14

第 14 章

C 语言高级编程技术

使用过 Windows 系统的用户都感受到了图形用户界面的直观和高效。所有 Windows 系统的应用程序都拥有相同或相似的基本外观，包括窗口、菜单、工具条、状态栏等。用户只要掌握其中一个，就不难学会其他软件，从而降低了学习成本和难度。而且 Windows 是一个多任务的操作环境，它允许用户同时运行多个应用程序，或在一个程序中同时做几件事情。例如，我们可以边欣赏 MP3 的音乐边 IE 冲浪，可以在运行 Word 时同时编辑多个文档等。用户直接通过鼠标或键盘来使用应用程序，或在不同的应用程序之间进行切换，非常方便。这些都是单任务、命令行界面的 DOS 操作系统所无法比拟的。TC2.0 或 TC3.0 均是在 DOS 环境下运行的 C 系统。不过，无论采用 TC，还是 VC、BC，所产生的 C 可执行程序都是基于 DOS 系统的。

C 语言发展如此迅速，而且成为最受欢迎的语言之一，主要因为它具有强大的功能。C 是一种“中”级语言，它把高级语言的基本结构和语句与低级语言的实用性结合起来。C 语言可以对位、字节和地址进行操作，而这三者是计算机最基本的工作单元。C 语言具有各种各样的数据类型，并引入了指针概念，可使程序效率更高。另外 C 语言也具有强大的图形功能，支持多种显示器和驱动器。而且计算功能、逻辑判断功能也比较强大，可以实现决策目的。C 系统提供了大量的功能各异的标准库函数，减轻了编程的负担。所以要用 C 语言实现具有类 Windows 系统应用程序界面特征的、或更生动复杂的 DOS 系统的程序，就必须掌握更高级的编程技术。

本章介绍了如何利用 TC 系统所提供的相关函数实现菜单设计、图形绘制、动画的播放、乐曲的演奏、汉字的显示、图片的显现等技术。

14.1 文本的屏幕输出和键盘输入

在前面几章的实例中，程序运行的结果都显示在黑色屏幕上，颜色单调。这并不是说 C 语言的所有结果都只能显示在黑色屏幕上，与其他的高级语言一样，C 语言也可以显示多样的界面。例如，编写一个程序，将屏幕垂直平分成两个窗口，左边窗口为蓝色背景，白色前景，右边窗口为绿色背景，红色前景。两个窗口都设计为文本输入，即在窗口中可以输入文字，在窗口屏幕中显示出来。使用【Tab】键在左右两个窗口中切换，每个窗口都有光标，活动窗口光标进行闪烁。

这时前面不曾接触到的新概念，如文本窗口、前景色、背景色，以及如何在屏幕中显示文本输入的窗口；如何设置窗口的前景色、背景色或闪烁等显示属性；如何通过按键来控制窗口的切换等。这就涉及了有关文本的屏幕输出和键盘的输入知识。下面就来介绍这两个方面的内容。

14.1.1 文本屏幕输出

显示器的屏幕显示方式有两种：文本方式和图形方式。本节将介绍文本方式，而图形方式将会在后面一节介绍。文本方式就是显示文本的模式，它的显示单位是字符而不是图形方式下的像素，因而在屏幕上显示字符的位置坐标就用行和列表示。Turbo C 的字符屏幕函数主要包括文本窗口大小的设定、窗口颜色的设置、窗口文本的清除和输入输出等函数。这些函数的有关信息（如宏定义等）均包含在 conio.h 头文件中，因此在用户程序中使用这些函数时，必须用 include 将 conio.h 包含进程序。

1. 文本窗口定义

Turbo C 默认定义的文本窗口为整个屏幕，共有 80 列 25 行的文本单元。规定整个屏幕的左上角为 1 行 1 列，右下角坐标为 25 行 80 列，并规定沿水平方向为 X 轴，方向朝右；沿垂直方向为 Y 轴，方向朝下。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。除了这种默认的 80 列 25 行的文本显示方式外，Turbo C 还支持另外 4 种文本显示方式，可以用文本显示方式设置函数 textmode 来进行设置，该函数调用的形式为：

```
textmode(newmode)
```

其中，newmode 参数可以选用如表 14-1 所示的任一种方式，可以用表中指出的方式代号，也可以用对应的符号常量。该函数将清除屏幕，以整个屏幕为当前窗口，并移动到屏幕的左上角。该函数无返回值。

表 14-1 文本显示方式

方　式	符号常量	显示列×行数和颜色
0	BW40	40×25 黑白显示
1	C40	40×25 彩色显示
2	BW80	80×25 黑白显示

续表

方　式	符号常量	显示列×行数和颜色
3	C80	80×25 彩色显示
7	MONO	80×25 单色显示
-1	LASTMODE	上一次的显示方式

表中的 LASTMODE 方式指上一次设置的文本显示方式，它常用于在图形方式到文本方式的切换。关于颜色，将在文本颜色设置函数中介绍，MONO 方式用在 MGA 显示器上。

Turbo C 还可以通过窗口设置函数 window 让用户根据自己的需要重新设定显示窗口。window 函数的调用形式为：

```
window(left,top,right,bottom);
```

该函数的作用就是用来定义屏幕上的一个矩形域作为窗口，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。其参数 left（列）和 top（行）表示是窗口左上角的坐标；right（列）和 bottom（行）表示窗口的右下角坐标，它们都是整型数据；坐标（left，top）和（right，bottom）是相对于整个屏幕而言的。

例如，要定义一个窗口左上角在屏幕（10，8）处，大小为 10 列 20 行的窗口可写成：

```
window(10, 8, 20, 28);
```

编者手记

请读者注意，这里坐标的表示与在平时习惯上的坐标表示有所不同。习惯的表示是在圆括号中将行数放在前面列数在后面,但这里的表示是将列数放在前面行数放在后面，例如上面的坐标（10，8）在这里表示是 10 列 8 行，而不是 10 行 8 列，请读者一定要注意这一点。

注意，若 window 函数中的坐标超过了屏幕坐标的界限，则窗口的定义就失去了意义，也就是说定义将不起作用，但程序编译链接时并不出错。另外，一个屏幕可以定义多个窗口，但现行窗口只能有一个（因为 DOS 为单任务操作系统），当需要用另一窗口时，可将定义该窗口的 window 函数再调用一次，此时该窗口便成为现行窗口了。

2．文本窗口颜色和其他属性的设置

Turbo C 为了控制文本显示的前景色（字符颜色）和背景色，提供了控制颜色的函数，常用的有下面几个函数。

（1）字符颜色设置函数 textcolor()

该函数调用形式为：

```
textcolor(color);
```

该函数的作用就是设置显示的前景色，即字符显示的颜色。其参数 color 可用表 14-2 所列的表中的符号常数或数值表示。

表 14-2　颜色表

符号常数	数　值	显 示 色	说　明
BLACK	0	黑	前景、背景色
BLUE	1	蓝	前景、背景色
GREEN	2	绿	前景、背景色
CYAN	3	青	前景、背景色
RED	4	红	前景、背景色
MAGENTA	5	洋红	前景、背景色
BROWN	6	棕	前景、背景色
LIGHTGRAY	7	淡灰	前景、背景色
DARKGRAY	8	深灰	用于前景色
LIGHTBLUE	9	淡蓝	用于前景色
LIGHTGREEN	10	淡绿	用于前景色
LIGHTCYAN	11	淡青	用于前景色
LIGHTRED	12	淡红	用于前景色
LIGHTMAGENTA	13	淡洋红	用于前景色
YELLOW	14	黄	用于前景色
WHITE	15	白	用于前景色
BLINK	128	闪烁	用于前景色

表中的符号常数与相应的数值等价，二者可以互换。例如设定蓝色前景色可以使用 textcolor(1)，也可以使用 textcolor (BLUE)，两者没有任何区别，只不过后者比较容易记忆，一看就知道是蓝色。

（2）文本背景颜色设置函数 textbackground()

该函数的调用形式为：

```
textbackground(color);
```

该函数的作用就是用来设置文本显示的背景颜色，其参数 color 仅能选择表 14-2 中的前 8 和颜色，即值为 0～7。

（3）文本属性设置函数 textattr()

该函数的调用形式为：

```
textattr(attr);
```

该函数的作用就是设置文本显示的属性，所谓显示的属性指字符显示的颜色和背景色，字符显示是否闪烁。也就是说，通过 textattr 函数可以同时设置文本的字符和背景颜色。其参数 attr 的值表示颜色形式编码的信息，即参数 attr 可用 1 字节（8 位）来描述，其各位含义如图 14-1 所示。

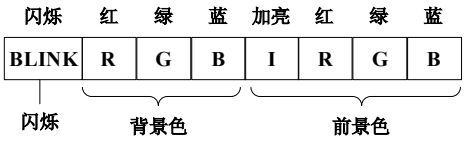

图 14-1　属性字节各位含义

其中低 4 位用来设置字符显示颜色（对应颜色值 0～15），4～6 位用来设置显示背景色（颜色为 0～7），第 7 位最高位用来设置显示的字是否闪烁，例如：

要求在蓝色背景下显示黄色的字符，可用此函数设置：

```
textatter(YELLOW+(BLUE<<4));
```

要求在白色背景下显示闪烁的蓝色字符，可设置为：

```
textatter((WHITE<<4)+BLUE+BLINK);
```

或者：

```
textatter((15<<4)+1+128);
```

其中 WHITE<<4 表示左移 4 位，即其对应的二进制值左移 4 位，变成 4～6 位，这恰是设置背景色的位，也就是用 15 的二进制表示的数左移 4 位。

注意

（1）对于背景只有 0 到 7 共 8 种颜色，取大于 7 小于 15 的数，则代表的颜色与减 7 后的值对应的颜色相同；

（2）用 textbackground()和 textcolor()函数设置了窗口的背景与字符颜色后，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颜色。

（3）用 textattr()函数时背景颜色应左移 4 位，才能使 3 位背景颜色移到正确位置。

3．字符显示亮度控制函数

字符显示亮度控制函数有：highvideo()函数、lowvideo()函数和 normvideo()函数。它们的调用方式和作用如下：

```
highvideo();
```

该函数将设置用高亮度显示字符。

```
lowvideo();
```

该函数将设置用低亮度显示字符。

```
normvideo();
```

该函数将设置通常亮度显示字符。

4．控制台文本的输入输出函数

（1）窗口内文本的输出函数

前面介绍过的 printf()，putc()，puts()，putchar()和输出函数以整个屏幕为窗口，它们不受由 window 设置的窗口限制，也无法用函数控制它们输出的位置，但 Turbo C 提供了三个文本

输出函数，它们受窗口的控制，窗口内显示光标的位置，就是它开始输出的位置。当输出行右边超过窗口右边界时，自动移到窗口内的下一行开始输出，当输出到窗口底部边界时，窗口内的内容将自动产生上卷，直到完全输出完为止，这三个函数均受当前光标的控制，每输出一个字符光标后移一个字符位置。这三个输出函数原型为：

```
cprintf(格式化字符串，表达式表);
cputs(字符串);
putch(字符);
```

它们的使用格式同 printf()，puts()和 putc()，其中 cprintf()是将按格式化串定义的字符串或数据输出到定义的窗口中，其输出格式串同 printf 函数，不过它的输出受当前光标控制，且输出特点如上所述，cputs 同 puts，是在定义的窗口中输出一个字符串，而 putch()则是输出一个字符到窗口，它实际上是函数 putc 的一个宏定义，即将输出定向到屏幕。

（2）窗口内文本的输入函数

可直接使用 stdio.h 中的 getch 或 getche 函数。

编者手记

需要说明的是，getche()函数从键盘上获得一个字，在屏幕上显示的时候，如果字符超过了窗口右边界，则会被自动转移到下一行的开始位置。

实例 14-1　多窗口显示实例

实例说明	首先定义一个字符指针数组指向可用背景和前景显示七种颜色，然后用 for 循环连续开辟七个窗口，每个窗口背景色同指针 s[i]指向的颜色一样，每个窗口上显示大写的颜色名（由 cputs(s[i])），其颜色为窗口颜色的下一顺序颜色，偶数窗口颜色名用高亮度显示，奇数窗口用低亮度显示。
实例代码	光盘\\chapter14\14-1\
知识要点	文本屏幕输出。

（1）编辑源代码

启动 Turbo C 3.0 编译器，在编辑区中输入代码 14-1 中的代码。

代码 14-1　实例 14-1 的代码

```
01. #include <conio.h>
02. #include <stdio.h>
03. main()
04. {
05.     int i;
06.     char
*s[]={"BLACK","BLUE","GREEN","CYAN","RED","MAGENTA","BROWN","LIGHTGRAY"};
07.     textmode(C80);//设置显示文本方式 C80
08.     textbackground(0);//设置背景色
09.     clrscr();//清屏
10.     for(i=1;i<8;i++)
```

```
11.     {
12.         window(10+i*5,5+i,30+i*5,15+i);
13.         textbackground(i);
14.         clrscr();
15.         textcolor(7+i);//设置窗口内字符显示颜色
16.         if(i%2==0)
17.             highvideo();//i 为偶数的窗口字符高度显示
18.         else
19.             lowvideo();//奇数窗口字符低亮度显示
20.         cputs(s[i]);//输出字符到窗口内
21.     }
22.     getch();
23. }
```

代码执行步骤如下。

① 代码第 7 行调用 textmode 函数设置窗口文本显示方式为 C80。

② 代码第 8、9 行调用 textbackground 函数设置窗口背景颜色为黑色。然后调用 clrscr 函数清屏。

③ 代码第 10 到第 21 行通过 for 循环依次开辟了七个窗口，每个窗口的显示坐标由 i 决定，每个窗口背景色同指针 s[i]指向的颜色一样，每个窗口上显示大写的颜色名（由 cputs(s[i])），其颜色为窗口颜色的下一顺序颜色，偶数窗口颜色名用高亮度显示，奇数窗口用低亮度显示。

④ 代码第 22 行调用 getch 函数接收一个字符。

（2）保存源文件

按【F2】键，在弹出的保存文件窗口中，选择或输入文件保存到的目录并修改该文件名为 Ex14-1.c。

（3）编译、链接、运行程序

按【F9】键，编译并链接代码 14-1，然后按【Ctrl+F9】快捷键运行该程序。最后按【Alt+F5】快捷键查看程序运行结果。结果如图 14-2 所示。

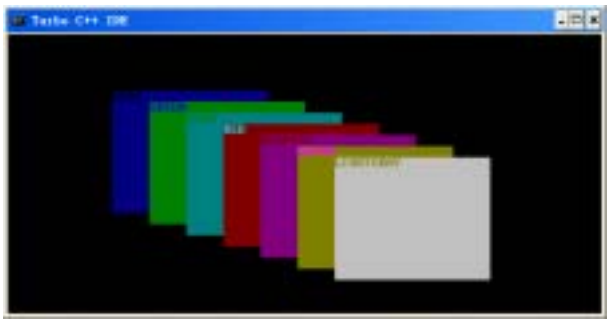

图 14-2　实例 14-1 的运行结果

5．有关屏幕操作的函数

有关屏幕操作的函数主要有：clrscr()函数、clreol()函数、delline()函数、gotoxy()函数、movetext()函数、gettext()函数和 puttext()函数。它们的调用形式和作用如下：

```
clrscr();
```

该函数将清除窗口中的文本，并将光标移到当前窗口的左上角，即（1, 1）处。

```
clreol();
```

该函数将清除当前窗口中从光标位置开始到本行结尾的所有字符，但不改变光标原来的位置。

```
delline();
```

该函数将删除一行字符，该行是光标所在行。

```
gotoxy(x, y);
```

该函数很有用，用来定位光标在当前窗口中的位置。这里 x，y 是指光标要定位处的坐标（相对于当前窗口而言）。当 x，y 超出了窗口的大小时，该函数就不起作用了。

```
movetext(x1, y1, x2, y2, x3, y3);
```

该函数将把屏幕上左上角为（xl, y1），右下角为（x2, y2）的矩形内文本拷贝到左上角为（x3, y3）的一个新矩形区内。这里 x，y 坐标是以整个屏幕为窗口坐标系，即屏幕左上角为（1, 1）。该函数与开设的窗口无关，且原矩形区文本不变。

```
gettext(xl, yl, x2, y2,buffer);
```

其中 buffer 是一个 void 类型指针。该函数将把左上角为（xl, y1），右下角为（x2, y2）的屏幕矩形区内的文本存到由指针 buffer 指向的一个内存缓冲区内，当操作成功，返回 1；否则，返回 0。

因一个在屏幕上显示的字符需占显示存储器 VRAM 的两字节，即第一字节是该字符的 ASCII 码，第二字节为属性字节，即表示其显示的前景、背景色及是否闪烁，所以 buffer 指向的内存缓冲区的字节总数的计算为：

字节总数=矩形内行数×每行列数×2

其中：矩形内行数=y2-y1+l，每行列数=x2-xl+1（每行列数是指矩形内每行的列数）。矩形内文本字符在缓冲区内存放的次序是从左到右，从上到下，每个字符占连续两字节并依次存放。

```
puttext(x1, y1, x2, y2, buffer);
```

