程序 SCR01.C,首先关闭屏幕,然后用 16 种不同的颜色画出了 16 个矩形块,接着打开屏幕,好像是一下子弹出来的。

/\*

SCR10.C——打开\关闭屏幕技术

\*/

#include "graphics.h"

#include "conio.h"

void CloseScr();

void OpenScr();

void InitGra();

void Quit();

void DrawBar();

int main()

{

InitGra();

CloseScr();

DrawBar();

getch();

OpenScr();

getch();

Quit();

return 0;

}

void CloseScr() /\* 关闭屏幕技术 \*/

{

unsigned char Byt;

outportb(0x3c4,1);

/\* 赋操作定序地址寄存器索引号, \*/

/\* 选择 1 号寄存器(时钟模式寄存器) \*/

Byt = inportb(0x3c5);

/\* 读出定序数据寄存器中的字节数据 \*/

Byt |= 0x20;

/\* 执行或操作,使该字节的第 5 位为 1,表示 \*/

/\* 使屏幕不显示任何内容 \*/

outportb(0x3c5,Byt);

/\* 将该字节数据送到定序数据寄存器 \*/

```

    return;
}

void OpenScr()                /* 打开屏幕技术 */
{
    unsigned char Byt;
    outportb(0x3c4,1);        /* 赋操作定序地址寄存器索引号, */
                                /* 选择 1 号寄存器(时钟模式寄存器) */
    Byt = inportb(0x3c5);      /* 读出定序数据寄存器中的字节数据 */
    Byt &= 0xdf;               /* 执行与操作,使该字节的第 5 位为 0,表示 */
                                /* 使屏幕显示内容 */
    outportb(0x3c5,Byt);      /* 将该字节数据送到定序数据寄存器 */
    return;
}

void InitGra(void)            /* 初始化屏幕为图形方式 */
{
    int GraphDrive = DETECT, GraphMode;
    registerbgidriver(EGAVGA_driver);
    initgraph(&GraphDrive,&GraphMode,"");
}

void Quit()
{
    closegraph();
    exit(0);
}

void DrawBar()                /* 画彩色矩形块 */
{
    int i;
    for (i = 0; i < 16; i++)
    {
        setfillstyle(1,i);
        bar(i * 40,0,(i+1) * 40,479);
    }
}

```

### 9.3 连续颜色变换技术

这种技术是对屏幕上的一种或几种颜色进行连续颜色变换,比如可以将红色连续变换成其它的任意 16 种颜色中的一种,造成一种霓虹灯闪烁的效果。在 Turbo C/Borland C 中,有一

个函数可以方便地设置调色板,来实现颜色的连续变换,其格式如下:

```
void setpalette(int colornum,int color);
```

其功能是将调色板的 colornum 颜色号变为 color 所代表的颜色值。因此,只要不断变换 color 的值,就可以得到连续颜色变换的效果。程序 SETPLT.C 采用 15 种颜色(黑色除外)画出了 30 个“灯”(实心圆),组成一个矩形,然后通过设置调色板,顺序变换各种颜色的值,视觉上好像每个灯都在移动。

```
/*
```

```
    SETPLT.C——连续颜色变换技术
```

```
*/
```

```
#include "graphics.h"
```

```
##include "conio.h"
```

```
void Palett();
```

```
void InitGra();
```

```
void Quit();
```

```
void DrawLight();
```

```
int main()
```

```
{
```

```
    InitGra();
```

```
    DrawLight();
```

```
    getch();
```

```
    Palett();
```

```
    Quit();
```

```
    return 0;
```

```
}
```

```
void Palett()
```

```
/* 设置调色板 */
```

```
{
```

```
    int i,j;
```

```
    for (;;) 
```

```
    {
```

```
        for (i = 1;i < 16;i++)
```

```
        {
```

```
            for (j = 1;j < 16;j++)
```

```
            {
```

```
                if (bioskey(1)) return;
```

```
                /* 有键按下时,退出 */
```

```
                setpalette(j,j+i);
```

```
                /* 设置调色板 */
```

```
                delay(10);
```

```
                /* 延时 10 毫秒 */
```

```
            }
```

```
        }
```

```
    }
```

```

    }
}

void InitGra()                                /* 初始化屏幕为图形方式 */
{
    int GraphDrive = DETECT, GraphMode;
    registerbgidriver(EGAVGA_driver);
    initgraph(&GraphDrive, &GraphMode, "");
}

void Quit()
{
    closegraph();
    exit(0);
}

void DrawLight()                              /* 画灯 */
{
    int i, r, x1, x2, y1, y2;
    r = 10;
    x1 = 120;
    y1 = 100;
    x2 = x1 + 36 * r;
    y2 = y1 + 24 * r;
    for (i = 1; i < 11; i++)
    {
        setfillstyle(1, i);                  /* 设置填充颜色 */
        setcolor(i);                          /* 设置画线颜色 */
        pieslice(x1, y1, 0, 360, r);          /* 画实心圆(灯) */
        setfillstyle(1, 16 - i);
        setcolor(16 - i);
        pieslice(x1, y2, 0, 360, r);
        x1 += 4 * r;
    }
    x1 = 120;
    y1 = 140;
    for (i = 11; i < 16; i++)
    {
        setfillstyle(1, 26 - i);
        setcolor(26 - i);
        pieslice(x1, y1, 0, 360, r);
        setfillstyle(1, i);
    }
}