其中 buffer 是一个 void 类型指针。该函数则是将 gettext()函数存入内存 buffer 中的文字内容拷贝到屏幕上指定的位置。

编者手记

这里提醒读者注意两点：

（1）gettext()函数和 puttext()函数中的坐标是对整个屏幕而言的，即是屏幕的绝对坐标，而不是相对窗口的坐标；

（2）movetext()函数是拷贝而不是移动窗口区域内容，即使用该函数后，原位置区域的文本内容仍然存在。

6．状态查询函数

有时需要知道当前屏幕的显示方式，当前窗口的坐标、当前光标的位置，文本的显示属性等，Turbo C 提供了一些函数用来得到这些信息。

（1）得到屏幕文本显示有关信息的函数 gettextinfo()，其调用形式为：

```
gettextinfo(struct text_info 类型指针);
```

这里的 text_info 是在 conio.h 头文件中定义的一个结构，该结构的定义是：

```
structtext_info(
    unsigned char winleft; /* 窗口左上角 x 坐标 */
    unsigned char wintop; /* 窗口左上角 y 坐标 */
    unsigned char winright; /* 窗口右下角 x 坐标 */
    unsigned char winbottom; /* 窗口右下角 y 坐标 */
    unsigned char attributes; /* 文本属性 */
    unsigned char normattr; /* 通常属性 */
    unsigned char currmode; /* 当前文本方式 */
    unsigned char screenheight; /* 屏高 */
    unsigned char screenwidth; /* 屏宽 */
    unsigned char curx; /* 当前光标的 x 值 */
    unsigned char cury; /* 当前光标的 y 值 */
};
```

（2）得到当前光标位置的函数 wherex()和 wherey()。它们的调用形式如下：

```
wherex();
wherey();
```

这两个函数将分别得到当前窗口中光标的 x 和 y 坐标。例如下面的调用，可以输出当前光标的 x，y 坐标：

```
printf("The cursor is at %d,%d\n",wherex(),wherey());
```

14.1.2 键盘输入

当我们按下键盘上某键时，系统如何知道某键被按下呢？简单的应用中可采用两种办法：一是直接使用 Turbo C 提供的键盘操作函数 bioskey()来识别，二是通过 int86()函数实现。

函数 bioskey()的调用的形式为：

```
bioskey(cmd);
```

它在 bios.h 头文件中进行了说明，该函数完成直接键盘操作，参数 cmd 的值决定执行什么操作。cmd 可以取 0、1、2 三个值。

当 cmd 是 0 时，bioskey 函数返回按键的键值，该值是一个 16 位的二进制数。若没有键按下，则该函数一直等待，直到有键按下。当按下一个普通键时，其返回值的低 8 位数存放该字符的 ASCII 码；对于特殊键（如方向键、F1～F12 等等），低 8 位为 0，高 8 位字节存放该键的扫描码。

当 cmd 是 1 时，bioskey 函数查询是否有键按下。若有则返回非零值，否则返回 0。

当 cmd 是 2 时，bioskey 函数将返回一些控制键是否被按过，按过的状态由该函数返回的低 8 位的各位值来表示，如表 14-3 所示。

表 14-3　参数 cmd 为 2 时低 8 位的各位值

cmd	对应的 16 进制数	含　义
0	0x01	右边的【Shift】键被按下
1	0x02	左边的【Shift】键被按下
2	0x04	【Ctrl】键被按下
3	0x08	【Alt】键被按下
4	0x10	【Scroll Lock】已打开
5	0x20	【Num Lock】已打开
6	0x40	【Caps Lock】已打开
7	0x80	【Inset】已打开

当某位为 1 时，表示相应的键已按，或相应的控制功能已有效，如选参数 cmd 为 1，若 key 值为 0x09，则表示右边的【Shift】键被按，同时又按了【Alt】键。

函数 int86()的调用形式为：

```
int86(中断号,&输入指针名,&输出指针名);
```

这个函数在 bios.h 头文件中进行了说明，它的第一个参数 intr_num 表示 BIOS 调用类型号，相当于 int n 调用的中断类型号 n；第二个参数表示是指向联合类型 REGS 的指针，它用于接收调用的功能号及其他一些指定的入口参数，以便传给相应的寄存器，第三个参数也是一个指向联合类型 REGS 的指针，它用于接收功能调用后的返回值，即出口参数，如调用的结果、状态信息，这些值从相关寄存器中得到。

实例 14-2　将屏幕平分成两个窗口

实例说明	编写一段 C 代码，实现将屏幕垂直平分成两个窗口，左边窗口为蓝色背景，白色前景，右边窗口为绿色背景，黄色前景。两个窗口都设计为文本输入，即在窗口中可以输入文字，在窗口屏幕中显示出来。使用【Tab】键在左右两个窗口中切换，每个窗口都有光标，活动窗口光标进行闪烁。
实例代码	光盘\\chapter14\14-2\
知识要点	文本的屏幕输出和键盘输入。

（1）编辑源代码

启动 Turbo C 3.0 编译器，在编辑区中输入代码 14-2 中的代码。

代码 14-2　实例 14-2 的代码

```
01. #include <stdio.h>
02. #include <conio.h>
03. #include <bios.h>
04. char leftbuf[40*25*2];/*切换时保存左窗口文本*/
05. char rightbuf[40*25*2];/*切换时保存右窗口文本*/
```

```
06. int leftx, lefty;/*切换时保存左窗口当前坐标*/
07. int rightx, righty;/*切换时保存右窗口当前坐标*/
08. void draw_left_win();/*重绘左边窗口*/
09. void draw_right_win();/*重绘右边窗口*/
10. int main()
11. {
12.     int key;
13.     int turn;
14.     textmode(C80);//设置显示文本方式 C80
15.     textbackground(0);//设置背景色
16.     textcolor(WHITE);//设置前景色即文本字符的颜色
17.     clrscr();//清屏
18.     gotoxy(60,1);//定位光标在当前窗口的（60，1）处
19.     cprintf("Press Esc to Quit");//输出一行字符串
20.     /*右边窗口为绿色背景，黄色前景*/
21.     window(41,2,79,24); //绘制右窗口
22.     textbackground(2);//设置右窗口背景色
23.     textcolor(14);//设置右窗口前景色
24.     clrscr();//清屏
25.     gettext(41,2,79,24, rightbuf);//保存右窗口中的文本
26.     /*左边窗口为蓝色背景，白色前景*/
27.     window(2,2,40,24);//绘制左窗口
28.     textbackground(1);//设置左窗口背景色
29.     textcolor(15);//设置左窗口前景色
30.     clrscr();//清屏
31.     gettext(2,2,40,24, leftbuf);//保存左窗口中的文本
32.     turn = 1; /*初始激活右窗口*/
33.     for(;;)
34.     {
35.         key=bioskey(0);
36.         if(key == 0x011b)
37.             exit(0);
38.         key=key&0xff; /*获取窗口输入的文本的 ASCII 码值*/
39.         if(key == '\t')
40.         {
41.             if(turn == 1) /*切换到右窗口*/
42.             {
43.                 gettext(2,2,40,24, leftbuf);
44.                 leftx = wherex();
45.                 lefty = wherey();
46.                 draw_right_win();
47.                 turn = 0;
48.             }
49.             else if(turn == 0) /*切换到左窗口*/
50.             {
51.                 gettext(41,2,79,24, rightbuf);
52.                 rightx = wherex();
53.                 righty = wherey();
54.                 draw_left_win();
55.                 turn = 1;
56.             }
57.         }
```


```
58.          else
59.              putch(key); /*当前光标处显示新输入的文本字符*/
60.      }
61. }
62. void draw_right_win()//重绘右窗口函数
63. {
64.      window(41,2,79,24);
65.      textbackground(2);
66.      textcolor(14);
67.      clrscr();
68.      puttext(41,2,79,24, rightbuf);
69.      gotoxy(rightx, righty);
70. }
71. void draw_left_win()//重绘左窗口函数
72. {
73.      window(2,2,40,24);
74.      textbackground(1);
75.      textcolor(15);
76.      clrscr();
77.      puttext(2,2,40,24, leftbuf);
78.      gotoxy(leftx, lefty);
79. }
```

代码执行步骤如下。

① 代码第 1 行到第 9 行先是包含相应的头文件，然后定义了相关的变量，最后声明将在 main 函数中被调用的两个自定义函数。

② 代码第 12、13 行定义了两个整型变量 key 和 turn。其中 key 用来存储用户当前按键值，turn 用来实现左右窗口间的切换。

③ 代码第 14 行到第 17 行是设置窗口显示文本方式为 C80，设置窗口背景色为黑色，设置窗口前景色为白色，然后清屏。

④ 代码第 31、32 行是将左窗口的文本保存在字符数组 leftbuf[ ]中，然后定义 turn 的值等于 1。

⑤ 代码第 33 行到第 61 行是一个 for 循环，该循环将永远执行下去，直到循环体中执行了退出循环或退出程序的命令。代码第 35 行是调用 bioskey(0)获得当前用户按下的键值，并存储在变量 key 中。代码第 36 行是判断 key 是否等于 0x011b（即键【Esc】），若是则执行代码第 37 行退出本程序，若不是则继续执行其下面的代码。

编者手记

这里的 16 进制数 0x011b 为键【Esc】的键盘扫描码。有关键盘扫描码的内容超出了本书的范围，这里就不做介绍，若读者想要了解相关的内容，请读者查阅相关的资料。

代码第 38 行将 key 的值和 0xff 相与获得用户所按键的 ASCII 码值，并重新保存在 key 中。代码第 39 行是判断该键是否为【Tab】键（ASCII 码值等于字符‘\t’），若是则执行代码第 41

行再判断 turn 的值是否等于 1，若是则执行代码第 43 行到第 47 行，先调用 gettext 函数将左窗口中文本保存到 leftbuf[ ]数组中，接着调用 wherex 和 wherey 函数获得当前光标的位置，然后调用 draw_right_win()函数重新绘制右窗口，最后将 turn 的值设置为 0；若 turn 的值等于 0 则执行代码第 51 行到第 55 行，先调用 gettext 函数将右窗口中文本保存到 rightbuf[ ]数组中，接着调用 wherex 和 wherey 函数获得；若该键不是为【Tab】键则指向代码第 59 行在当前光标处显示新输入的文本字符。

⑥ 代码第 62 行定义 draw_right_win 函数实现重新绘制右窗口，该函数返回为 void 类型，函数无参数，其函数体为代码第 64 行到第 69 行。

⑦ 代码第 64 行是调用 window 函数绘制右窗口；代码第 65 行设置右窗口背景色；代码第 66 行设置右窗口前景色；代码第 67 行清屏；代码第 68 行将当前右窗口中的文本保存到字符数组 rightbuf[ ]中；代码第 69 行定位右窗口中光标的位置。

⑧ 代码第 71 行定义 draw_left_win 函数实现重新绘制左窗口，该函数返回为 void 类型，函数无参数，其函数体为代码第 73 行到代码第 78 行。

⑨ 代码第 73 行是调用 window 函数绘制左窗口；代码第 74 行设置左窗口背景色；代码第 75 行设置左窗口前景色；代码第 76 行清屏；代码第 77 行将当前左窗口中的文本保存到字符数组 leftbuf[ ]中；代码第 78 行定位左窗口中光标的位置。

（2）保存源文件

按【F2】键，在弹出的保存文件窗口中，选择或输入文件保存到的目录并修改该文件名为 Ex14-2.c。

（3）编译、链接、运行程序

按【F9】键，编译并链接代码 14-2，然后按【Ctrl+F9】快捷键运行该程序。最后按【Alt+F5】快捷键查看程序运行结果。结果如图 14-3 所示。

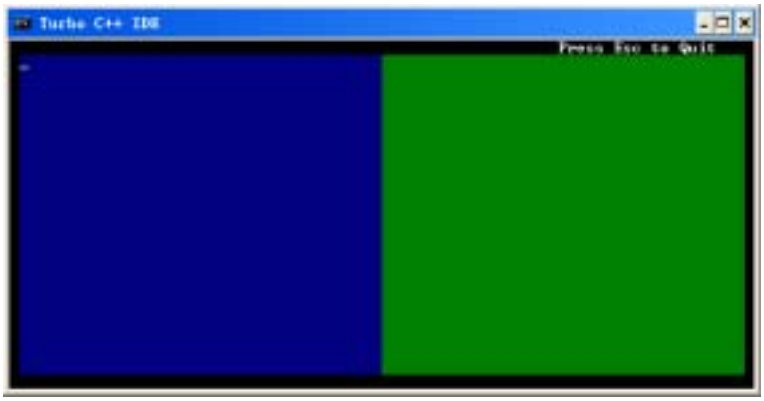

图 14-3　实例 14-2 的运行结果

接着可在两个窗口中输入文本，并使用【Tab】键在左右两个窗口中切换。例如在左窗口中输入字符串“hello”，然后按【Tab】键则切换到右窗口，在右窗口中输入字符串“nihao”，如图 14-4 所示。

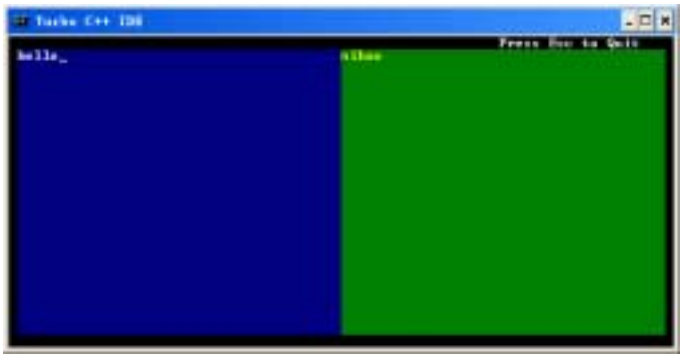

图 14-4　在左右窗口中输入文本

本程序首先用文本窗口函数 window()画出两个窗口，用 textcolor()，textbackground()，clrscr()等进行窗口属性的设置。用【Tab】键进行两个窗口间的循环切换，在每次切换前先调用 gettext()函数把当前矩形窗口上的字符拷贝到由 buf 所指向的内存中，在切换到另一个窗口后调用 puttext()把先前存储在该窗口 buf 所指向的内存中的字符拷贝到当前窗口中，并用 gotoxy()把光标移到原先所在位置，因此可以接着先前的文本继续编辑。

程序中运用一个临时变量 turn 的值 0 和 1 来进行两个窗口间的循环切换，当值为 0 时调用 draw_left_win()函数重绘左边窗口，当值为 1 时调用 draw_right_win()函数重绘右边窗口。程序用 bioskey(0)来读取键盘输入的文本并存在内存中，然后用 putch(key)输出。

14.2　Turbo C 图形处理

计算机图形程序设计是程序设计中较难且又最吸引人的部分。为了用户设计图形程序方便，不同版本和公司的 C 编译系统都提供了许多画图库函数，用户设计图形程序时，只要在需要的地方，设置相应的参数对其调用即可（这些画图库函数不是 C 语言标准要求的，一般 C 语言程序设计课本中没做介绍）。

Turbo C 提供了非常丰富的图形函数以实现图形程序设计，所有图形函数的原型均在头文件 graphics.h 中，本节将介绍编制图形程序的各种库函数和相应的其他一些知识。

使用图形函数时要确保有显示器图形驱动程序“*.BGI”，同时将集成开发环境 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.

14.2.1　图形模式的初始化和关闭

不同的显示器适配器有不同的图形分辨率。即使同一显示器适配器，在不同模式下也有不同分辨率。因此，在屏幕作图之前，必须根据显示器适配器种类将显示器设置成为某种图形模式，在未设置图形模式之前，微机系统默认屏幕为文本模式（80 列，25 行字符模式），此时所有图形函数均不能工作。

1．图形模式的初始化函数

Turbo C 提供了函数 initgraph 用来设置屏幕为图形模式，其原型为：

```
void far initgraph(int far *gdriver, int far *gmode, char *path);
```

编者手记

注意其中的 far 表示远指针，具体介绍请读者查阅相关资料。

其中 gdriver 和 gmode 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。有关图形驱动器、图形模式的符号常数及对应的分辨率见表 14-4。

图形驱动程序由 Turbo C 出版商提供，文件扩展名为.BGI。根据不同的图形适配器有不同的图形驱动程序。例如对于 EGA、VGA 图形适配器就调用驱动程序 EGAVGA.BGI。

表 14-4　图形驱动器、模式的符号常数及数值

图形驱动器（gdriver）		图形模式（gmode）		色　调	分 辨 率	页　数
符号常数	数　值	符号常数	数　值			
CGA	1	CGAC0	0	C0	320*200	1
		CGAC1	1	C1	320*200	1
		CGAC2	2	C2	320*200	1
		CGAC3	3	C3	320*200	1
		CGAHI	4	2 色	640*200	1
MCGA	2	MCGAC0	0	C0	320*200	1
		MCGAC1	1	C1	320*200	1
		MCGAC2	2	C2	320*200	1
		MCGAC3	3	C3	320*200	1
		MCGAMED	4	2 色	640*200	1
		MCGAHI	5	2 色	640*480	1
EGA	3	EGALO	0	16 色	640*200	4
		EGAHI	1	16 色	640*350	2
EGA64	4	EGA64LO	0	16 色	640*200	1
		EGA64HI	1	4 色	640*350	1
EGAMON	5	EGAMONHI	0	2 色	640*350	1
IBM8514	6	IBM8514LO	0	256 色	640*480	
		IBM8514HI	1	256 色	1024*768	
HERC	7	HERCMONOHI	0	2 色	720*348	1
ATT400	8	ATT400C0	0	C0	320*200	1
		ATT400C1	1	C1	320*200	1
		ATT400C2	2	C2	320*200	1
		ATT400C3	3	C3	320*200	1
		ATT400MED	4	2 色	320*200	1
		ATT400HI	5	2 色	320*200	1
VGA	9	VGALO	0	16 色	640*200	2
		VGAMED	1	16 色	640*350	2
		VGAHI	2	16 色	640*480	1
PC3270	10	PC3270HI	0	2 色	720*350	1
DETECT	0	用于硬件测试				

当我们使用的存储模式为 tiny（微型）、small（小型）或 medium（中型）时，不需要远指针，因而可以将初始化函数调用格式写成如下形式（该说明适用于后面所述的任一函数）：

```
initgraph(&graphdriver, &graphmode,"");
```

其中驱动程序目录路径为空字符（“”）时，表示就在当前目录下。当我们不知道所用显示适配器名称时，可将 graphdriver 设成 DETECT，它将自动检测所用显示适配器类型，并将相应的驱动程序装入，将其最高的显示模式作为当前显示模式，如表 14-5 所示。

表 14-5　检测到的适配器及其选中的显示模式

检测到的适配器	选中的显示模式
CGA	4（640×200，2 色即 CGAHI）
EGA	1（640×350，16 色，即 EGAHI）
VGA	2（640×480，16 色，即 VGAHI）

一旦执行了初始化，显示器即被设置成相应模式的图形方式。

例如下面的代码，是一般画图程序的开始部分，它包括对图形模式的初始化：

```
#include <graphics.h>
main()
{
    int graphdriver=DETECT;
    int graphmode;
    initgraph(&graphdriver,&graphmode,"");
    …
}
```

上面初始化过程中，将由 DETECT 检测所用适配器类型，并将当前目录下相应的驱动程序装入，并采用最高分辨率显示模式作为 graphmode 的值。

若已知所用图形适配器为 VGA 时，想采用 640×480 的高分辨显示模式 VGAHI，则图形初始化部分可写成：

```
int graphdriver=VGA;
int graphmode=VGAHI;
initgraph(&graphdriver,&graphmode,"")
```

2．自动检测显示器硬件的函数

当 graphdriver=DETECT 时，实际上 initgraph 函数又调用了自动检测显示器硬件的函数 detectgraph，它完成对适配器的检查并得到显示器类型号和相应的最高分辨率模式，若所设适配器不是规定的那些类型，则返回-2，表示适配器不存在，该函数的原型说明是：

```
void far detectgraph(int far *gdriver, int far *gmode);
```

其中 gdriver 和 gmode 的意义与上面相同。调用 detectgraph 时，该函数将把检测到的适配器类型赋予 gdriver，再把该类型适配器支持的最高分辨率模式赋给 gmode。例如下面的代码，自动进行硬件测试后进行图形初始化。

```
#include <graphics.h>
int main()
```

```
{
    int gdriver, gmode;
    detectgraph(&gdriver, &gmode);/*自动测试硬件*/
    /*输出测试结果*/
    printf("the graphics driver is %d, mode is %d\n",gdriver,gmode);
getch();
    initgraph(&gdriver, &gmode, "c:\\tc"); /* 根据测试结果初始化图形*/
    …
}
```

上例程序中先对图形显示器自动检测，然后再用图形初始化函数进行初始化设置。但更简单的方法还是在前面提到的方法，即用 gdriver= DETECT 语句后再跟 initgraph()函数就行了。

3．清屏和恢复显示方式的函数

画图前一般需清除屏幕，使得屏幕如同一张白纸，以画最新最美的图画，因而必须使用清屏函数。清屏函数的原型是：

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

该函数作用范围为整个屏幕，如果用函数 setviewport 定义一个图视窗口，则可用清除图视口函数 clearviewport，它仅清除图视口区域内的内容。

编者手记

关于 setviewport 和 clearviewport 函数将在本节后面介绍图视口操作函数时介绍。

当画图程序结束，回到文本方式时，要退出图形模式，回到文本方式，可使用 closegraph 函数，该函数的说明原型是：

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

由于进入 C 环境进行编程时，即进入文本方式，因而为了在画图程序结束后恢复原来的最初状况，一般在画图程序结束前调用该函数，使其恢复到文本方式。

为了不关闭图形系统，使相应适配器的驱动程序和字符集（字库）仍驻留在内存，但又回到原来所设置的模式，则可用恢复工作模式函数，它也同时进行清屏操作，它的说明原型是：

```
void far restorecrtmode(void);
```

该函数常和另一设置图形工作模式函数 setgraphmode 交互使用，使得显示器工作方式在图形和文本方式之间来回切换，这在编制菜单程序和说明程序时很有用处。

14.2.2 独立图形运行程序的建立

Turbo C 对于用 initgraph()函数直接进行的图形初始化程序，在编译和链接时并没有将相应的驱动程序（*.BGI）装入到执行程序，当程序进行到 intitgraph()语句时，再从该函数中第三个形式参数 char *path 中所规定的路径中去找相应的驱动程序。若没有驱动程序，则在 C:\TC 中去找，如 C:\TC 中仍没有或 TC 不存在，将会出现错误：

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

因此，为了使用方便，可建立一个不需要驱动程序就能独立运行的可执行图形程序，Turbo C 中规定用下述步骤（这里以 EGA、VGA 显示器为例）：

（1）在 C:\TC 子目录下输入命令：

```
BGIOBJ EGAVGA
```

此命令将驱动程序 EGAVGA.BGI 转换成 EGAVGA.OBJ 的目标文件。

（2）在 C:\TC 子目录下输入命令：

```
TLIB LIB\GRAPHICS.LIB+EGAVGA
```

此命令的意思是将 EGAVGA.OBJ 的目标模块装到 GRAPHICS.LIB 库文件中。

（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中。

经过上面处理，编译链接后的执行程序可在任何目录或其他兼容机上运行。例如下面的代码（假设已做了前两个步骤）。

```
#include<stdio.h>
#include<graphics.h>
int main()
{
    int gdriver=DETECT,gmode;
    registerbgidriver(EGAVGA_driver);  / *建立独立图形运行程序 */
    initgraph(gdriver, gmode,"c:\\tc");
    …
}
```

上例编译链接后产生的执行程序可独立运行。如不初始化成 EGA 或 CGA 分辨率，而想初始化为 CGA 分辨率，则只需要将上述步骤中有 EGA 和 VGA 的地方用 CGA 代替即可。

14.2.3 屏幕颜色的设置和清屏函数

对于图形模式的屏幕颜色设置，同样分为背景色的设置和前景色的设置。在 Turbo C 中分别用下面两个函数。

（1）设置背景色函数

```
void far setbkcolor(int color);
```

该函数将当前图形屏幕的背景色设置为 color。

（2）设置作图色（即前景色）函数

```
void far setcolor(int color);
```

该函数将当前图形屏幕的当前笔画颜色置为 color。

其中 color 为图形方式下颜色的规定数值，对 EGA，VGA 显示器适配器，有关颜色的符号常数及数值见表 14-6 所示。调用函数时参数 color 可以是符号常数也可以是相应的数值。

表 14-6　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
BLACK	0	黑色	DARKGRAY	8	深灰
BLUE	1	蓝色	LIGHTBLUE	9	深蓝
GREEN	2	绿色	LIGHTGREEN	10	淡绿
CYAN	3	青色	LIGHTCYAN	11	淡青
RED	4	红色	LIGHTRED	12	淡红
MAGENTA	5	洋红	LIGHTMAGENTA	13	淡洋红
BROWN	6	棕色	YELLOW	14	黄色
LIGHTGRAY	7	淡灰	WHITE	15	白色

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);
```

编者手记

另外还有下面几个函数，在编程中可能会用到。

getbkcolor() 返回背景色函数

getcolor() 返回当前绘图颜色

getmaxcolor() 返回最大颜色值函数

实例 14-3　有关颜色设置、清屏函数的使用

实例说明	编写一段 C 代码，实现以不同的颜色和半径在图形模式下画圆，然后以默认的颜色和不同的半径画圆并同时改变背景色。
实例代码	光盘\\chapter14\14-3\
知识要点	屏幕颜色的设置和清屏函数的使用

（1）编辑源代码

启动 Turbo C 3.0 编译器，在编辑区中输入代码 14-3 中的代码。

代码 14-3　实例 14-3 的代码

```
01. #include<stdio.h>
02. #include<graphics.h>
03. void main()
04. {
05.     int gdriver, gmode, i;
06.     gdriver=DETECT;
07.     initgraph(&gdriver, &gmode, "c:\\tc\\bgi");/*图形初始化*/
08.     setbkcolor(0);                  //设置图形背景
09.     cleardevice();                  //清除屏幕
10.     for(i=0; i<=15; i++)
11.     {
12.         setcolor(i);               //设置不同作图色
13.         circle(320, 240, 20+i*10);  //画半径不同的圆
```

```
14.         delay(500);                 /*延迟 500 毫秒*/
15.     }
16.     for(i=0; i<=15; i++)
17.     {
18.         setbkcolor(i);              /*设置不同背景色*/
19.         cleardevice();
20.         circle(320, 240, 20+i*10);
21.         delay(500);
22.     }
23.     closegraph();                   //退出图形模式
24. }
```

代码执行步骤如下。

① 代码第 5、6 行定义三个整型变量。其中 gdriver 表示图形驱动器，gmode 表示图形模式，变量 i 为循环变量。代码第 6 行将 gdriver 定义为 DETECT。

② 代码第 7 行调用 initgraph 函数初始化图形模式。

③ 代码第 8、9 行先设置图形背景色为黑色，然后清除屏幕，使得屏幕如同一张白纸，以画最新最美的图画。

④ 代码第 10 到第 15 行通过 for 循环实现以不同的颜色，从小到大画圆。代码第 12 行是设置不同的作图色，代码第 13 行是调用 circle 画圆函数以坐标（320，240），20+i*10（i 是变量）半径画圆。代码第 14 行是调用延时函数 delay，以便延时 500 毫秒后再画下一个圆。

编者手记

关于 circle 函数将在随后一节中介绍，而延时函数 delay，这里做一个简单的说明。该函数的作用就是将程序的执行暂停一段时间（单位毫秒），它的调用形式为：

```
delay(n);
```

其中，n 为暂停的毫秒数。

⑤ 代码第 16 到第 22 行也是用 for 循环实现以默认的作图颜色，从小到大画圆并改变背景色。代码第 18 行是设置图形模式的不同背景色，代码第 19 行是调用 cleardevice()清屏。代码第 20 行调用 circle 函数以不同的半径画圆。代码第 21 调用函数 delay 函数延时 500 毫秒指向下一次循环或执行 for 循环下的代码。

⑥ 代码第 23 行是调用 closegraph()函数退出图形模式。

（2）保存源文件

按【F2】键，在弹出的保存文件窗口中，选择或输入文件保存到的目录并修改该文件名为 Ex14-3.c。

（3）编译、链接、运行程序

按【F9】键，编译并链接代码 14-3，然后按【Ctrl+F9】快捷键运行该程序。便会看到屏幕上显示慢慢出现以不同颜色所画的，从小到大出现的圆，然后出现以白色画的，从小到大出现的圆，同时其对应的背景也在随之变化。

另外，Turbo C 也提供了几个获得现行颜色设置情况的函数。

```
int far getbkcolor(void);   //返回现行背景颜色值
int far getcolor(void);     //返回现行作图颜色值
int far getmaxcolor(void);  //返回最高可用的颜色值
```

14.2.4 基本图形函数

图形由点、线、面组成，Turbo C 提供了一些函数，以完成这些操作，而所谓面则可由对封闭图形填上颜色来实现。当图形模式初始化后，在此阶段将要进行的画图操作均采用默认值作为参数的当前值，如画图屏幕为全屏，当前开始画图坐标为（0, 0）（又称当前画笔位置，虽然这个笔是无形的），又如采用画图的背景颜色和前景颜色、图形的填充方式，以及可以采用的字符集（字库）等均为默认。

1．画点函数

该函数原型为：

```
void far putpixel(int x, int y, int color);
```

该函数表示由指定的像素画一个按 color 所确定颜色的点。对于颜色 color 的值可从表 14-6 中获得，而对 x, y 是指图形像素的坐标。

编者手记

这里特别说明一下：

在图形模式下，是按像素来定义坐标的。例如 VGA 适配器，它的最高分辨率为 640×480，其中 640 为整个屏幕从左到右所有像素的个数，480 为整个屏幕从上到下所有象员的个数。屏幕的左上角坐标为（0, 0），右下角坐标为（639, 479），水平方向从左到右为 x 轴正向，垂直方向从上到下为 y 轴正向，如图 14-5 所示。

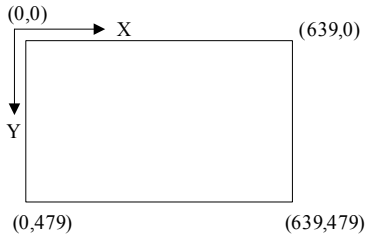

图 14-5　显示屏在 640×480 分辨率下的坐标

TURBO C 的图形函数都是相对于图形屏幕坐标，即像素来说的。

关于点的另外一个函数是：

```
int far getpixel(int x, int y);
```

该函数与 putpixel()相对应，它用来获得在（x, y）点位置上像素的颜色值。

2．有关画图坐标位置的函数

在屏幕上画线时，如同在纸上画线一样。画笔要放在开始画图的位置，并经常要抬笔移动，以便到另一位置再画。我们也可想象在屏上画图时，有一无形的画笔，可以控制它的定位、移动等，也可知道它能移动的最大位置限制等。完成这些功能的函数有：

（1）移动画笔到指定的（x, y）位置，移动过程不画：

```
void far moveto(int x，int y);
```

（2）画笔从现行位置（x, y）处移到一位置增量处（x+dx，y+dx），移动过程不画：

```
void far moverel(int dx，int dy);
```

（3）得到当前画笔所在位置。

```
int far getx(void);
```

得到当前画笔的 x 位置。

```
int far gety(void);
```

得到当前画笔的 y 位置。

另外，还有两个函数 getmaxx 函数和 getmaxy，它们分别用来获得 x 轴和 y 轴的最大值。它们的原型为：

```
int far getmaxx(void);
int far getmaxy(void);
```

3．画线函数

这类函数提供了从一个点到另一个点用设定的颜色画一条直线的功能，起始点的设定方法不同，因而有下面不同的画线函数。

（1）两点之间画线函数。

```
void far line(int x0，int y0，int x1，int y1);
```

从（x0, y0）点到（x1, y1）点画一直线。

（2）从现行画笔位置到某点画线函数。

```
void far lineto(int x，int y);
```

将从现行画笔位置到（x，y）点画一直线。

（3）从现行画笔位置到一增量位置画线函数

```
void far linerel(int dx，int dy);
```

将从现行画笔位置（x，y）到按相对增量确定的点处（x+dx，y+dy）画一直线。

4．画矩形函数

画矩形函数 rectangle 将画出一个矩形框，它的原型为：

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

该函数将以（x1, y1）为左上角，（x2, y2）为右下角画一矩形框。

5．画椭圆、圆及圆弧的函数

在画图的函数中，有关于角的概念。在 Turbo C 中是这样规定的：屏的 x 轴方向为 0°，当半径从此处逆时针方向旋转时，则依次是 90°、180°、270°，当 360°时，则和 x 轴正向重合，即旋转了一周。如图 14-6 所示。

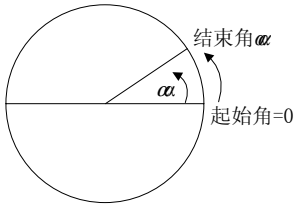

图 14-6　起始角和终止角

（1）画椭圆函数

```
void ellipse(int x，int y，int stangle，int endangel，int xradius，int yradius);
```

该函数将以（x, y）为中心，以 xradius 和 yradius 为 x 轴和 y 轴半径，从起始角 stangle 开始到 endangle 角结束，画一椭圆线。当 stangle=0，endangle=360 时，则画出的是一个完整的椭圆，否则画出的将是椭圆弧。关于起始角和终止角规定如图 14-5 所示。

（2）画圆函数

```
void far circle(int x，int y，int radius);
```

该函数将以（x, y）为圆心，radius 为半径画个圆。

（3）画圆弧函数

```
void far arc(int x，int y，int stangle，int endangle，int radius);
```

该函数将以（x, y）为圆心，radius 为半径，从 stangle 为起始角开始，到 endangle 为结束角画一圆弧。

6．画多边形函数

画多边形函数为 drawpoly，其原型为：

```
void far drawpoly(int numpoints, int far *polypoints);
```

画一个顶点数为 numpoints，各顶点坐标是由 polypoints 给出的多边形。polypoints 整型数组必须至少有 2 倍顶点数个元素。每一个顶点的坐标都定义为 x，y，并且 x 在前。值得注意的是，当画一个封闭的多边形时，numpoints 的值取实际多边形的顶点数加 1，并且数组 polypoints 中第一个和最后一个点的坐标相同。

实例 14-4　用 drawpoly()函数画一个箭头

实例说明	用 drawpoly()函数在蓝色背景下用淡红色画一个空心箭头。
实例代码	光盘\\chapter14\14-4\
知识要点	drawpoly()函数的使用。

（1）编辑源代码

启动 Turbo C 3.0 编译器，在编辑区中输入代码 14-4 中的代码。

代码 14-4　实例 14-4 的代码