```



```

        setcolor(i);
        pieslice(x2,y1,0,360,r);
        y1 += 4*r;
    }
}

```

## 9.4 屏幕的淡入淡出技术

将屏幕上的信息慢慢地暗下去,再慢慢地亮起来,通常称为淡入淡出技术。可以按照 2.3 中介绍的方法,将 256 个 DAC 寄存器一次一次减 1,一次一次再重新设置,直到 DAC 寄存器全部减为 0,就产生了慢慢消失的效果;尔后将其一次次加 1,一直加到和原来表相同,就实现了慢慢恢复显示的效果,程序 SLCG.C 是实现这些功能的源程序。

```

/*
   SLCG.C——屏幕的淡入、淡出技术
*/
#include "graphics.h"
#include "conio.h"
void DarkBrit();
void ChgColor(int,char,char,char);
void InitGra();
void Quit();
void DrawBar();
int main()
{
    InitGra();
    DrawBar();
    getch();
    DarkBrit();
    Quit();
    return 0;
}
void DarkBrit()
{
    int i,Flg = 1;
    char Red[256],Grn[256],Blu[256];
    char Red0[256],Grn0[256],Blu0[256];
    for (i = 0;i < 256;i++)
    {

```

```

    outportb(0x3c7,i);          /* 设置读调色表方式 */
    Red[i] = Red0[i] = inportb(0x3c9); /* 红色分量 */
    Grn[i] = Grn0[i] = inportb(0x3c9); /* 绿色分量 */
    Blu[i] = Blu0[i] = inportb(0x3c9); /* 蓝色分量 */
}
while (Flg)
{
    Flg = 0;
    for (i = 0; i < 256; i++)
    {
        if (Red[i])
        {
            Flg = 1;
            Red[i]--;
        }
        if (Grn[i])
        {
            Flg = 1;
            Grn[i]--;
        }
        if (Blu[i])
        {
            Flg = 1;
            Blu[i]--;
        }
        ChgColor(i,Red[i],Grn[i],Blu[i]); /* 修改颜色寄存器 */
    }
}
Flg = 1;
while (Flg)
{
    Flg = 0;
    for (i = 0; i < 256; i++)
    {
        if (Red[i] != Red0[i])
        {
            Flg = 1;
            Red[i]++;
        }
    }
}

```

```

        if (Grn[i] != Grn0[i])
        {
            Flg = 1;
            Grn[i]++;
        }
        if (Blu[i] != Blu0[i])
        {
            Flg = 1;
            Blu[i]++;
        }
        ChgColor(i,Red[i],Grn[i],Blu[i]);
    }
}
return;
}

void ChgColor(int No,char Red,char Grn,char Blu) /* 修改调色寄存器 */
{
    outportb(0x3c8,No); /* 颜色号 */
    outportb(0x3c9,Red); /* 红色分量 */
    outportb(0x3c9,Grn); /* 绿色分量 */
    outportb(0x3c9,Blu); /* 蓝色分量 */
}

void InitGra() /* 初始化屏幕为图形方式 */
{
    int GraphDrive = DETECT,GraphMode;
    registerbgidriver(EGAVGA_driver);
    initgraph(&GraphDrive,&GraphMode,"");
}

void Quit()
{
    closegraph();
    exit(0);
}

void DrawBar() /* 画出彩色矩形块 */
{
    int i;
    for (i = 0;i < 16;i++)
    {
        setfillstyle(1,i);
    }
}

```

```
bar(i * 40, 0, (i + 1) * 40, 479);
```

## 9.5 屏幕图案设计

在进行屏幕界面设计中,背景图案的设计可通过利用C语言库函数画图的方法直接画出,但这样做太麻烦。Turbo C/Borland C提供了一个非常方便实用的函数 `setfillstyle()`,它可以方便地以 12 种图模进行填充:

```
void far setfillstyle(int pattern, int color);
```

`color` 的值是当前屏幕图形模式时,颜色的有效值, `pattern` 的值及与其等价的符号常数如表 9-1。

表 9-1 关于填充式样 `pattern` 的规定

符号常数	数值	含 义
<code>EMPTY_FILL</code>	0	以背景颜色填充
<code>SOLID_FILL</code>	1	以实填充
<code>LINE_FILL</code>	2	以直线填充
<code>LTSLASH_FILL</code>	3	以斜线填充(阴影线)
<code>SLASH_FILL</code>	4	以粗斜线填充(粗阴影线)
<code>BKSLASH_FILL</code>	5	以粗反斜线填充(粗阴影线)
<code>LTBKSLASH_FILL</code>	6	以反斜线填充(阴影线)
<code>HATCH_FILL</code>	7	以直方网格填充
<code>XHATCH_FILL</code>	8	以斜网格填充
<code>INTTERLEAVE_FILL</code>	9	以间隔点填充
<code>WIDE_DOT_FILL</code>	10	以稀疏点填充
<code>CLOS_DOT_FILL</code>	11	以密集点填充
<code>USER_FILL</code>	12	以用户自定义的模式填充

关于图形填充模式的选择,请看下面的例子:

```
/*
```

FILL.C——封闭图形填充模式

```
*/
```

```
#include "stdio.h"
```

```
#include "alloc.h"
```

```
#include "graphics.h"
```

```
#include "conio.h"
```

```
#include "process.h"
```

```

void InitGra(void);
int main(void)
{
    int i;
    InitGra();
    for (i = 0; i < 12; i++)
    {
        setfillstyle(i,BLUE);
        bar(i * 50,100,(i+1) * 50,379);
    }
    getch();
    closegraph();
    exit (0);
}

void InitGra(void)
{
    int GraphMode,DrpghDrive = DETECT;
    registerbgidriver(EGAVGA _driver) ;
    initgraph(&DrpghDrive,&GraphMode,"") ;
}

```

另外,当用户需要定义其它的填充图案时,可以用函数 setfillpattern();

```
void setfillpattern(char * pattern,int color);
```

设置用户定义的填充图模和颜色以填充封闭图形。

其中:pattern 是一个指向 8 个字节的指针,这 8 个字节定义了 8×8 点阵的图形。每个字节的 8 位二进制数表示水平 8 点,8 个字节表示 8 行,然后以此为模型向整个封闭区域填充。

1							1	0X81
	1		1	1		1		0X5A
		1			1			0X24
	1		1	1		1		0X5A
	1		1	1		1		0X5A
		1			1			0X24
	1		1	1		1		0X5A
1							1	0X81

图 9-1

例如,我们要定义图 9-2 这样的图案,可以采用图 9-1 的填充图模。

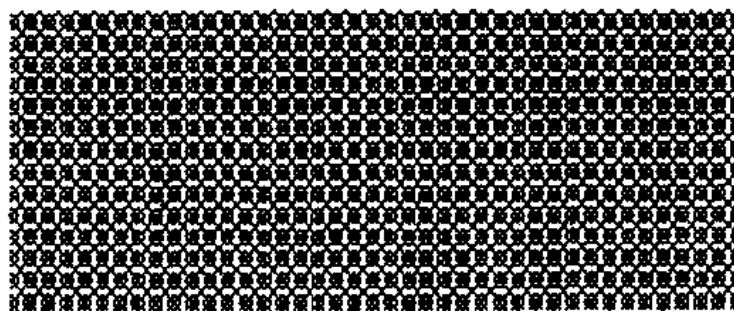


图 9-2

FILLPAT.C 显示了三种填充模式：

```
/*  
    FILLPAT.C——屏幕图案填充  
*/  
#include "stdio.h"  
#include "conio.h"  
#include "stdlib.h"  
#include "graphics.h"  
void InitGra(void);  
void FillScr(void);  
void FillPatn1(void);  
void FillPatn2(void);  
void FillPatn3(void);  
void Quit(void);  
int main(void)  
{  
    InitGra();  
    FillScr();  
    FillPatn1();  
    FillPatn2();  
    FillPatn3();  
    getch();  
    Quit();  
    return (0);  
}  
void InitGra(void)  
{  
    int GraphMode, DraphDrive = DETECT;
```

```

    registerbgidriver(EGAVGA _driver) ;
    initgraph(&DrphDrive,&GraphMode,"") ;
}
void FillScr(void)
{
    setcolor(14);
    setbkcolor(4);
    cleardevice();
}
void FillPatn1(void)
{
    char s1[] = {0x81,0x3c,0x42,0x5a,0x5a,0x42,0x3c,0x81};
    setfillpattern(s1,14);
    bar(100,30,420,140);
}
void FillPatn2(void)
{
    char s2[] = {0x81,0x5a,0x24,0x5a,0x5a,0x24,0x5a,0x81};
    setfillpattern(s2,14);
    bar(100,160,420,270);
}
void FillPatn3(void)
{
    char s3[] = {0x80,0x08,0x94,0x22,0x94,0x08,0x80,0x55};
    setfillpattern(s3,14);
    bar(100,300,420,420);
}
void Quit(void)
{
    closegraph();
    exit (0);
}

```

# 声音及音乐程序设计

## 第 **10** 章

本章内容：

10.1 发声函数及音乐程序设计

10.2 振铃的声响技术



随着计算机软件的商品化,界面越来越重要。界面除了屏幕上的图形之外,如果在软件系统中加入声音和音乐,会使您的软件更受欢迎。本章从两个方面介绍声音及音乐程序设计的方法。