```
#include<stdlib.h>
#include<graphics.h>
void main()
{
    int gdriver, gmode, i;
    int arw[16]={200, 102, 300, 102, 300, 107, 330,100, 300, 93, 300, 98,
200, 98, 200, 102}; //定义数组
    gdriver=DETECT;
    initgraph(&gdriver, &gmode, "c:\\tc\\bgi");
    setbkcolor(BLUE); //设置背景色为蓝色
    cleardevice(); //清屏
    setcolor(12);       /*设置作图颜色为淡红色*/
    drawpoly(8,arw);   /*画一箭头*/
    getch();
    closegraph(); //退出图形模式
}
```

该代码先是定义了三个整型变量，然后定义了一个整型数组 arw，该整型数组存储了箭头的各个顶点的坐标。接着初始化图形模式，设置背景色，清屏后再设置作图色。然后调用 drawpoly 函数画一箭头。然后调用 getch 函数接收一个字符。最后退出图形模式。

（2）保存源文件

按【F2】键，在弹出的保存文件窗口中，选择或输入文件保存到的目录并修改该文件名为 Ex14-4.c。

（3）编译、链接、运行程序

按【F9】键，编译并链接代码 14-4，然后按【Ctrl+F9】快捷键运行该程序。该程序的运行结果如图 14-7 所示。

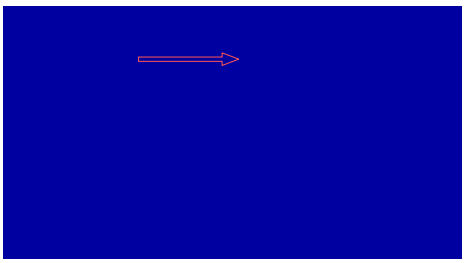

图 14-7　用 drawpoly()函数画一个箭头

14.2.5 画线的线型函数

在没有对线的特性进行设定之前，Turbo C 用其默认值，即一点宽的实线，但 Turbo C 也提供了可以改变线型的函数。线型包括：宽度和形状。其中宽度只有两种选择：一点宽和三点宽。而线的形状则有五种。下面介绍有关线型的设置函数。

1．设定线型函数

Turbo C 提供了函数 setlinestyle 用来改变线的宽度、类型，其原型为：

```
void far setlinestyle(int linestyle, unsigned upattern, int thickness);
```

当线的宽度参数（thickness）不设定时，取默认值，即 1 像素宽，当设定为 3 时，可取 3 像素宽，取值见表 14-7 所示。

表 14-7　线宽（thickness）

符 号 名	值	含　义
NORM_WIDTH	1	1 像素宽
THICK_WIDTH	3	3 像素宽

当线型参数（1inestyle）不设定时，取默认值，即实线；设定时，可有 5 种选择如表 14-8 所列。

表 14-8　线的形状（linestyle）

符 号 名	值	含　义
SOLID_LINE	0	实线
DOTTED_LINE	1	点线
CENTER_LINE	2	中心线
DASHED_LINE	3	点画线
USERBIT_LINE	4	用户自定义线

upattern 参数只有在 1inestyle 取 4 或 USERBIT_LINE 时才有意义，即表示在用户自定义线型时，该参数才有用。该参数若表示成 16 位二进制数，则每一位代表一像素。若该位是 1，则代表的像素用前景色显示，若是 0，则代表的像素用背景色显示（实际没有显示）。例如图 14-8 表示了由 16 像素构成的一个 16 像素长的线段，线宽为 1 像素宽。当 1inestyle 不为 USERBIT_LINE 值时，虽然 upattern 的值不起作用，但仍需为它提供一个值，一般取为 0。

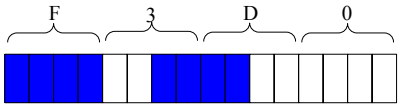

图 14-8　用 setlinestyle(4，0xF3D0，1)设置的线型

编者手记

通过图 14-7 的例子，可以推则：当 upattern=0xFFFF（16 位二进制数为

1111111111111111），则画实线；当 upattern=0x9999（16 位二进制数为 1001100110011001），则画每隔两像素交替显示的虚线；如果要画长虚线，那么 upattern 的值可为 0xFF00（16 位二进制数为 1111111100000000）和 0xF00F（1111000000001111）。

2．得到当前画线信息的函数

与设定线型函数 setlinestyle 相对应的是得到当前有关线的信息的函数：

```
void far getlinesettings(struct linesettingstype far *lineinfo);
```

该函数将把当前有关线的信息存放到由 lineinfo 指向的结构中，其结构 linesetingstype 定义如下：

```
structlinesettingstype
{
    int linestyle;
    unsigned upattern;
    int thickness;
};
```

例如下面两句程序可以读出当前线的特性。

```
struct linesettingstype *info;
getlinesettings(info);
```

3．设定画线方式函数

Turbo C 还提供了用来规定画线的方式的函数：

```
void far setwritemode(int mode);
```

如果 mode=0，则表示画线时将所画位置的原来信息覆盖了（这是 Turbo C 的默认方式）。如果 mode=1，则表示画线时用现在特性的线与所画之处原有的线进行异或（XOR）操作，实际上画出的线是原有线与现在规定的线进行异或后的结果。因此，当线的特性不变时，进行两次画线操作相当于没有画线。

实例 14-5　使用画线的线型函数作图

实例说明	编写一段 C 代码，实现先在图形模式下画一圆，然后在圆外以三点宽实线画矩形，然后以一点宽用户定义线画两条直线。
实例代码	光盘\\chapter14\14-5\
知识要点	线型函数的使用。

（1）编辑源代码

启动 Turbo C 3.0 编译器，在编辑区中输入代码 14-5 中的代码。

代码 14-5　实例 14-5 的代码

```
01. #include<stdlib.h>
02. #include<graphics.h>
```

```
03. void main()
04. {
05.     int gdriver, gmode, i;
06.     gdriver=DETECT;
07.     initgraph(&gdriver, &gmode, "c:\\tc\\bgi");
08.     setbkcolor(BLUE); //设置背景色
09.     cleardevice(); //清屏
10.     setcolor(RED); //设置作图色
11.     circle(320, 240, 98); //画圆
12.     setlinestyle(0, 0, 3);   /*设置三点宽实线*/
13.     setcolor(2); //设置作图色
14.     rectangle(220, 140, 420, 340); //画矩形
15.     setcolor(WHITE); //设置作图色
16.     setlinestyle(4, 0xaaaa, 1);   /*设置一点宽用户定义线*/
17.     line(220, 240, 420, 240); //画线
18.     line(320, 140, 320, 340); //画线
19.     getch();
20.     closegraph();//退出图形模式
21. }
```

代码执行步骤如下。

① 代码第 5 行到第 7 行是初始化图形模式。

② 代码第 8、9 行是设置背景色为蓝色并清屏。

③ 代码第 10、11 行是以红色为作图色画一个以坐标（320, 240）为圆心，半径为 98 的圆。

④ 代码第 12、13、14 行是先设置画线方式为三点宽实线，然后设置作图色为绿色，最后画一个矩形框。

⑤ 代码第 15 行到第 18 行是先设置作图色为白色，然后设置画线方式为一点宽用户定义线，最后画两条垂直的线。

⑥ 代码第 19、20 行接收用户输入的一个字符，然后退出图形模式。

（2）保存源文件

按【F2】键，在弹出的保存文件窗口中，选择或输入文件保存到的目录并修改该文件名为 Ex14-5.c。

（3）编译、链接、运行程序

按【F9】键，编译并链接代码 14-5，然后按【Ctrl+F9】快捷键运行该程序。该程序的运行结果如图 14-9 所示。

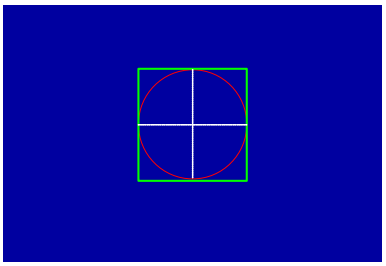

图 14-9　实例 14-5 的运行结果

14.2.6 封闭图形的填充

所谓填充就是用规定的颜色和图模（图形式样）填满一个封闭图形。Turbo C 提供了一些函数，这些函数首先画出一个封闭的轮廓，然后再按设定的颜色和模式进行填充。

1．轮廓填充

Turbo C 提供了一些先画出基本图形轮廓，再按规定图模和颜色填充整个封闭图形的函数。在没有改变填充方式时，Turbo C 将以默认方式填充。下面介绍这些函数。

（1）画条形图函数

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

该函数将以（xl, y1）为左上角，（x2, y2）为右下角画一实形条状图，没有边框，图的颜色和填充模式可以设定。若没有设定，则使用默认模式。

（2）画三维立体直方图函数

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

该函数以（x1, y1）为左上角，（x2, y2）为右下角的矩形框作为底，depth 为深度画出一个三维立体直方图，再按设定或默认模式和颜色填充矩形框。该图第三维（长度为 depth 的那一维）的方向不随任何参数而变，即始终为 45°的方向。当 topflag 非 0 时，画出三维顶。否则将不画出三维顶（实际上很少这样使用），如图 14-10 所示。

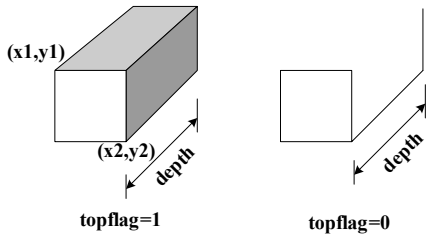

图 14-10　画三维立体直方图函数的参数名定义

（3）画扇形图函数

```
void far pieslice(int x，int y，int stangle，int endangle，int radius);
```

该函数将以（x, y）为圆心，radius 为半径，从 stangle 为起始角，endangle 为结束角，画一扇形图，并以设定或默认模式和颜色填充。当 stang1e 为 0，endangle 为 360 时，则画出一完整的圆。

（4）画椭圆扇形函数

```
viod far sector(int x，int y，int stangle，int endang1e，int xradius，int yradius);
```

该函数将以（x，y）为圆心，以 xradius 和 yradius 为 x 轴和 y 轴半径，从起始角 stang1e 开始到 endang1e 角结束，画一椭圆扇形图，并按设定或默认的填充模式和颜色填充。当 stang1e 为 0，endangle 为 360 时，则画出一完整的椭圆图。

（5）画椭圆图函数

```
void far fillellipse(int x, int y, int xradius, int yradius);
```

该函数将以（x, y）为圆心，以 xradius 和 yradius 为 x 轴和 y 轴半径，画一椭圆图，并以设定或默认模式和颜色填充。

（6）画多边形图函数

```
void far fillpoly(int numpoints, int far *polypoints);
```

该函数将画出一个顶点数为 numpoints，各顶点坐标由 polypoints 给出的多边形，并以设定或默认模式和颜色填充。也即画出边数为 polypoints-1 的多边形并以设定好的或默认的模式和颜色填充它，当为一封闭图形时，numpohts 应为多边形的顶点数加 1，并且第一个顶点坐标应和最后一个顶点坐标相同。

2．设定填充方式

Turbo C 有四个与填充方式有关的函数。下面分别介绍。

（1）填色函数

```
void far setfilestyle(int pattern, int color);
```

该函数将用设定的 color 颜色和 pattern 填充模式对后面画出的轮廓图进行填充，这些图轮廓是由待定函数画出的。color 表示填充色，它的值是当前屏幕图形模式时颜色的有效值。pattern 表示填充模式，可用表 14-9 中的值或符号名表示。

表 14-9　填充模式（pattern）的规定

符 号 名	值	含　义
EMPTY_FILL	0	用背景色填充（空填）
SOLID_FILL	1	用单色实填充（实填）
LINE_FILL	2	用“一”线填充
LTSLASH_FILL	3	用“//”线填充
SLASH_FILL	4	用粗“//”线填充
BKSLASH_FILL	5	用“\\”线填充
LTBKSLASH_FILL	6	用粗“\\”线填充
HATCH_FILL	7	用方网格线填充
XHATCH_FILL	8	用斜网格线填充
INTTERLEAVE_FILL	9	用间隔点填充
WIDE_DOT_FILL	10	用稀疏点填充
CLOSE_DOT_FILL	11	用密集点填充
USER_FILL	12	用户定义样式填充

当 pattern 选用 USER_FILL 用户自定义样式填充时，setfillstyle 函数对填充的模式和颜色不起任何作用，若要选用 USER_FILL 样式填充时，可选用下面的函数。

（2）用户自定义填充函数

```
void far setfillpattern(char *upattefn, int color);
```

该函数设置用户自定义可填充模式，以 color 指出的颜色对封闭图形进行填充。这里的 color 即为图形方式下颜色的规定数值。参数 upattern 是一个指向 8 字节存储区的指针，这 8 字节表示了一个 8×8 像素点阵组成的填充模式，它是由用户自定义的，它将用来对封闭图形填充。8 字节的图形模式是这样形成的：每字节代表一行，而每字节的每一个二进制位代表该行的对应列上的像素。是 1，则用 color 显示，是 0，则不显示。例如：

```
char diamond[8]={0x10,0x38,0x7c,0xfe,0x7c,0x38,0x10,0x00};
```

diamond 为 8 字节的数组，每字节对应于填充模式中的 8 像素，字节中的 1 位，画出一个由 color 设定颜色的像素，字节中的 0 位则不画，实际上，diamond 数组定义了一个 7*7 的小钻石图样，右边和底部都留有一像素的边缘。

（3）得到填充模式和颜色的函数

```
void far getfillsettings(struct fillsettingstype far * fillinfo);
```

该函数用来得到当前的填充模式和颜色，这些信息存在结构指针变量 fillinfo 指向的结构中。其中 fillsettingstype 结构定义如下：

```
struct fillsettingstype
{
    int pattern;  /* 当前填充模式 */
    int color; /* 填充颜色 */
};
```

另外还有一个函数。

```
void far getfillpattern(char *upattern);
```

函数一旦调用，就会把用户自定义的填充模式的 8 字节存入 upattern 所指向的字符数组，该数组必须至少 8 字节长，用户自定义模式以 8 个 8 字节的模式排列，如果还没有调用 setfillpattern()设置用户定义的填充模式，那么函数将 upattern 所指向的数组各元素的值全设置为 0xff。

实例 14-6　图形填充

实例说明	编写一段 C 代码，实现先以不同的填充模式和颜色画矩形图、长方体、扇形和椭圆扇形，然后再自定义一种填充模式和红色画它们。
实例代码	光盘\\chapter14\14-6\
知识要点	封闭图形的填充。

（1）编辑源代码

启动 Turbo C 3.0 编译器，在编辑区中输入代码 14-6 中的代码。

代码 14-6　实例 14-6 的代码

```
01. #include<graphics.h>
02. void main(){
03.     char str[8]={10,20,30,40,50,60,70,80}; /*用户定义填充模式*/
04.     int gdriver,gmode,i;
```

```
05.     struct fillsettingstype save; /*定义一个用来存储填充信息的结构变量*/
06.     gdriver=DETECT;
07.     initgraph(&gdriver,&gmode,"c:\\tc\\bgi");//初始化图形模式
08.     setbkcolor(BLUE); //设置背景色
09.     cleardevice();     //清屏
10.     for(i=0;i<13;i++)
11.     {
12.         setcolor(i+3);
13.         setfillstyle(i,2+i);              //设置填充类型
14.         bar(100,150,200,50);              //画矩形并填充
15.         bar3d(300,100,500,200,70,1);      //画长方体并填充
16.         pieslice(200,300,90,180,90);      //画扇形并填充
17.         sector(500,300,180,270,200,100);//画椭圆扇形并填充
18.         delay(1000);                      //延时 1000 毫秒
19.     }
20.     cleardevice();
21.     setcolor(14); //设置画笔颜色
22.     setfillpattern(str, RED); //自定义填充模式和颜色
23.     bar(100,150,200,50); //画矩形
24.     bar3d(300,100,500,200,70,0); //画长方体
25.     pieslice(200,300,0,360,90); //画圆
26.     sector(500,300,0,360,100,50); //画椭圆
27.     getch();
28.     getfillsettings(&save);       //获得当前的填充模式和填充颜色
29.     closegraph();
30.     clrscr();
31.     /*输出当前填充模式和填充颜色*/
32.     printf("The pattern is %d, The color of filling is %d", save.pattern,
save.color);
33.     getch();
34. }
```

代码执行步骤如下。

① 代码第 3 行定义了一个含 8 个元素的字符数组，该数组将在代码的后面用来作为用户自定义的填充模式。

② 代码第 4、5 行定义了三个整型变量和一个 struct fillsettingstype 结构体变量。

③ 代码第 6、7 行初始化图形模式。

④ 代码第 8、9 行设置背景色为蓝色，然后清屏。

⑤ 代码第 10 到第 19 行是使用 for 循环语句实现以不同的作图色、填充模式和填充颜色画各种图。首次执行代码第 10 行设变量 i 的初值为 0，然后判断 i 的值是否小于 13，若是则执行代码第 12 行到第 18 行。代码第 12 行设置当前画笔颜色为 i+3=3，即青色；代码第 13 行设置当前填充模式为 i（即 0，表示用背景色填充），填充颜色为 2+i=2，即绿色，（实际上，这里是空填，填充色绿色没有作用）；代码第 14 行是画一个矩形图并以设定的模式和颜色填充；代码第 15 行是画一个长方体并以设定的模式和颜色填充；代码第 16 行是画一个扇形并以设定的模式和颜色填充；代码第 17 行画一个椭圆扇形并以设定的模式和颜色填充；代码第 18 行调用 delay 函数延时 1 000 毫秒；然后将变量 i 的值加 1，继续下一次循环，直到 i 的值增加到 13

为止。

编者手记

这里请读者注意：setcolor 函数设置的颜色实际上就是各图形边框的颜色，但除了 bar 函数画的矩形的边框。因为用 bar 函数画的矩形无边框。

⑥ 代码第 20、21、22 行是先清屏，然后设置画笔颜色，接着自定义填充模式和颜色。

⑦ 代码第 23 行到第 26 行是先以自定义填充模式和颜色画一矩形；接着以自定义填充模式和颜色画一无顶的长方体；接着以自定义填充模式和颜色画一圆；接着以自定义填充模式和颜色画一椭圆。

⑧ 代码第 27、28 行是调用 getch 函数接收一个字符，然后调用 getfillsettings 获得当前填充模式和填充颜色。

⑨ 代码第 29、30 行是退出图形模式进入文本模式并清屏。

⑩ 代码第 32 行输出当前填充模式和填充颜色。

（2）保存源文件

按【F2】键，在弹出的保存文件窗口中，选择或输入文件保存到的目录并修改该文件名为 Ex14-6.c。

（3）编译、链接、运行程序

按【F9】键，编译并链接代码 14-6，然后按【Ctrl+F9】快捷键运行该程序。便会看到屏幕上出现四个图形，一个矩形（无边框）、一个长方体、一个扇形、一个椭圆扇形，而且它们的填充和边框颜色在变化（矩形只有填充色在变化）。最后出现以用户定义填充模式、填充色为红色、边框为黄色的一个矩形、一个无顶长方体、一个圆、一个椭圆。

运行结束后按任意键，则在屏幕上显示出现行填充模式和颜色的常数值。

3．任意封闭图形的填充

前面介绍的填充函数，只能对由上述特定函数产生的图形进行颜色填充，但还不能对任意封闭图形进行填充。为此，Turbo C 提供了一个可对任意封闭图形填充的函数，其原型为：