## 10.1 发声函数及音乐程序设计

在 Turbo C/Borland C 中,最基本的发声方法是打印 ASCII 码为 7 的字符。即:

```
printf("%c\n",7);
```

或 `printf("\007");`

要开发音乐程序,可用支持音乐程序设计的函数 `sound()`,其格式为:

```
void sound(unsigned Frequency);
```

该函数以指定的频率打开 PC 机的扬声器,频率以赫兹为单位。若想关闭扬声器,可用函数 `nosound()`。

为了使用方便,我们可以编制一个乐谱文件,存放某首乐曲的音调频率和音长等信息,这样,每次只要运行音乐程序带一个乐谱文件的命令行参数即可。

我们可以先制定一个制作乐谱文件的规则:

最高音:在每个音符的前面加“e”;

高音:在每个音符的前面加“h”;

中音:在每个音符的前面加“m”;

低音:在每个音符的前面加“l”;

“e”,“h”,“m”,“l”,与其控制的音符构成音高,决定发音频率;音高的后面是音长,可用整数或小数输入,以控制延时时间,中间用空格分开。乐谱文件的最前端是一个整数,表示音长的基数,一般为:300,600,900,1200。乐谱文件的最末尾是“#”,表示乐谱文件的结束。

例如,下面的一首乐曲选段:

5 35 1— | 6 16 5—

则制成乐谱文件应为:

600 m5 l m3 0.5 m5 0.5 h1 2 m6 l h1 0.5 m6 0.5 m5 2 #

每个音的音长=音长基数×节拍数,其中,音长基数是乐谱文件的第一个字符,如上面的乐谱文件为 600,每个音的音频可用一模拟频率值输入。表 10-1 是中音 C 及其前后 4 个 8 度中各个音符的近似频率值。

表 10-1

低音	频率	中音	频率	高音	频率	最高音	频率
1	131	1	262	1	523	1	1047
2	147	2	296	2	587	2	1175
3	165	3	330	3	659	3	1319
4	176	4	349	4	699	4	1397
5	196	5	392	5	784	5	1568
6	220	6	440	6	880	6	1760
7	247	7	494	7	988	7	1976

源程序清单如下:

/\*

MUSIC.C 音乐程序

运行方法:

X>MUSIC <乐谱文件名>

\*/

#include "stdlib.h"

#include "stdio.h"

#include "dos.h"

#define TRUE 1

#define FALSE 0

```
int Freq[4][8] = {0,131,147,165,176,196,220,247,
                  0,262,296,330,349,392,440,494,
                  0,523,587,659,699,784,880,988,
                  0,1047,1174,1319,1397,1568,1760,1976}; /* 频率值 */
```

int main(int argc, char \*argv[])

{

FILE \*fp;

int Rate, Sd, i = 0, j;

char SdHg, Flg = TRUE;

float SdLg, Str[100][2];

if (argc != 2)

{

printf("命令行参数错误!"), exit(1);

}

fp = fopen(argv[1], "r");

if (fp == NULL)

{

printf("乐谱文件打不开!"), exit(1);

}

fscanf(fp, "%d", &Rate);

while (!feof(fp) && Flg)

{

fscanf(fp, "%c%d %f", &SdHg, &Sd, &SdLg); /\* 读乐谱文件 \*/

Str[i][1] = Rate \* SdLg;

switch(SdHg)

{

case 'e': Str[i++][0] = Freq[3][Sd]; /\* 最高音 \*/

break;

```

        case 'h':Str[i++][0] = Freq[2][Sd];          /* 高音 */
            break;
        case 'm':Str[i++][0] = Freq[1][Sd];          /* 中音 */
            break;
        case 'l':Str[i++][0] = Freq[0][Sd];          /* 低音 */
            break;
        case '#':Flg = FALSE;                        /* 结束 */
    }
}
for (j = 0;j < i;j++)
{
    sound(Str[j][0]);                                /* 音高 */
    delay(Str[j][1]);                                /* 音长 */
}
nosound();                                           /* 关闭扬声器 */
return 0;
}

```

## 10.2 振铃的声响技术

上面介绍的是音乐程序的设计,有时可能还需要其它类型的音响技术。例如,在适当的时候向用户发出电话振铃声,等待用户按键响应等,消毒软件 CPAV 就很好地应用了振铃向用户报警。

RING.C 提供的 Ring()和 Bell()可直接供为读者采用。RING()函数中,Str 为一字符串,用于选择各种振铃标准:

Str = "10000"为中国交换机的振铃标准,通断比为 1:4,即振铃 1 秒,等 4 秒。

RING()函数将每个 1/0 秒称为 1 拍,其中 0 拍不发音,为了实现电话铃的颤音,1 拍的音分为若干个小周期。

```

/*
RING.C——振铃程序
*/
#include "stdio.h"
#include "conio.h"
#include "string.h"
#include "dos.h"
#define TRUE 1
char Ring(char *Str);
void Bell(char *Str);

```

```

int main(void)
{
    Ring("110111000");
    Bell("053535350");
    getch();
    Bell("076543210");
    return 0;
}

char Ring(char *Str)
{
    int Frq,i,On = TRUE,Ptr = 0,mx = 4,Stp = 6;
    int Cunt = 0,Unt = 8,Dlt = 4,Bas = 600;
    char Flg[40] = "110000";
    strcpy(Flg,Str);
    for (;)
    {
        if (Cunt == Unt)
        {
            Cunt = 0;
            Ptr ++;
            if (! Flg[Ptr]) Ptr = 0;
            On = (Flg[Ptr] == '0' ? 0 : 1);
            if (! On) nosound();
        }
        Frq = Bas;
        for (i = 0;i < mx;i++)
        {
            Frq += Stp;
            if (On) sound(Frq);
            delay(Dlt);
        }
        for (i = 0;i < mx;i++)
        {
            Frq -= Stp;
            if (On) sound(Frq);
            delay(Dlt);
        }
        if (kbhit())
        {

```

```

        nosound();
        return (getch());
    }
    Cunt++;
}
}
void Bell(char * Str)
{
    int BasTon[] = {000,262,296,330,349,392,440,494};
    int Dlt = 100;
    char * Ptr;
    for (Ptr = Str; *Ptr; Ptr++)
    {
        sound(BasTon[ *Ptr - '0' ]);
        delay(Dlt);
        nosound();
    }
}
}

```

# 动画程序设计

第 **11** 章

在应用软件中,加入适当的动画,更能体现出软件的水平。动画显示实际上是一系列静止图象在不同位置的重现。其基本方法是在屏幕上先擦除一个静止图象,然后在相邻的适当位置绘出下一幅图象,程序重复地执行擦除和绘制过程,就可以产生所需要的动画效果。获得效果满意的动态图形的关键,在于以较快的速度进行图形的清除与显示,使人的视觉不易觉察到图形擦绘的过程。

动画涉及到背景画面和运动子画面。背景画面往往覆盖整个显示屏幕,并且一般保持静止。背景画面的设计比较复杂,是子画面的工作环境。背景画面的图形往往取自一些图形数据文件,这些图形数据文件是事先用图形编辑软件编辑好,之后保存于磁盘上以便需要时及时调入。子画面一般是动态设计的,也可以由子画面图形数据文件调入。

动态子画面的保持动态一般可通过下面的方法来实现:

- (1) 异或(XOR)运算,动态子画面在同一显示位置上连续进行两次 XOR 运算,使画面出现和消失。然后在下一显示位置作同样的操作,这样反复进行,便形成动画效果;
- (2) 通过 setactivepage()和 setvisualpage()函数设置不同显示页。先用 setactivepage()函数在不同页面上画出一幅幅图象,再用 setvisualpage()函数交替显示,便可以实现动画。

下面的例子 SAUCER.C 是摘自 Turbo C/Borland C 2.0 系统盘中的 BGIDEM0.C,并经作者修改,更容易阅读和理解,它演示了一个飞碟的飞行情况,程序是通过采用第一种方法实现的,采用 getimage()函数保存飞碟图象,然后在其它的随机位置用 putimage()函数释放,从而形成动画的效果。

随机数是由函数 random()产生的,由一个位置飞到另一个位置,可能往前飞,也有可能往后飞,往前或往后的确定是通过以下的判断来实现的:

```
if ((step / 2) % 2) step = -1 * step;
```

源程序如下:

```
/*  
SAUCER.C——飞碟的飞行  
*/  
#include "dos.h"  
#include "math.h"  
#include "conio.h"  
#include "stdio.h"  
#include "stdlib.h"  
#include "stdarg.h"  
#include "graphics.h"  
void InitGra(void);  
void SaucFly(void);  
void DrSaucer(int,int,int);  
void DrStar();  
int main()  
{
```

```

InitGra();          /* 设置系统图形模式 */
SaucFly();
closegraph();       /* 返回系统文本模式 */
return(0);
}

```

```
void SaucFly(void)
```

```

{
    int x,y,x1,y1,x2,y2,i,step,x0,y0,r;
    void * Buf;
    r = 20;          /* 飞碟的大小 */
    x0 = 320;y0 = 200;
    DrStar();
    DrSaucer(x0,y0,r);
    x1 = x0-r-1;
    y1 = y0-14;
    x2 = x0+r+1;
    y2 = y0+r/3+3;
    Buf = malloc(imagesize(x1,y1,x2,y2));
    getimage(x1,y1,x2,y2,Buf);
    putimage(x1,y1,Buf,XOR_PUT);    /* 擦除图象 */
    x = 320;
    y = 240;
    while (! kbhit())
    {
        putimage(x,y,Buf,XOR_PUT);    /* 画出图象 */
        delay(80);
        putimage(x,y,Buf,XOR_PUT);    /* 擦除图象 */
        step = random(2*r);            /* 飞碟飞行 */
        if ((step/2) % 2 != 0) step = -1 * step;
        x += step;
        step = random(r);              /* 随机产生下一个位置 */
        if ((step/2) % 2 != 0) step = -1 * step;
        y += step;
        if (y < 0) y = 0;
        x += step;
        if (x < 0) x = 0;
    }
    free(Buf);
}