```
void far floodfill(int x, int y, int border);
```

该函数将对一封闭图形进行填充，其颜色和模式将由设定的或默认的填充模式与颜色决定。其中参数（x, y）为封闭图形中的任一点，border 是封闭图形的边框颜色，也就是封闭图形轮廓的颜色。编程时该函数位于画图形的函数之后，即要填充该图形。调用了该函数后，将用规定的颜色和填充模式填满整个封闭图形。

另外，需要注意的是：

（1）若（x, y）点位于封闭图形边界上，该函数将不进行填充。

（2）若对不是封闭的图形进行填充，则会填到别的地方，即会溢出。

（3）若（x, y）点在封闭图形之外，将对封闭图形外进行填充。

（4）由参数 border 指出的颜色必须与封闭图形的轮廓线的颜色一致，否则会填到别的地方去。

14.2.7 有关图形窗口和图形屏幕操作函数

1. 图形窗口操作函数

像文本方式下可以设定屏幕窗口一样，图形方式下也可以在屏幕上某一区域设定窗口，只是设定的为图形窗口而已，下面就来介绍关于图形窗口的相关函数。

（1）图形窗口设置函数

在图形方式下可以在屏幕上某一区域设置一个窗口，这样以后的画图操作均在这个窗口内进行，且以这个窗口的左上角（0, 0）作为坐标原点，而不再用物理屏幕坐标（屏左角为（0, 0）点）。在图形窗口内画的图形将显示出来，超出图形窗口的部分可以不让其显示出来，也可以让其显示出来（不剪断），该函数原型说明为：

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

其中（x1, y1）为图形窗口的左上角坐标，（x2, y2）为所设置的图形窗口右下角坐标，它们都是以原屏幕物理坐标为参考的。clipflag 参数若为非 0，则所画图形超出图形窗口的部分将被切除而不显示出来。若 clipflag 为 0，则超出图形窗口的图形部分仍将显示出来。

（1）图形窗口的清除与取信息函数

① 图形窗口清除函数

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

该函数将清除图形窗口内的图像。

② 取图形窗口信息函数

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

该函数将取得当前设置的图形窗口的信息，它存于由结构 viewporttype 定义的结构变量 viewport 中，结构 viewporttype 定义如下：

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

使用图形窗口设置函数 setviewport，可以在屏上设置不同的图形窗口，甚至部分可以重叠，但注意最近一次设置的窗口才是当前窗口，后面的图形操作都视为在此窗口中进行，其他窗口均无效。若不清除那些窗口的内容，则它们仍在屏上保持，当要对它们处理时，可再一次设置那个窗口一次，这样它就又变成当前窗口了。

使用 setbkcolor 设置背景色时，对整个屏幕背景起作用，它不是只改变图形窗口内的背景。在用 setcolor 设置前景色时，它对图形窗口内画图起作用。若下一次设置图形窗口没有设置颜色，那么上次在另一图形窗口内设置的颜色在本次设置的图形窗口内仍起作用。

2．图形屏幕操作

对图形屏幕操作的函数，还有一类是设置显示页函数。在图形方式下存储在 VRAM（显示适配器的显示存储器）中的一满屏图像信息称为一页。每个页一般为 64K 字节，VRAM 可以存储要显示的图像几个页（视 VRAM 容量而定，最大可达 8 页），Turbo C 支持页的功能有限，按在图形方式下显示的模式最多支持 4 页（EGALO 显示方式），一般为两页（注意对 CGA，仅有一页），因存储图像的页显示时，一次只能显示一页，因此必须设定某页为当前显示的页（又称可视页），默认时定为 0 页。

正在由用户编辑图形的页称为当前编辑页（又称激活的页），这个页不等于显示页，即若用户不设定该页为当前显示页时，在该页上编辑的图形将不会在屏上显示出来。默认时，设定 0 页为当前编辑页，即若不用下述的页设置函数进行设置，就认定 0 页既是编辑页，又是当前显示页。

设置激活页和显示页的函数如下：

```
void far setactivepage(int pagenum);
void far setvisualpage(int pagenum);
```

这两个函数只能用于 EGA、VGA 等显示适配器。前者设置由 pagenum 指出的页为激活的页，后者设置可显示的页。当设定了激活的页，即编辑页后，则程序中其后的画图操作均在该页进行，若它不定为显示页，则其上的图像信息并不会在屏上显示出来。关于图形模式和可以设定的页数，可参阅表 14-4。

```
void far getimage(int xl,int yl, int x2,int y2, void far *mapbuf);
void far putimge(int x,int,y,void * mapbuf, int op);
unsined far imagesize(int xl,int yl,int x2,int y2);
```

这三个函数用于将屏幕上的图像复制到内存，然后再将内存中图像送回到屏幕上。首先通过函数 imagesize()测试要保存左上角为（xl, yl），右上角为（x2, y2）的图形屏幕区域内的全部内容需多少字节，然后再给 mapbuf 分配一个所测数字节内存空间的指针。通过调用 getimage()函数就可将该区域内的图像保存在内存中，需要时可用 putimage()函数将该图像输出到左上角为点（x, y）位置上，其中 getimage()函数中参数 op 规定如何释放内存中图像。关于这个参数的定义参见表 14-10。

表 14-10　putimage()函数中的 op 值

符号常数	数　值	含　义
COPY_PUT	0	复制
XOR_PUT	1	与屏幕图像异或的复制
OR_PUT	2	与屏幕图像或后的复制
AND_PUT	3	与屏幕图像与后的复制
NOT_PUT	4	复制反像的图形

对于 imagesize()函数，只能返回字节数小于 64K 字节的图像区域，否则将会出错，出错时返回-1。

14.2.8 图形方式下的文本输出函数

在图形方式下，虽然也可以用 printf()，puts()，putchar()函数输出文本，但只能在屏上用白色显示，无法选择输出的颜色，尤其想在屏上定位输出文本，更是困难，且输出格式也是不能变的 80 列×25 行形式。Turbo C 提供了一些专门用在图形方式下的文本输出函数，它们可以用来选择输出位置、输出字型和大小、输出方向等。

1．文本输出函数

（1）当前位置文本输出函数

```
void far outtext(char far *textstring);
```

该函数将在图形屏幕的当前位置上输出由字符串指针 textsering 指出的文本字符串。该函数没有定位参数，只能在当前位置输出字符串。

（2）定位文本输出函数

```
void far outtextxy(int x, int y, char far *textstring);
```

该函数将在指定的（x, y）位置输出字符串。（x, y）位置如何确定，还需要用位置确定函数 settextjustify()来确定；选用何种字形显示、字体大小及横向或纵向显示，还需用 settextstyle()函数来确定（settextjustify 函数和 settextstyle 函数随后将会介绍）。这些均要在文本输出函数之前确定。若没有使用函数确定，则输出来用默认方式。例如，当执行函数 outtextxy(10, 10, “Turbo C”)时，Turbo C 将采用默认方式显示在如图 14-11 所示位置，其“T”字的左上角位置为（10, 10），字形为 8×8 点阵(8 像素宽，8 像素高)，尺寸 1:1，即和字库中的字同大。

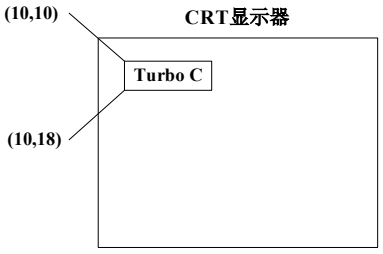

图 14-11　outtextxy(10, 10, “Turbo c”)输出

2．有关文本字体、字型和输出方式的设置函数

有关图形方式下的文本输出函数，可以通过 setcolor()函数设置输出文本的颜色。另外，也可以改变文本字体大小以及选择是水平方向输出还是垂直方向输出。

（1）文本输出位置函数

```
void far settextjustify(int horiz, int vert);
```

该函数将确定输出字符串时，如何定位（x, y）。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、中部和底部三个位置，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()函数的第一个参数 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屏幕的当前位置。有关参数 horiz 和

vert 的取值和相应的符号名如表 14-11 所示。

表 14-11　参数 horiz、vert 的取值

参数 horiz			参数 vert		
符 号 名	值	含　义	符 号 名	值	含　义
LEFT_TEXT	0	左对齐	BOTTOM_TEXT	0	底部对齐
CENTER_TEXT	1	水平方向中心对齐	CENTER_TEXT	1	垂直方向中心对齐
RIGHT_TEXT	2	右对齐	TOP_TEXT	2	顶部对齐

若 horiz 取 LEFT_TEXT（或取 0），则（x, y）点是以输出的第一个字符的左边为开始位置，即（x, y）定位于此。但是以第一个字符左边的顶部、中部，还是底部定位（x, y），还不能确定，也就是说，（x, y）点是指输出字符串第一个字符的左边位置，但是在第一个字符左边的垂直方向上的位置还需由 vert 参数决定。如 vert 取 TOP_TEXT（即 2），则（x, y）在垂直方向定位于第一个字符左边位置的顶部。

（2）定义文本字型函数

```
void far settextstyle(int font, int direction, int char size);
```

该函数用来设置文本输出的字形、方向和大小，其相应参数 font、参数 direction 和参数 size 的取值如表 14-12 所示。

表 14-12　font、direction、size 的取值

	符 号 名	值	含　义
font	DEFAULT_FONT	0	8×8 字符点阵（默认值）
	TRIPLEX_FONT	1	三倍笔划体字
	SMALL_FONT	2	小字笔划体字
	SANS_SERIF_FONT	3	无衬线笔划体字
	GOTHIC_FONT	4	黑体笔划体字
direction	符 号 名	值	含　义
	HORIZ_DIR	0	水平输出
	VERT_DIR	1	垂直输出
size		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	用户自定义字符大小

（3）用户对文本字符大小的设置

前面介绍的 settextstyle()函数，可以设定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*8 点阵字以内的字体），只能在水平和垂直方向以相同的放大倍数放大。为此 Turbo C 2.0 又提供了另外一个 setusercharsize()函数，对笔划字体可以分别设置水平和垂直方向的放大倍数。该函数的调用格式为：

```
void far setusercharsize(int mulx, int divx, int muly, int divy);
```

该函数用来设置笔划型字和放大系数，它只有在 settextstyle()函数中的 size 参数为 0（或 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默认大小乘以 mulx/divx 为输出字符宽，乘以 muly/divy 为输出字符高。

3．文本输出字符串函数

该函数原型为：

```
int sprintf(char *string，char *format[, argument，…]);
```

该函数将把变量值 argument，按 format 指定的格式，输出到由指针 string 指定的字符串中去，该字符串就代表了其输出。这个函数虽然不是图形专用函数，但它在图形方式下的文本输出中很有用，因为用 outtext()或 outtextxy()函数输出时，输出量是文本字符串，当我们要输出数值时不太方便。因而可用 sprintf 函数将数值输出到一个字符数组中，再让文本输出函数输出这个字符数组中的字符串即可。

实例 14-7　文本输出

实例说明	编写一段 C 代码，实现在图形模式下输出不同样式的文本。
实例代码	光盘\\chapter14\14-7\
知识要点	图形方式下的文本输出函数的使用。

（1）编辑源代码

启动 Turbo C 3.0 编译器，在编辑区中输入代码 14-7 中的代码。

代码 14-7　实例 14-7 的代码

```
01. #include <graphics.h>
02. void main()
03. {
04.     int i, graphdriver,graphmode;
05.     char s[30];
06.     graphdriver=DETECT;
07.     initgraph(&graphdriver,&graphmode,"c:\\tc\\bgi");
08.     cleardevice();
09.     setbkcolor(BLUE);//设置背景色
10.     setviewport(40,40,600,440,1); // 定义图形窗口
11.     setfillstyle(1,2);//以绿色实填
12.     setcolor(YELLOW);//设置画笔颜色为黄色
```