```



```

}
void InitGra()
{
    int GraphMode, DraphDrive = DETECT;
    registerbgidriver(EGAVGA_driver);
    initgraph(&DraphDrive, &GraphMode, "");
}
void DrStar() /* 画一些“星星” */
{
    int i;
    setbkcolor(1);
    cleardevice();
    for (i=0; i<1000; ++i)
        putpixel(random(639), random(479), random(15)+1);
}
void DrSaucer(int x0, int y0, int r) /* 画飞碟 */
{
    setfillstyle(1, 15);
    fillellipse(x0, y0, r, (r/3)+2);
    ellipse(x0, y0-4, 190, 357, r, r/3);
    line(x0+7, y0-6, x0+10, y0-12);
    circle(x0+10, y0-12, 2);
    line(x0-7, y0-6, x0-10, y0-12);
    circle(x0-10, y0-12, 2);
}

```

第二种方法是设置不同的显示页实现动画。图形的输出活动页函数 `setactivepage()` 和设置可见图形页号函数 `setvisualpage()` 的调用格式为：

```

void far setactivepage(int page);
void far setvisualpage(int page);

```

图形页实际上是一个虚拟页面，是内存中开辟的一个图形缓冲区。活动图形页可以是当前显示页面，也可以是非显示页面，当用函数 `setactivepage()` 选定某一页为活动图形页后，其后所有的图形输出都针对这一页。有了多个图形页，程序就可以将图形输出到一个非显示屏幕上，然后通过调用 `setvisualpage()` 函数改变可见页来快速显示关闭图形页面中的画面。多个图形页交替显示的过程如下：

在所用的两个页面中，当一个可见页面用于显示时，另一个关闭页用于绘图，当新的画面生成后，就把两页进行转换，原作为显示用的页面再用来绘制新的图形。一般可将画面顺序作如下安排：第一页用于显示动画过程的第 1, 3, 5, … 等幅画面，第二页用于显示动画过程的第 2, 4, 6, … 等幅画面，如此交替显示下去，即实现了平滑的动画效果。程序 `MVBL.C` 实现了一个小球的移动：

```

/*
    MVBL.C——小球的移动
*/
#include "graphics.h"
#include "conio.h"
#include "stdio.h"
#include "stdlib.h"
int x0,x1,y;
int radiu = 20;                /* 小球的半径 */
void InitGra(void);
void Quit();
void DrawPage1();
void DrawPage0();
void ClsPage1();
void ClsPage0();
int main()
{
    int i = 0;
    int x = 0;
    int step = 2;
    x = 120;
    y = 240;
    x0 = x1 = x;
    InitGra();
    DrawPage1();
    for (i = 0; i < 200; i++)
    {
        setvisualpage(1);
        x0 += step;
        ClsPage0();
        Setvisualpage(0);
        ClsPage1();
        x1 = x0+step;
        DrawPage1();
    }
    Quit();
    return 0;
}

void InitGra(void)                /* 初始化屏幕为图形方式 */

```

```

{
    int GraphDrive = DETECT, GraphMode;
    registerbgidriver(EGAVGA _driver);
    initgraph(&GraphDrive, &GraphMode, "");
}

void Quit()
{
    closegraph();
    exit (0);
}

void DrawPage1() /* 在第一页画 */
{
    setactivepage(1);
    setfillstyle(1, BLUE);
    fillellipse(x1, y, radiu, radiu);
    return;
}

void DrawPage0() /* 在第 0 页画 */
{
    setactivepage(0);
    setfillstyle(1, BLUE);
    fillellipse(x0, y, radiu, radiu);
    return;
}

void ClsPage1() /* 清除第 1 页 */
{
    setactivepage(1);
    setcolor(0);
    setfillstyle(EMPTY _FILL, 0);
    fillellipse(x1, y, radiu, radiu);
    ellipse(x1, y, 0, 360, radiu, radiu);
    return;
}

void ClsPage0() /* 清除第一页 */
{
    setactivepage(0);
    setcolor(0);
    setfillstyle(EMPTY _FILL, 0);
    fillellipse(x0, y, radiu, radiu);

```

```

        ellipse(x1,y,0,360,radu,radu);
    return;
}

```

作为动画程序的另外一个例子,CLOCK.C 演示的是一个以钟表的形式实时显示计算机系统时间的程序,其实现的方法是这样的:

首先通过 DOS 中断功能调用 2CH 读出系统时间,根据这个时间值,计算出时针,分针和秒针的对应位置,并画出:时针是三点线宽,黄色;分针和秒针都是一点线宽,分针为黄色,秒针为红色。然后是循环,每经过 100(ms),用异或方式(XOR)擦除原来的秒针,再在下一个位置画出,如果循环中有按键,则退出;否则循环 300 次后,重复执行上述过程,时针,分针和秒针均作了相应的走动。

```
/*
```

CLOCK.C——以钟表的形式实时显示系统时间

```
*/
```

```
#include "stdlib.h"
```

```
#include "conio.h"
```

```
#include "stdio.h"
```

```
#include "graphics.h"
```

```
#include "math.h"
```

```
#include "dos.h"
```

```
#define BB 3.14159/180
```

```
void InitGra(void);
```

```
void SysTime(int x,int y,int r);
```

```
int Clock(int x,int y,int r);
```

```
void ClockPict(int x,int y,int r);
```

```
int main(void)
```

```
{
```

```
    int x,y,r;
```

```
/* 定义表盘的中心及半径 */
```

```
    x = 400;
```

```
    y = 240;
```

```
    r = 42;
```

```
    InitGra();
```

```
    SysTime(x,y,r);
```

```
    closegraph();
```

```
    return(0);
```

```
}
```

```
void InitGra(void)
```

```
{
```

```
    int GraphDrive = DETECT,GraphMode;
```

```
    registerbgidriver(EGAVGA_driver);
```

```

    initgraph(&GraphDrive,&GraphMode,"");
}
void SysTime(int x,int y,int r) /* 表盘中心坐标,半径 */
{
    ClockPict(x,y,r);
    while ((Clock(x,y,r) != 1) && (! kbhit())); /* 钟表运行直到按键为止 */
}
int Clock(int x,int y,int r) /* 表盘中心,表盘半径 */
{
    float hr,mt,sd,dh,dm,ds,ds0;
    int i,Fst = 1;
    union REGS in,out;
    setcolor(0);setfillstyle(1,0);
    pieslice(x,y,0,360,r-11);
    in.h.ah = 0x2c;
    int86(0x21,&in,&out);
    hr = out.h.ch; /* 时 */
    mt = out.h.cl; /* 分 */
    sd = out.h.dh; /* 秒 */
    if (hr > 12) hr = hr-12;
    hr = hr+mt/60;
    dh = 270+30*hr;if (dh > 360) dh = dh-360;dh = dh*BB;
    dm = 270+6*mt;if (dm > 360) dm = dm-360;dm = dm*BB;
    ds = 270+6*sd;if (ds > 360) ds = ds-360;ds = ds*BB;
    setcolor(15);setlinestyle(0,0,3);
    line(x,y,x+(r-20)*cos(dh),y+(r-20)*sin(dh)); /* 画时针 */
    setlinestyle(0,0,1);
    line(x,y,x+(r-15)*cos(dm),y+(r-15)*sin(dm)); /* 画分针 */
    setwriteMode(XOR_PUT);
    for (i = 0;i < 300;i++)
    {
        in.h.ah = 0x2c; /* 循环内执行秒针的走动 */
        int86(0x21,&in,&out);
        sd = out.h.dh;
        ds = 270+6*sd;
        if (ds > 360) ds = ds-360;
        ds = ds*BB;
        if (Fst) ds0 = ds;
        setlinestyle(0,0,1);setcolor(12);
    }
}

```

```

    if (! Fst) line(x,y,x+(r-12)*cos(ds0),y+(r-12)*sin(ds0));
                                /* 擦去原秒针 */
    line(x,y,x+(r-12)*cos(ds),y+(r-12)*sin(ds)); /* 重画秒针 */
    ds0 = ds;Fst = 0;
    if (kbhit()) return (1);    /* 如果有按键,返回 */
    else delay(100);
}
}

void ClockPict(int x,int y,int r) /* 画表盘 */
{
    float af;
    int i,Dlt;
    setwritemode(COPY__PUT);setlinestyle(0,0,1);
    setcolor(0);setfillstyle(1,0);
    pieslice(x,y,0,360,r+2);
    setcolor(14);
    circle(x,y,r);
    line(x+r+5,y-2,x+r+10,y-2);
    line(x+r+5,y+2,x+r+10,y+2);
    setlinestyle(0,0,3);
    rectangle(x+r+5,y-6,x+r+10,y+6);
    circle(x,y,r+5);
    for (i = 0;i < 360;i += 30) /* 画表的时刻刻度 */
    {
        af = i * BB;
        if (i==0||i==90||i==180||i==270) Dlt = 8; /* 3,6,9,12点刻度稍长 */
        else Dlt = 5;
        line(x+(r-Dlt)*cos(af),y+(r-Dlt)*sin(af),x+r*cos(af),y+r*sin(af));
    }
}
}