```
13.     rectangle(0,0,560,400);//画一矩形
14.     floodfill(50,50,14); //用绿色填画出的矩形框
15.     rectangle(20,20,540,380);//画一矩形
16.     setfillstyle(1,13);//设置以淡洋红色实填
17.     floodfill(19,19,14); //用淡洋红色填画出的矩形框
18.     setcolor(15);//设置画笔颜色为白色
19.     settextstyle(1,0,6); // 设要显示字符串的字型、方向和尺寸
20.     outtextxy(100,60,"Welcome Your");//输出字符串
21.     setviewport(100,200,540,380,0); //又定义一个图形窗口
22.     setcolor(14);//设置画笔颜色为黄色
23.     setfillstyle(1,12);//设置以淡红色实填
24.     rectangle(20,20,420,120);//画一矩形
25.     floodfill(21,100,14); //用淡红色填充
26.     settextstyle(2,0,9);
27.     i=620;
28.     sprintf(s, "Your score is %d", i); //将数字转化为字符串
29.     setcolor(YELLOW);
30.     outtextxy(60,40, s); /* 用黄色显示字符串 */
31.     setcolor(1);//设置画笔颜色为蓝色
32.     settextstyle(3, 0, 0);//设置输出的字符大小由用户自定义
33.     setusercharsize(4, 1, 1, 1);//自定义文本字符大小
34.     outtextxy(70, 80, "Good");//显示字符串
35.     getch();
36.     closegraph();
37. }
```

代码执行步骤如下。

① 代码第 4 行到第 8 行定义相应的变量和数组，然后初始化图形模式并清屏。

② 代码第 9、10 行设置图形屏幕的背景色为蓝色，然后定义一个图形窗口。

③ 代码第 11 到第 14 行在该图形窗口中先设置填充模式和填充颜色，即以绿色实填；接着以黄色为作图色，以（40, 40）为左上角坐标，以（560, 400）为右下角坐标画一矩形；然后用填充颜色（绿色）实填边框为黄色的封闭图形，即实填矩形。

④ 代码第 15、16、17 行是先以（20, 20）为左上角，以（540, 380）为右下角坐标画一矩形；然后设置以淡洋红色实填该矩形和上一个矩形中间的地方。

⑤ 代码第 18、19、20 行是设置画笔颜色为白色，然后设置要显示的字符串的字型、方向和大小；接着在当前图形窗口的坐标（100, 60）处输出字符串。

⑥ 代码第 21 行到第 25 行是先在当前图形窗口中定义一个图形窗口；接着设置作图色、填充模式和填充颜色；然后以该图形窗口中（20, 20）为矩形的左上角坐标，以其（420, 120）为矩形的右下角坐标画一个矩形；然后用填充颜色填充矩形。

⑦ 代码第 26 行到第 30 行是先设置要显示字符串的字型、方向和尺寸；接着为整型变量 i 赋值，并用 sprintf 函数将该整型变量和相关的字符以字符串形式存储字符数组 s 中；然后用黄色以设定的形式显示该字符串。

⑧ 代码第 31 行到第 34 行是先设置画笔颜色为蓝色，接着设置要显示字符的样式有用户自定义；然后调用 setusercharsize 函数自定义要显示的字符样式；最后显示字符串。

⑨ 代码第 35、36 行是接收用户输入的一个字符，然后退出图形模式。

（2）保存源文件

按【F2】键，在弹出的保存文件窗口中，选择或输入文件保存到的目录并修改该文件名为 Ex14-7.c。

（3）编译、链接、运行程序

按【F9】键，编译并链接代码 14-7，然后按【Ctrl+F9】快捷键运行该程序。结果如图 14-12 所示。

图 14-12　实例 14-7 的运行结果

14.3　菜单设计

所谓菜单，顾名思义，就是像饭店里的菜单，由顾客根据菜单上罗列的菜单名来进行点菜，然后店家根据顾客点的菜，进行加工制作，最后送到顾客面前，计算机的菜单也如同上述菜单的作用，因一个实用软件，往往有多种操作功能，可完成多种任务，因此为了方便顾客，在程序开始运行时，首先在屏幕上开一个窗口，显示许多有关的信息，这些信息分解成若干项，这就是通常说的菜单窗口，这些信息项就叫做菜单项，用户可以通过光标移动键，或其他键，选择合适的菜单项，当再一次确认后，程序便执行相应项的功能。一个高质量的菜单，如同一张彩色美丽的画面，不仅使人视觉上有美的感受，更重要的是给用户使用软件带来方便，使得操作简化，避免了误操作带来的不良后果，且能迅速得到帮助信息，Turbo C 运行时，屏幕上出现的图像，就是一个典型的菜单，它提示了在 Turbo C 集成环境下，各种选择项，由用户选择和确认，并由 Turbo C 执行。

菜单式样有长条形、矩形。在屏上出现方向有纵向、横向、屏幕滚动（菜单较长，一屏显示不完），如图 14-13 所示。

由于显示器有文本显示方式和图形显示方式之分，因而菜单也有文本菜单方式下制作的菜单和图形方式下制作的菜单。就菜单生成的方式而言，又有固定式菜单、弹出式菜单和下拉式菜单之分。

固定式菜单，就是程序运行一开始就出现在屏幕上的一种菜单，它仅存在一次，或始终停留在画面上，或功能选择完成后就消失了，不再出现。这种画面，生成简单。

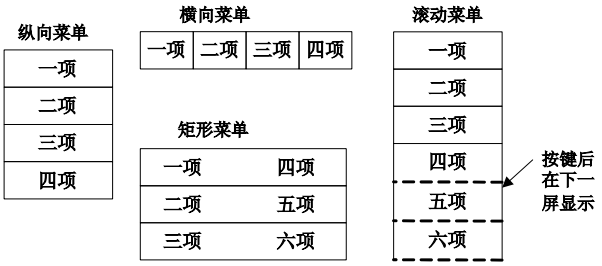

图 14-13　菜单样式

所谓弹出式菜单，实际上就是一种动态变化的菜单，如出现菜单后，当选中某项后，菜单立即消失，出现选中菜单项的功能程序运行。若又需要另外和用户的接口功能，则又弹出一个菜单，供用户选择，一般来说，弹出式菜单仅用在一级深度的选择中，即在该菜单中选中某菜单项后，无需再进行第二次选择，也就是说，选中的该菜单项再不会带有子项要选择。

下拉式菜单则不同于弹出式菜单，它是在选中菜单中某项后，接着在其下面又出现选中项的子项菜单，又要再一次进行选择，也许还要再深度地进行选择，几个下拉式菜单可以同时出现在屏幕上，Turbo C 开始时呈现在用户面前的菜单，就是一种典型的下拉式菜单，这个菜单反映了 Turbo C 集编辑、编译、链接、执行为一体的一个集成化的环境，首先屏幕上显示一横向主菜单，当选中某一项后，就在此项下面又出现一纵向子菜单。可以用光标移动键向下移动光条，来选择该项的子项，这个菜单就称下拉式菜单。

14.3.1　菜单设计要点

菜单设计一般要有如下几个部分。

1．菜单窗口图像的存储和重放

当弹出一个窗口时，原窗口区域内的内容就被窗口覆盖了。为了在弹出的窗口消失后，能恢复原被覆盖的内容，必须事先将原来的内容保存起来，这可以用 getimage()函数，保存区域所占字节数，可以用 imagesize()函数来测试，并由 malloc()函数来根据测得的字节数分配显示存储器的缓冲区。

当要重现被覆盖的内容时，可以用 putimage()函数将保存在缓冲区的原图像通过 OP 操作（一般用 COPY_PUT）重现在原区域。

在文本下，可以使用 gettext 函数和 puttext 函数存放某一缓冲区的文本，而存放的缓冲区，则由区域所占的行数和列数乘 2 来决定，这可事先定义一个字符数组来实现，如：

```
char buf[行数*列数*2];
```

2．菜单窗口和菜单项的生成

按照用户的按键生成一个相应的设计好的菜单图像，并将实现存放在字符指针数组中的菜单各项内容，填入相应的图像位置中，并用颜色（一般是红色）标出相应项选择对应的热键（即选择该项时，按下的键）。

3．生成光条

在菜单项上要压上光条，用户按【Up】或【Down】键，使该光条在整个菜单项上移动，以标明要选择的菜单项，当按回车键或热键后，相应菜单项功能被实现，这可能通过对相应菜单项的图像存取和菜单项图像改变背景色后的重放来实现光条。

4．键识别

当按下菜单项所示的功能键（或称热键）或光标移动键时，要得到这些键的键盘扫描码，才能得知何键按下。可设计一个键分析程序将扫描码返回，菜单根据键值，转入相应的功能处理。可采用 DOS 的 int86()功能调用。也可用键盘操作函数 bioskey()来得到键的扫描码。

5．功能执行

菜单中应根据用户的菜单项选择，转入相应的功能程序去执行，这个部分可以嵌入到菜单程序中去，对较大功能程序，可以做成功能模块，在菜单中由键分析程序，根据按键直接调用。

为了让读者更好地理解菜单设计，下面是分别在文本模式下（实例 14-8）和图形模式下的菜单设计（实例 14-9）。

实例 14-8　菜单程序（一）

实例说明	编写一段 C 代码，实现在文本方式下产生一个下拉式菜单，程序运行时首先在屏幕顶行产生一个浅灰底黑字的主菜单，各菜单项的第一个字母加红，表示为热键，当选择主菜单第一项，即按【Alt+F】快捷键时，便产生一个下拉式子菜单，可用【Up】和【Down】键使压在第一个子菜单项上的黑色光条上下移动，当压在某子菜单项上，且按回车后，程序便转去执行相应子菜单项的内容。由于篇幅关系，这里只做了第一个主菜单项和对应的子菜单，且子菜单项对应的操作只在程序相应处做了说明，并无具体内容，对主菜单项其他各项未做选它时相应的子菜单，但作法和第一项 File 的相同，故不赘述。
实例代码	光盘\\chapter14\14-8\
知识要点	菜单设计。

（1）编辑源代码

启动 Turbo C 3.0 编译器，在编辑区中输入代码 14-8 的代码。

代码 14-8　实例 14-8 的代码

```
01. #include <dos.h>
02. #include <conio.h>
03. #define Key_DOWN 0x5100  //Down 键的键盘扫描码
04. #define Key_UP 0x4900    //Up 键的键盘扫描码
05. #define Key_ESC 0x011b   //Esc 键的键盘扫描码
06. #define Key_ALT_F 0x2100 //Alt+F 键的键盘扫描码
07. #define Key_ALT_X 0x2d00 //Alt+X 键的键盘扫描码
08. #define Key_ENTER 0x1c0d //Enter 键的键盘扫描码
```

```
09. main()
10. {
11.     int i,key,x,y,l;
12.     char *menu[] = {"File","Edit","Run","Option","Help","Setup","Zoom",
"Menu"};
13.     /* 主菜单各项 */
14.     char *red[] = { "F","E","R","O","H","S","Z","M" }; /* 加上红色热键 */
15.     char *f[] = {"Load file", "Save file", "Print", "Modify ", "Quit Alt_x"};
16.     /* File 项的子菜单 */
17.     char buf[16*10*2],buf1[16*2]; /* 定义保存文本的缓冲区 */
18.     while(1)
19.     {
20.         textbackground(BLUE);//设置文本显示的背景颜色
21.         clrscr();
22.         textmode(C80);//设置文本显示方式
23.         window(1,1,80,1);//定义显示主菜单的窗口
24.         textbackground(LIGHTGRAY);
25.         textcolor(BLACK);//设置前景色
26.         clrscr();
27.         gotoxy(5,1);//坐标定位
28.         for(i=0,l=0;i<8;i++)
29.         {
30.             x=wherex(); /* 得到当前光标的坐标 */
31.             y=wherey();
32.             cprintf("%s",menu[i]); /* 显示各菜单项 */
33.             l=strlen(menu[i]); /* 得到菜单项的长度 */
34.             gotoxy(x,y);
35.             textcolor(RED);
36.             cprintf("%s",red[i]); /* 在主菜单项各头字符写上红字符 */
37.             x=x+l+5;
38.             gotoxy(x,y);
39.             textcolor(BLACK); /* 为显示下一个菜单项移动光标 */
40.         }
41.         gotoxy(5,1);
42.         key=bioskey(0);
43.         switch (key){
44.             case Key_ALT_X:
45.                 exit(0); /* ALT_X 则退出 */
46.             case Key_ALT_F:
47.             {
48.                 textbackground(BLACK);
49.                 textcolor(WHITE);
50.                 gotoxy(5,1);
51.                 cprintf("%s",menu[0]); /* 加黑 File 项 */
52.                 gettext(5,2,20,12,buf); /* 保存窗口原来的文本 */
53.                 window(5,2,20,9);/* 设置做矩形框的窗口 */
54.                 textbackground(LIGHTGRAY);
55.                 textcolor(BLACK);
56.                 clrscr();
57.                 for(i=2;i<7;i++) /* 显示子菜单各项 */
58.                 {   gotoxy(2,i);
59.                     cprintf("%s",f[i-2]);
60.                 }
```

```
61.                 gettext(2,2,18,3,buf1); /*将下拉菜单的内容保存在 buf1*/
62.                 textbackground(BLACK);
63.                 textcolor(WHITE);
64.                 gotoxy(2,2);
65.                 cprintf("%s",f[0]);/*加黑下拉菜单的第一项 load file*/
66.                 gotoxy(2,2);
67.                 y=2;
68.                 while ((key=bioskey(0))!=Key_ALT_X) // 等待选择下拉菜单项
69.                 {
70.                     if ((key==Key_UP)||(key==Key_DOWN))
71.                     {
72.                         puttext(2,y,18,y+1,buf1); // 恢复原先的项
73.                         if(key==Key_UP)
74.                             y=(y==2?6:y-1);
75.                         else
76.                             y=(y==6?2:y+1);
77.                         gettext(2,y,18,y+1,buf1);//保存要压上光条的子菜单项
78.                         textbackground(BLACK);
79.                         textcolor(WHITE);
80.                         gotoxy(2,y);
81.                         cprintf("%s",f[y-2]); /* 产生黑条压在所选项上 */
82.                         gotoxy(2,y);
83.                     }
84.                     else
85.                     //若是回车键，判断是哪一子菜单按的回车，在此没有相应的特殊处理
86.                         if (key==Key_ENTER)
87.                         {
88.                             switch ( y-1 ){
89.                                 case 1: /* 是子菜单项第一项:Load file */
90.                                     break;
91.                                 case 2: /* Save file */
92.                                     break;
93.                                 case 3: /* print */
94.                                     break;
95.                                 case 4: /* modify */
96.                                     break;
97.                                 case 5:
98.                                     exit(0);
99.                                 default:
100.                                        break;
101.                            }
102.                            break;
103.                        }
104.                        else
105.                            if (key==Key_ESC)
106.                                break; //是 Esc 键,返回主菜单
107.                    }
108.                    if (key==Key_ALT_X) exit(0);
109.
110.                }
111.            }
112.        }
113.    }
```

代码执行步骤如下。

① 代码第 1 行到第 8 行定义了相关的文件包含命令，以及相关键的宏定义。

② 代码第 11 行到第 17 行定义了相关的变量以及相关的字符数组。

③ 代码第 18 行是 while 循环语句，其循环体为代码第 19 行到第 112 行。

④ 代码第 20 行到第 27 行先是设置文本显示的背景颜色为蓝色并清屏；接着设置文本显示方式为 C80；然后定义了一个显示主菜单项的窗口，并定义该窗口中文本显示的背景颜色为淡灰色，前景色为黑色；之后清屏；然后将当前光标定位在坐标（5, 1）。

⑤ 代码第 28 行到第 39 行是 for 循环。首次执行到代码第 28 行时先是将变量 i 和 l 的初始值赋为 0，然后判断变量 i 的值是否小于 8，若是则执行代码第 30 行到第 39 行。

代码第 30、31 调用 wherex()函数和 wherey()函数获得当前光标的坐标 x 和 y。代码第 32 行调用 cprintf 函数输出字符数组 menu 中的第 i 个元素（i 从 0 开始）。代码第 33 行调用获得字符数组 menu 中的第 i 个元素的字符长度存储在变量 l 中。代码第 34 行是将光标重新定位到坐标（x, y）处。代码第 35 行是设置当前作图色为红色。代码第 36 行是以红色输出字符输出 red 的第 i 个元素。代码第 37 行是将 x+l+5 的值赋给变量 x。代码第 38 行将当前光标的值定位到坐标（x, y）处（注意此时的 x 的值为 x+l+5）。代码第 40 行是设置当前作图色为黑色。接着将变量 i 的值加 1，然后再继续下一次循环，直到变量 i 的值增加到 8，则执行该循环下面的语句。代码第 41 行将当前光标定位在坐标（5, 1）处。

⑥ 代码第 42 行调用 bioskey(0)获得用户当前的所按键的键盘扫描码赋给变量 key。接着在代码第 43 行判断 key 的值。若 key 等于 Key_ALT_X（即用户按下了【Alt+X】快捷键），则执行代码第 45 行退出本程序。若 key 等于 Key_ALT_F（即用户按下了【Alt+F】快捷键），则执行代码第 48 行到第 110 行。

⑦ 代码第 48 行到第 53 行先定义文本显示的背景颜色为黑色，前景色为白色；接着将当前光标定位在坐标（5, 1）处，并输出字符数组的第 0 个元素值；接着将左上角坐标为（5, 2）和右下角坐标为（20, 12）的矩形区域内的文本保存在字符数组 buf 中；然后设置一个左上角坐标为（5, 2）和右下角坐标为（20, 9）的矩形窗口。

⑧ 代码第 54 行到第 67 行是先设置当前文本显示的背景颜色为淡灰色，前景色为黑色并清屏；接着用 for 循环一次显示子菜单项；然后将左上角坐标为（2, 2）和右下角坐标为（18, 3）的矩形区域内的文本保存在字符数组 buf1 中；下面设置当前文本显示的背景颜色为黑色，前景色为白色；之后将当前光标定位在坐标（2, 2）处，再下面输出下拉菜单的第一项 load file，并再次将当前光标定位在坐标（2, 2）处。接着将变量 y 的值赋为 2。

⑨ 代码第 68 行到第 107 行是 while 循环，该循环先是判断当前用户输入的键是否等于【Alt+X】快捷键，若不是则执行代码第 70 行到第 106 行。代码第 70 行是判断当前用户输入的键（key 的值）是否为【Up】键或【Down】键，若是则执行代码第 72 行到第 82 行，否则执行代码第 86 行到第 106 行。

代码第 72 行是将 buf1 内存储的文本重现显示在其原来的位置。代码第 73 行是判断 key 是否为【Up】键，若是则执行代码第 74 行判断 y 的值是否等于 2，若是将 y 的值赋为 6，否则将 y 的值减 1 再赋给变量 y；若 key 不为【Up】键，则执行代码第 76 行，判断 y 的值是否

等于 6，若是则将 y 的值赋为 2，否则将 y 的值加 1 再赋给变量 y。代码第 77 行是保存要压上光条的子菜单项。代码第 78、79 行是设置当前文本显示的背景色为黑色，前景色为白色。代码第 80 行到第 82 行是将当前光标定位在（2, y）处，并输出相应的子菜单，然后再将当前光标重新定位在（2, y）处。

代码第 86 行是判断 key 是否为【Enter】键，若是则执行代码第 88 行到第 102 行，否则执行代码第 105、106 行。第 88 行到第 102 行代码实际上是各下拉子菜单项要执行的内容，但由于篇幅的原因，这里没有编写，感兴趣的读者可以自己添加一些内容。代码第 105 行是判断 key 是否为【Esc】键，若是则执行代码第 106 行退出本循环。

⑩ 代码第 108 行是判断 key 是否为【Alt+X】快捷键，若是则退出本程序。

（2）保存源文件

按【F2】键，在弹出的保存文件窗口中，选择或输入文件保存到的目录并修改该文件名为 Ex14-8.c。

（3）编译、链接、运行程序

按【F9】键，编译并链接代码 14-8，然后按【Ctrl+F9】快捷键运行该程序。结果如图 14-14 所示。

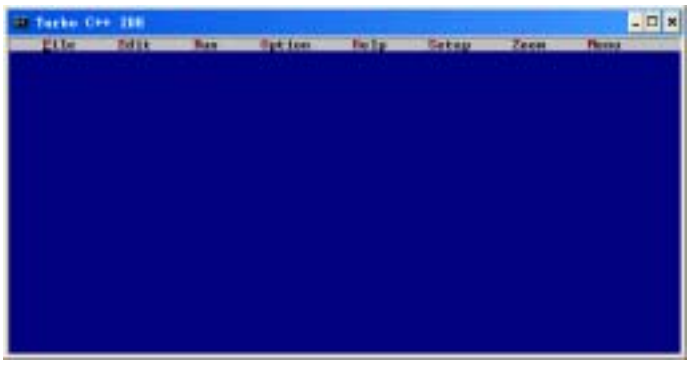

图 14-14　实例 14-8 的运行结果

接着，按【Alt+X】快捷键，则菜单【File】便产生一个下拉式子菜单，可用【Up】和【Down】键使压在第一个子菜单项上的黑色光条上下移动。如图 14-15 所示。

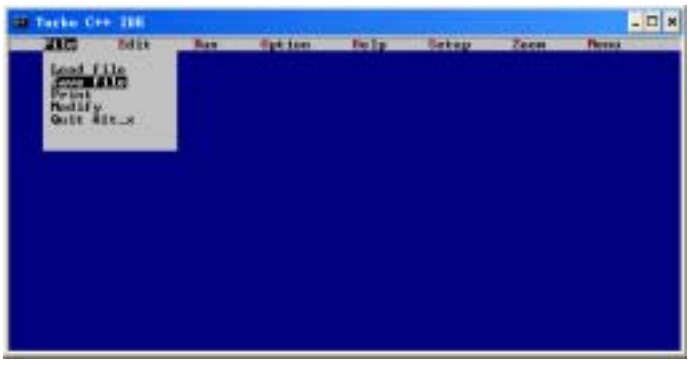

图 14-15　下拉菜单

实例 14-9　菜单程序（二）

实例说明	编写一段 C 代码，实现一个下拉式弹出菜单，该菜单在图形方式下使用，在该菜单中，主要菜单项用“Menu0”、“Menu1”和“Menu2”来表示。当主菜单名称显示红色，则表示该菜单项被选中，并且可通过左右键（←→）来移动选择菜单项。当选择某个主菜单项后，按回车键，则出现该菜单项的子菜单，可通过上下键（↑ ↓）来移动选择子菜单项。程序可按【Esc】键弹出子菜单（当有子菜单时），也可按【Esc】键或【Alt+X】快捷键退出程序。
实例代码	光盘\\chapter14\14-9\
知识要点	菜单设计。

（1）编辑源代码

启动 Turbo C 3.0 编译器，在编辑区中输入代码。首先，输入如下代码：

```
#include <graphics.h>
#include <stdio.h>
#include <stdlib.h>
#include <bios.h>

#define MenuNum 3
#define FALSE 0
#define TRUE 1
#define START 1
#define LEFTSHIFT 2
#define RIGHTSHIFT 3
#define ENTER 4
#define ESC 5
#define UP 6
#define DOWN 7
#define ALTX 8
```

该代码是本程序的包含文件命令和相关的宏定义。

接着，定义相关的全局变量，以及初始化图形模式函数，如代码 14-9 所示。

代码 14-9　定义相关的全局变量和初始化图形模式

```
01. typedef struct{              /*菜单的数据结构*/
02.     int menuID;
03.     char MenuName[8];
04.     int itemCount;
05.     char itemName[4][8];
06. }menu;
07. void *saveImage;       /*保存菜单覆盖的区域*/
08. int mHeight,mWidth;          /*窗口高,宽*/
09. int mutex=0;
10. menu MainMenu[]={{0,"Menu0",4,{"Open","New","Save","Exit"}},{1,"Menu1",
2,{"Copy","Paste"}},{2,"Menu2",2,{"Find","Instead"}}};
```

```
11. void init()//初始化图形模式
12. {
13.     int gdriver,gmode;
14.     gdriver=DETECT;
15.     initgraph(&gdriver,&gmode,"c:\\tc\\bgi");
16. }
```

代码执行步骤如下。

① 代码第 1 行到第 6 行定义了一个结构体类型，别名为 menu。

② 代码第 7 行定义了一个 void 类型的指针变量 saveImage，该指针所指向的区域将用来保存菜单覆盖的区域。

③ 代码第 8、9 行定义了三个整型变量，mHeight 和 mWidth 分别用来表示窗口的高和宽，而 mutex 用来表示当前菜单项是主菜单还是子菜单，为 0 时表示主菜单，为 1 时表示子菜单。

④ 代码第 10 行定义一个 menu 类型的数组并赋初值。

⑤ 代码第 11 行到第 16 行定义了一个函数，该函数用来初始化图形模式。

下面，定义一个绘制主菜单内容的函数，如代码 14-10 所示。

代码第 14-10　绘制主菜单

```
01. initm()
02. {
03.     int L,T,R,i;//定义变量
04.     mWidth=550/MenuNum;//获得菜单项的宽度
05.     mHeight=20;//获得高度
06.     L=50;  T=50;  R=mWidth+L;
07.     setfillstyle(SOLID_FILL,1);//定义填充模式和填充颜色
08.     bar(50,50,600,400);//画矩形并填充
09.     setfillstyle(SOLID_FILL,7);//定义填充模式和填充颜色
10.     bar(50,50,600,70);//画矩形并填充
11.     setcolor(RED);//定义作图色
12.     settextstyle(1,HORIZ_DIR,1);//设置文本输出的字形、方向和大小
13.     outtextxy(L+12,T,MainMenu[0].MenuName);//输出菜单名称
14.     L=R;R=mWidth+L;
15.     for(i=1;i<MenuNum;i++)//输出所有主菜单
16.     {
17.         setcolor(BLACK);
18.         settextstyle(1,HORIZ_DIR,1);
19.         outtextxy(L+12,T,MainMenu[i].MenuName);
20.         L=R;R=R+mWidth;
21.     }
22. }
```

代码执行步骤如下。

① 代码第 1 行定义名称为 initm 的函数，该函数用来在图形屏幕上显示主菜单项。该函数无返回值，无参数。其函数体为代码第 3 行到第 21 行。

② 代码第 3 行到第 6 行是定义相应的变量，并为相应的变量赋初值。

③ 代码第 7、8 行是在图形屏幕上以坐标（50, 50）为左上角，以（600, 400）为右下角画一个矩形并用蓝色实填。

④ 代码第 9、10 行是在在图形屏幕上以坐标（50, 50）为左上角，以（600, 70）为右下角画一个矩形并用淡灰色实填。

⑤ 代码第 11、12、13 行是以红色为作图色并设置文本输出的字形、方向和大小，然后在淡灰色区域的相应位置上输出第一个主菜单名称。

⑥ 代码第 14 行重新定义文本输出的坐标位置。

⑦ 代码第 15 行到第 21 行通过 for 循环输出剩下的几个主菜单名称。代码第 17 行定义当前画笔颜色为黑色，然后定义文本输出的字形、方向和大小，接着输出菜单名称，最后再重新定义文本输出的坐标位置。

再下面，定义显示子菜单的函数，如代码 14-11 所示。

代码 14-11　显示子菜单

```
01. void showItems(int NewID)                /*显示*/
02. {
03.     int LL,TT,j;
04.     LL=mWidth*NewID+50;//获得第一个子菜单输出位置
05.     TT=70;
06.     //开辟一个单元
07.     saveImage=malloc(imagesize(LL,70,LL+mWidth,70+25*(MainMenu[NewID].
itemCount)));
08.     //屏幕上的图像复制到刚开辟的内存空间中
09.     getimage(LL,70,LL+mWidth,70+25*(MainMenu[NewID].itemCount),
saveImage);
10.     setfillstyle(SOLID_FILL,7);//用淡灰色实填
11.     settextstyle(1,HORIZ_DIR,1);//设置文本输出的字形、方向和大小
12.     //画一矩形并填充
13.     bar(LL,70,LL+mWidth-80,70+25*(MainMenu[NewID].itemCount));
14.     setcolor(RED);//设置作图色
15.     //画一矩形框
16.     rectangle(LL+5,70,LL+mWidth-85,65+25*(MainMenu[NewID].itemCount));
17.     //输出第一个菜单项
18.     outtextxy(LL+15,TT,(MainMenu[NewID].itemName[0]));
19.     setcolor(BLACK);
20.     //输出主菜单项
21.     outtextxy(LL+12,50,(MainMenu[NewID].MenuName));
22.     //输出随后的几个子菜单项
23.     for(j=1;j<(MainMenu[NewID].itemCount);j++)
24.     {
25.         TT=TT+25;
26.         outtextxy(LL+15,TT,MainMenu[NewID].itemName[j]);
27.     }
28. }
```

代码执行步骤如下。

① 代码第 1 行是定义函数，该函数名称为 showItems，函数无返回值，函数的形参为一

个整型变量，该整型变量表示当前要移动到的主菜单编号。其函数体为代码第 3 行到第 27 行。

② 代码第 3、4、5 行是定义相应的变量，并为一些变量赋初值。

③ 代码第 7 行从内存中开辟了一个相应大小的空间。

④ 代码第 9 行是将屏幕上相关大小的图像存储在刚刚开辟的内存空间中。

⑤ 代码第 10、11 行设置以淡灰色实填模式，并设置文本输出的字形、方向和大小。

⑥ 代码第 13 行是画一矩形并以淡灰色实填。

⑦ 代码第 14、16 行是以红色为作图色画一个矩形框。

⑧ 代码第 18 行到第 21 行是先输出当前主菜单的第一个子菜单名称，然后用黑色在原处重新输出主菜单名称。

⑨ 代码第 23 行到第 27 行是用 for 循环，输出剩下的几个子菜单项。

然后，定义主菜单和相应子菜单项的移动的函数，如代码 14-12 所示。

代码 14-12　主菜单和相应子菜单项的移动

```
01. void process(int OldID,int NewID)       /*移动主菜单*/
02. {
03.     int L,T;
04.     L=50+mWidth*OldID;//获得当前菜单位置
05.     T=50;
06.     settextstyle(1,HORIZ_DIR,1);
07.     setcolor(BLACK);//用黑色重画当前菜单
08.     outtextxy(L+12,T,MainMenu[OldID].MenuName);
09.     L=50+mWidth*NewID;
10.     setcolor(RED);//用红色重画要移动到的菜单项
11.     outtextxy(L+12,T,MainMenu[NewID].MenuName);
12. }
13. void process1(int OldID,int NewID,int m)       /*子菜单移动*/
14. {
15.     int L,T;
16.     L=50+mWidth*m;//获得当前子菜单项的位置
17.     T=70+OldID*25;
18.     settextstyle(1,HORIZ_DIR,1);
19.     setcolor(BLACK);//用黑色重画当前子菜单项的位置
20.     outtextxy(L+15,T,MainMenu[m].itemName[OldID]);
21.     T=70+NewID*25;
22.     setcolor(RED);//用红色重画要移动到的子菜单项的位置
23.     outtextxy(L+15,T,MainMenu[m].itemName[NewID]);
24. }
```

代码执行步骤如下。

① 代码第 1 行定义一个函数，该函数名称为 process，函数无返回值，函数的形参为两个整型变量，分别表示当前主菜单的编号，和要移动到的主菜单的编号。其函数体为代码第 3 行到第 11 行。

② 代码第 3、4、5 行定义相应变量并赋初值。

③ 代码第 6、7、8 行是用黑色及相同的文本格式在原位置上重新输出当前菜单名称。

④ 代码第 9、10、11 行是用红色及相同的文本格式在原位置上重新输出要移动到的菜单名称。

⑤ 代码第 13 行是定义名称为 process1 的函数，该函数无返回值，函数的形参为三个整型变量，分别表示当前子菜单的编号、要移动到的子菜单的编号，以及其相应的主菜单的编号。其函数体为代码第 15 行到第 23 行。

⑥ 代码第 15、16、17 行是定义相应变量并赋初值。

⑦ 代码第 18、19、20 行是用黑色及相同的文本格式在原位置上重新输出当前子菜单名称。

⑧ 代码第 21、22、23 行是用红色及相同的文本格式在原位置上重新输出要移动到的子菜单名称。

最后在主函数中调用相应的函数和变量，实现一个下拉式弹出菜单。如代码 14-13 所示。

代码第 14-13　主函数代码

```
01. void main()
02. {
03.     int OldID,NewID,head,tail,selectID,quit,c;//定义变量
04.     int OldID1,NewID1,head1,tail1;
05.     head=0;  tail=2;//为一些变量赋初值
06.     OldID=0;  NewID=0;
07.     OldID1=0;  NewID1=0;
08.     head1=0;
09.     quit=0;
10.     init();//初始化图形模式
11.     initm();//显示主菜单
12.     while(!quit)
13.     {
14.         //while(bioskey(1)==0);
15.         c=bioskey(0);//获得被按下键的值
16.         if(c==17400) selectID=START;
17.         else if(c==19200) selectID=LEFTSHIFT;//左边 shift 键
18.         else if(c==19712) selectID=RIGHTSHIFT;//右边 shift 键
19.         else if(c==7181) selectID=ENTER;//回车键
20.         else if(c==283) selectID=ESC;
21.         else if(c==20480) selectID=DOWN;
22.         else if(c==18432) selectID=UP;
23.         else if(c==11520) selectID=ALTX;
24.         else selectID=NULL;
25.         switch (selectID)
26.         {
27.             case START://开始
28.                 OldID=NewID;
29.                 NewID=0;
30.                 process(OldID,NewID);
31.                 break;
32.             case LEFTSHIFT://按下左边的 shift 键
```

```
33.                 if(mutex==0)//判断 mutex 的值是否等于 0
34.                 {
35.                     if(NewID==head)//若当前菜单项为第一个菜单项
36.                     {//则将要移动的菜单项改为最后一个菜单项
37.                         OldID=NewID;
38.                         NewID=tail;
39.                     }
40.                     else
41.                     {
42.                         OldID=NewID;
43.                         NewID--;
44.                     }
45.                     process(OldID,NewID);//移动主菜单项
46.                 }
47.                 break;
48.             case RIGHTSHIFT://按下右边的 shift 键
49.                 if(mutex==0)
50.                 {
51.                     if(NewID==tail)//若当前菜单项为最后一个菜单项
52.                     {//则将要移动的菜单项改为第一个菜单项
53.                         OldID=NewID;
54.                         NewID=head;
55.                     }
56.                     else
57.                     {
58.                         OldID=NewID;
59.                         NewID++;
60.                     }
61.                     process(OldID,NewID);////主菜单项的移动
62.                 }
63.                 break;
64.             case ENTER://若按下 enter 键，则显示当前菜单的子菜单
65.                 if(mutex==0)
66.                 {
67.                     showItems(NewID);//显示子菜单
68.                     mutex=1;
69.                     tail1=MainMenu[NewID].itemCount-1;//获得最后一个子菜单
项的编号
70.                 }
71.                 break;
72.             case ESC://若按下 ESC 键，则退出子菜单
73.                 if(mutex!=0)
74.                 {//将 saveImage 中图像送回到屏幕上
75.                     putimage(mWidth*NewID+50,70,saveImage,0);
76.                     setcolor(RED);//并用红色重画主菜单名
77.                     outtextxy(mWidth*NewID+62,50,(MainMenu[NewID].
MenuName));
78.                     mutex=0;
79.                 }
80.                 else//否则退出循环
81.                     quit=TRUE;
82.                 break;
```