```

## 附录一

## 书中程序出现章节对照表

章节	程序名	程序说明	页码
1.4	RDSYSDT1.C	用 DOS 功能调用显示系统时间	6
1.4	RDSYSDT2.C	用 BIOS 功能调用显示系统时间	6
2.3	COLOR16T.C	显示文本方式下 VGA 设置的 16 种颜色	18
2.3	COLOR16G.C	显示图形方式下 VGA 设置的 16 种颜色	20
2.4	CHG16.C	改变 VGA 设置的 16 种颜色	22
2.4	GETCOLOR.C	读取某颜色号对应的红、绿、蓝三色分量值	25
2.5	HAFTON.C	半色调明暗处理技术程序例	26
2.6	VGA256.C	显示 VGA 设置的 256 种颜色(256 色编程技术)	29
2.7	CHG256.C	改变 VGA 设置的 256 种颜色	31
2.8	3DBX.C	立体块程序	33
2.8	B3DBX.C	改变 VGA 设置的 16 种颜色,设计立体块	37
3.1	FUNCC.C	利用 C 语言库函数显示彩色汉字	40
3.1	BIOSCC.C	利用 BIOS 显示彩色汉字	41
3.1	GFUNCC.C	图形方式下利用 C 语言库函数显示汉字	43
3.2	PUTCC16.C	西文 DOS 下显示 16 点阵汉字	46
3.2	PUTCC24.C	西文 DOS 下显示 24 点阵汉字	49
3.2	PUTCCEX.C	西文 DOS 下显示 24 点阵放大汉字	52
3.2	RCCDSP.C	24 点阵汉字的无级放大、旋转与倾斜显示	55
3.2	CR16.C	建立 16 点阵小字库	58
3.2	RDC16.C	读取已建立的 16 点阵小字库并显示汉字	60
3.2	CR24.C	建立 24 点阵小字库	61
3.2	RD24.C	读已建立的 24 点阵小字库显示汉字	63
3.2	PUTSLP.C	华光矢量汉字的显示	66
3.2	RSLPDSP.C	矢量汉字的无级放大、旋转与倾斜显示	70
3.2	CRSLP.C	矢量小字库的建立	74
3.2	RDSLP.C	读已建立的矢量小字库并显示汉字	76
3.3	CC256.C	256 色汉字显示程序	79
4.3	POPWIN.C	弹出式窗口程序	88
4.3	HWIN.C	改进的弹出窗口程序	93
5.2	MSCRS.C	鼠标器编程例	107
6.1	MENUE.C	西文 DOS 下的西文下拉式菜单	117
6.1	SAVRST.C	屏幕图形保存、恢复程序例	125
6.1	MENUC.C	西文 DOS 下的汉字下拉式菜单	128
6.1	MENUP.C	西文 DOS 下的汉字弹出式下拉菜单	136
6.2	MENU0.C	中文 DOS 下的汉字下拉式菜单	144
6.3	MSMENU.C	用鼠标器选择立体按钮菜单	151
6.4	MSKYMU.C	鼠标与键盘兼容的下拉式菜单	158
6.5	MENUC	交替运行其它语言的汉字菜单	168

章节	程序名	程序说明	页码
7.3	GETD.C	可编辑的数据输入程序	189
7.4	GGETD.C	图形方式下的数据输入	194
7.5	OUTXT1.C	图形方式下文本字型和字体的设置(一)	198
7.5	OUTXT2.C	图形方式下文本字型和字体的设置(二)	199
7.5	SYSDAT.C	图形方式系统日期的日历牌式显示	201
7.6	FMTOUT.C	图形方式格式化数据输出	204
7.7	ROUTXT.C	下西文字符的放大旋转与倾斜显示	209
8.1	CRTICON.C	创建图符程序	213
8.2	ICONMU.C	图符菜单程序	219
9.1	CLRTEC.C	特技清屏程序	225
9.2	SCR10.C	整屏弹出特技程序	229
9.3	SETPLT.C	连续颜色变换的程序	231
9.4	SLCG.C	屏幕淡入淡出的程序	233
9.5	FILL.C	屏幕填充模式显示	236
9.5	FILLPAT.C	屏幕填充图案程序	238
10	MUSIC.C	音乐程序	242
10	RING.C	振铃程序	243
11	SAUCER.C	动画程序(飞碟飞行)	247
11	MVBL.C	动画程序(小球的移动)	250
11	CLOCK.C	实时时钟程序(以钟表的形式显示系统时间)	252

## 附录二

### 本书常用的 ASCII 码表

ASCII 码值		代表字符	ASCII 码值		代表字符
十进制	十六进制		十进制	十六进制	
07	07	响铃	187	BB	⌋
24	18	↑	188	BC	⌌
25	19	↓	191	BF	⌍
26	1A	→	192	C0	⌎
27	1B	←	196	C4	—
176	B0		200	C8	⌐
177	B1		201	C9	⌑
178	B2		205	CD	=
179	B3		217	D9	┘
186	BA		218	DA	┐





(1) 孙永毅等. 微型计算机 IBM PC 的原理与应用(续二)——图形显示器及其程序设计, 南京: 南京大学出版社, 1991 年

(2) 孙欣等. 面向对象程序设计 Turbo C++ 库函数参考, 北京: 海洋出版社, 1992 年

(3) 戴永贞. DOS/BIOS 系统功能调用及程序例, 北京: 海洋出版社, 1993 年

(4) 曹琦. 人机工程, 成都: 西南交通大学出版社, 1992 年

(5) 江秀汉. C 语言实用程序荟萃, 西安: 西安电子科技大学出版社, 1992 年

(6) 王军政. Turbo C 2.0 实用高级编程技巧, 北京: 科海培训中心, 1992 年

(7) 谭浩强. C 语言程序设计教程, 北京: 高教出版社, 1992 年

(8) 姒荷. 用户界面程序设计, 北京: 希望公司, 1992 年

(9) 《微计算机应用》杂志, 1991—1994 年

(10) 《计算机世界月刊》杂志, 1991—1994 年

(11) 《中国计算机用户》杂志, 1991—1994 年

(12) 《微型机及应用》, 1991—1994 年

(13) 《计算机应用研究》, 1991—1994 年

(14) 《微型计算机》, 1991—1994 年