```
83.             case DOWN:
84.                 if(mutex==1)
85.                 {
86.                     if(NewID1==tail1)//若当前的子菜单项为最后一个子菜单项
87.                     {//则将要移动到的子菜单项改为第一个子菜单项
88.                         OldID1=NewID1;
89.                         NewID1=head1;
90.                     }
91.                     else
92.                     {
93.                         OldID1=NewID1;
94.                         NewID1++;
95.                     }
96.                     process1(OldID1,NewID1,NewID);//子菜单项的移动
97.                 }
98.                 break;
99.             case UP:
100.                    if(mutex!=0)
101.                    {
102.                        if(NewID1==head1)//若当前的子菜单项为第一个子菜单项
103.                        {//则将要移动到的子菜单项改为最后一个子菜单项
104.                            OldID1=NewID1;
105.                            NewID1=tail1;
106.                        }
107.                        else
108.                        {
109.                            OldID1=NewID1;
110.                            NewID1--;
111.                        }
112.                        process1(OldID1,NewID1,NewID);
113.                    }
114.                    break;
115.                case ALTX:
116.                    exit(0);//退出程序
117.                default: break;
118.            }
119.        }
120.        getch();
121.        closegraph();
122.    }
```

代码执行步骤如下。

① 代码第 3 行到第 11 行定义相关的变量，并为一些变量赋初值。然后调用 init 函数进行图形模式初始化，和调用 initm 函数在图形屏幕上显示主菜单。

② 代码第 12 行是 while 循环，其循环条件为 quit 的等于 0，其循环体为代码第 14 行到第 118 行。

③ 代码第 15 行到第 24 行是获得当前按键的值，赋给变量 c。然后判断 c 的值，若 c 的值为 17 400，则将 START 赋给变量 selectID；若 c 的值为 19 200（即方向键←），则将 LEFTSHIFT 赋给变量 selectID；若 c 的值为 19 712（即方向键→），则将 RIGHTSHIFT 赋给变量 selectID；

若 c 的值为 7 181，则将 ENTER 赋给变量 selectID；若 c 的值为 283，则将 ESC 赋给变量 selectID；若 c 的值为 20 480，则将 DOWN 赋给变量 selectID；若 c 的值为 18 432，则将 UP 赋给变量 selectID；若 c 的值为 11 532，则将 ALTX 赋给变量 selectID；若前面都不是，则将 NULL 赋给变量 selectID。

④ 代码第 25 行是判断当前 selectID 的值。若其值为 START 则执行代码第 28 行到第 31 行先将要移动到的菜单编号设置为第一个主菜单，然后调用 process 函数将第一个菜单名称用红色显示，表示为当前选中的项，并退出该 switch 结构。

若 selectID 的值为 LEFTSHIFT 则执行代码第 33 行到第 47 行。代码第 33 行是判断 mutex 的值是否等于 0，若是则再判断 NewID 是否等于 head 的值，若是则将 NewID 的值赋给 OldID，并将 tail 的值赋给 NewID；若 NewID 不等于 head 的值，则将 NewID 的值赋给 OldID，并将 NewID 值减 1；然后调用 process 函数移动选择菜单；最后退出该 switch 结构。

若 selectID 的值为 RIGHTSHIFT 则执行代码第 49 行到第 63 行。代码第 49 行是判断 mutex 的值是否等于 0，若是则再判断 NewID 是否等于 tail 的值，若是则将 NewID 的值赋给 OldID，并将 head 的值赋给 NewID；若 NewID 不等于 tail 的值，则将 NewID 的值赋给 OldID，并将 NewID 值加 1；然后调用 process 函数移动选择菜单；最后退出该 switch 结构。

若 selectID 的值为 ENTER 则执行代码第 65 行到第 71 行。代码第 65 行是判断 mutex 的值是否等于 0，若是则调用 showItems 函数显示当前主菜单下的所有子菜单，然后将 mutex 的值赋为 1，并获得最后一个子菜单项的编号。最后退出该 switch 结构。

若 selectID 的值为 ESC 则执行代码第 73 行到第 82 行。代码第 73 行是判断 mutex 的值是否等于 0，若不是则将 saveImage 中图像送回到屏幕上，并用红色重新输出主菜单名称，然后将 mutex 的值赋为 0。否则将 TRUE 赋给变量 quit。最后退出该 switch 结构。

若 selectID 的值为 DOWN 则执行代码第 84 行到第 98 行。代码第 84 行是判断 mutex 的值是否等于 1，若是则再判断 NewID1 是否等于 tail1 的值，若是则将 NewID1 的值赋给 OldID1，并将 head1 的值赋给 NewID1；若 NewID1 不等于 tail1 的值，则将 NewID1 的值赋给 OldID1，并将 NewID1 值加 1；然后调用 process1 函数移动选择子菜单；最后退出该 switch 结构。

若 selectID 的值为 UP 则执行代码第 100 行到第 114 行。代码第 100 行是判断 mutex 的值是否等于 0，若不是则再判断 NewID1 是否等于 head1 的值，若是则将 NewID1 的值赋给 OldID1，并将 tail1 的值赋给 NewID1；若 NewID1 不等于 head1 的值，则将 NewID1 的值赋给 OldID1，并将 NewID1 值减 1；然后调用 process1 函数移动选择子菜单；最后退出该 switch 结构。

若 selectID 的值为 ALTX，则执行代码第 116 行退出程序。若为其他，则直接退出 switch 结构。

⑤ 代码第 120、121 行是接收用户输入的一个字符，然后退出图形模式。

（2）保存源文件

按【F2】键，在弹出的保存文件窗口中，选择或输入文件保存到的目录并修改该文件名为 Ex14-9.c。

（3）编译、链接、运行程序

按【F9】键，编译并链接代码 14-9，然后按【Ctrl+F9】快捷键运行该程序。结果如图 14-16 所示。

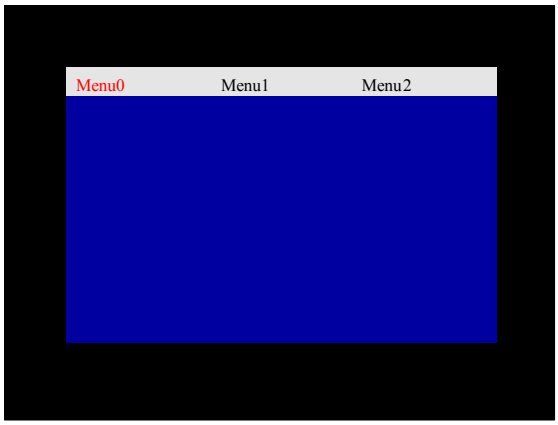

图 14-16　实例 14-9 的运行结果

当前第一个主菜单（Menu0）显示红色，表示该菜单被选择，用【←】和【→】键可以选择不同的菜单，例如按一下【→】键，则菜单 Menu1 显示红色，而 Menu0 变为黑色，表示当前选择了菜单 Menu1。

按【Enter】键时，相应被选中主菜单的子菜单就会出现，如图 14-17 所示。子菜单可通过【↑】和【↓】键来选择不同的菜单项。当按【Esc】键时，则当前菜单的子菜单就会消失。

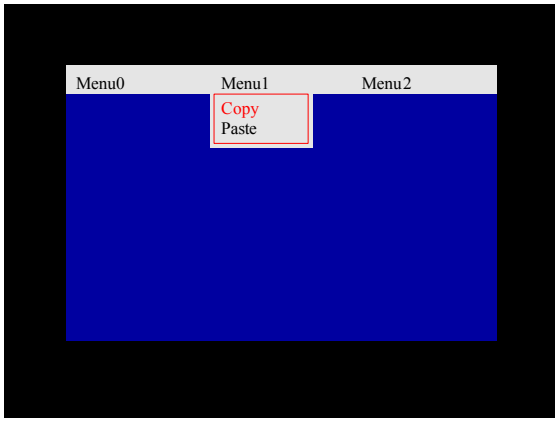

图 14-17　子菜单

14.4　动画技术

利用人的视觉暂留这一生理特点，即对动态的图像变化，仅能分辨出时间间隔为 25 毫米左右的变化，若太快，则分辨不出了，因而可以用类似电影的方法，将一个图像分解成不同时间出现的图像，然后一张张快速呈现在屏幕上，从视觉效果上看，就如同这些画面在连续变化一样，因而给人以动的视觉感觉。在屏幕上制作动画，这也是目前热门的题目，动画技术是计算机图形学中的一个内容，它可用于游戏娱乐，辅助教学，科学实验模拟、论证、仿真等计算

机辅助设计（CAD）。Turbo C 提供的一些图形处理函数可用于动画设计。总结起来，实现动画，可用如下方法来实现。

14.4.1 利用动态开辟图形窗口的方法

前面介绍了图形窗口的操作函数，这样我们可以利用图形窗口设置技术，实现图形窗口的动画效果，例如可在不同图形窗口中设置同样的图像，而让图形窗口沿 x 轴方向移动设置，这次出现前要清除上次图形窗口的内容，这样就会出现图像沿 x 轴移动的效果。也就是说，在位置动态变化，但大小不变的图形窗口中（用 setviewpot()函数），设置固定图形（也可是微小变化的图像），这样呈现在观察者面前的是当前图形窗口位置在动态变化，因而在屏上看到的图像就好像在动态变化一样。

例如下面的代码就是这样做的，不断地沿 x 轴开辟图形窗口，就像一个大小一样的窗口沿 x 轴在移动，由于总有 clearviewport 函数清除上次窗口的相同立方体，因而视觉效果上，就像一个立方体从左向右移动一样。程序中定义的 movebar 函数作用是开辟一个图形窗口，并画一个填色的立方体，保留一阵（delay（1000））然后清除它，主程序不断调用它，因每次顶点 x 坐标在增加，因而效果是立方体沿 x 轴从左向右在运动。

```
01. #include <graphics.h>
02. #include <dos.h>
03. main()
04. {
05.     int i,graphdriver,graphmode;
06.     graphdriver=DETECT;
07.     initgraph(&graphdriver,&graphmode,"c:\\tc\\bgi");//初始化图形模式
08.     for(i=0;i<25;i++)
09.     {
10.         setfillstyle(1,i);//设置填充模式和填充颜色
11.         movebar(i * 20);
12.     }
13.     closegraph();
14. }
15. movebar(int xorig) /* 设窗口并画填色小立方体 */
16. {
17.     setviewport(xorig,0,639,199,1); //定义图形窗口
18.     setcolor(5); //设置边框颜色
19.     bar3d(10,120,60,150,40,1); //画一立方体
20.     floodfill(70,130,5); //填充立方体
21.     floodfill(30,110,5);
22.     delay(1000);//延时 1000ms
23.     clearviewport();//清除图形窗口内的图像
24. }
```

代码执行步骤如下。

① 代码第 5 行到第 7 行是定义相关的变量并初始化图形模式。

② 代码第 8 行到第 12 行是用 for 循环，设置以不同的填充颜色实填，然后调用 movebar 函数实现在不同的位置上画立方体。

③ 代码第 13 行是退出图形模式。

④ 代码第 15 行定义了一个函数，该函数名称为 movebar，形参为一个整型变量。其函数体为代码第 17 行到第 23 行。

⑤ 代码第 17 行定义了一个图形窗口。代码第 18 行设置输出图形的边框颜色为洋红色。代码第 19 行是在图形窗口中画一个立方体。代码第 20、21 行是在为立方体的各面填充颜色。代码第 22 行是延时 1000ms。代码第 23 行清除图形窗口内的图像。

编者手记

注意，采用这种方法对较复杂图形不宜，因为在图形窗口内画这种图形要占较长时间，这样图形窗口位置切换的时间就变得较长，因而动画效果就变差。

14.4.2 利用页交替的方法

前面介绍过图形屏幕操作函数，setactivepage()函数和 setvisualpage()函数，利用这两个函数也可以实现一些动画的效果。例如，将当前显示页和编辑页分开（用 setvisualpage 函数和 setactivepage 函数），在编辑页上画好图形后，立即令该页变为显示页显示，然后在上次的显示页上（现在变为编辑页）进行画图，画好后，又再次交换，如此编辑页和显示页反复地交换，在观察者的视觉上，就出现了动画的效果。要让页的交替速度快，唯一的办法是缩短在页上的画图时间（即采用优化的画法）。

例如下面的代码，首先用 setactivepage(1)设置 1 页为编辑页，在上面画出一个红色边框、用绿色填充的圆，此图并不显示出来（因默认时，定义 0 页为可视页）。接着又定义 0 页为编辑页并清屏（即清 0 页），并在其上画出一个用洋红色填充的方块，然后定义 0 页为可视页，该方块将在屏上显示出来。接着进入 do 循环，设置 1 页为可视页，因而其上的圆便在屏上显示出来，方块的图像消失，用 delay（1000）将圆图像保持 1 000 毫秒即 1 秒，当不按键时，下一次循环又将 0 页设为可视页，因而方块的图像显示出来，圆图像又消失。保持 1 秒后，又重复刚开始的过程。这样我们就会看到：屏上同一位置洋红色方块和绿色圆交替出现，若将 delay 时间变少，将会出现动画的效果。

```
01. #include <graphics.h>
02. #include <dos.h>
03. main()
04. {
05.     int i,graphdriver,graphmode,size,page;
06.     graphdriver=DETECT;
07.     initgraph(&graphdriver,&graphmode,"c:\\tc\\bgi");
08.     cleardevice();
09.     setactivepage(1); /* 设置 1 页为编辑页 */
10.     setbkcolor(BLUE);
11.     setcolor(RED);
12.     setfillstyle(1,10);
13.     circle(130,270,30); /* 画圆 */
14.     floodfill(130,270,4); /* 用绿色填充圆 */
15.     setactivepage(0); /* 设置 0 页为编辑页 */
```

```
16.     cleardevice(); /* 清 0 页 */
17.     setfillstyle(1,5);
18.     bar(100,210,160,270); /* 画方块并填充洋红色 */
19.     setvisualpage(0); /* 设置 0 页为可视页 */
20.     page=1;
21.     do
22.     {
23.         setvisualpage(page); /* 显示设定页的图像 */
24.         delay(1000); /* 延迟 1000ms */
25.         page=page-1;
26.         if(page<0)
27.             page=1;
28.     } while(!kbhit());
29.     getch();
30.     closegraph();
31. }
```

代码执行步骤如下。

① 代码第 5 行到第 8 行是初始化图形窗口并清屏。

② 代码第 9 行到第 14 行是先设置 1 页为编辑页，然后在该页上以红色为边框色，画一圆并以绿色实填该圆。

③ 代码第 15 行到第 18 行是先设置 0 页为编辑页，并清屏，然后在该页上，画一矩形并用洋红色填充。

④ 代码第 19 行是设置 0 页为可视页。代码第 20 行定义变量 page 的初值为 1。

⑤ 代码第 21 行到第 28 行是 do...while 循环。该循环先是根据 page 值设置当前可视页；然后延时 1 000ms 后，将 page 的值减 1；接着，判断 page 的值是否小于 0，若是则将 page 赋为 1；最后用 kbhit()判断是否有键按下，若是则退出循环，若不是则继续下一轮循环。

⑥ 代码第 29、30 行是先接收用户输入的一个字符，然后退出图形模式。

编者手记

这里简单说明一下 kbhit()函数。该函数在头文件 stdio.h 中，用来检测键盘是否有键按下。如果有键按下，则返回对应键值；否则返回零。它与 bioskey()函数不同，kbhit 不等待键盘按键，无论有无按键都会立即返回。

14.4.3 利用画面存储再重放的方法

如下例所示，同制作幻灯片一样，将整个动画过程变成一个个片段，然后存到显示缓冲区内，当把它们按顺序重放到屏幕上时，就出现了动画效果，这可以用 getimage 函数和 putimage 函数来实现，这种方法较前两种都快，因它已事先将要重放的画面画好看，余下的问题，就是计算应在什么位置重放的问题了。

该程序中演示了利用这种方法产生的动画效果，程序将用 circle 函数画出圆并用洋红色填充圆的图形用 getimage 函数存到内存 buffer 中，然后再用 putimage 函数将该圆放到屏幕的原

来圆的相对位置，接着 do 循环不断地用 putimage 函数复制两个小圆（注意！每复制一次，由于存的图像面积覆盖了原来的圆，故显示一个面），其方向是左边圆沿 x 轴正向复制，右边圆沿 x 轴负向复制，由于复制速度较快，因而动态地看到小球相向运行，直至碰撞（即两个小球圆心相距 60 时，或说 i=184 时），接着 for 循环又不断复制两球，直至球碰到屏的尽端，当不按键时，又重复 do 循环，如此反复，就看到了两个洋红色小球碰撞、弹回、又碰撞，……，当按任一键时，此过程结束。


```
01. #include <graphics.h>
02. main()
03. {
04.     int i,j,graphdriver,graphmode,size;
05.     void *buffer;
06.     graphdriver=DETECT;
07.     initgraph(&graphdriver,&graphmode,"c:\\tc\\bgi");
08.     setbkcolor(BLUE);
09.     cleardevice();
10.     setcolor(YELLOW);
11.     setlinestyle(0,0,1);   //用细实线
12.     setfillstyle(1,5);     //用洋红实填充
13.     circle(100,200,30);
14.     floodfill(100,200,YELLOW);       //填充圆
15.     size=imagesize(69,169,131,231);  //指定图像占字节数
16.     buffer=malloc(size);            //为其分存储区
17.     getimage(69,169,131,231,buffer); //保存圆球图
18.     putimage(500,169,buffer,COPY_PUT);//在另一区域重新显示
19.     do
20.     {
21.         for(i=0;i<185;i++)
22.         {
23.             putimage(70+i,170,buffer,COPY_PUT);//左边球向右运动
24.             putimage(500-i,170,buffer,COPY_PUT);//右边球向左运动
25.         }
26.         for(i=0;i<185;i++)
27.         {
28.             putimage(255-i,170,buffer,COPY_PUT);//左边球向左运动
29.             putimage(315+i,170,buffer,COPY_PUT);//右边球向右运动
30.         }
31.     }while(!kbhit());//当不按键时重复上述过程
32.     getch();
33.     closegraph();
34. }
```

代码执行步骤如下。

① 代码第 4 行到第 9 行定义相关的变量并初始化图形模式，然后设置屏幕背景色为蓝色，接着清屏。

② 代码第 10 行到第 14 行设置以黄色细实线画一圆，并用洋红色实填该圆。

③ 代码第 15、16 行先调用 imagesize 获得圆球图像所占的字节数 size，然后在内存中开辟一个同样字节数的存储区 buffer。

④ 代码第 17、18 行调用 getimage 函数将圆球图像保存到内存区 buffer 中，然后调用 putimage 函数在另一区域重新显示圆球图像。

⑤ 代码第 19 行到第 31 行是 do-while 循环。该循环先是通过 for 循环将圆球图像依次从左右两边向中间复制，形成左右两球同时向中间运动的动画图像；接着，又用 for 循环将圆球图像依次从中间向左右两边复制，形成中间两球向左右两边运动的动画图像；然后判断是否有键被按，若没有则继续下一轮循环。

⑥ 代码第 32、33 行是接收用户输入的一个字符，然后退出图形模式。

14.4.4 直接对图像动态存储进行操作

利用显示适配器上控制图像显示的各个寄存器和图像存储器 VRAM，对其进行直接操作和控制，从而可以高效快速地实现动画效果。这可以用汇编语言，直接进行 BIOS 调用来实现，但这涉及许多硬件结构，有一定难度，这里就不做介绍了。

实例 14-10 运动的卫星

实例说明	程序将产生一个在繁星背景下的一个由经纬线组成的蓝色地球，然后一卫星从左到右缓慢飞过，周而复始，给人一种遨游太空的神秘感，屏幕的下方写出了 COMPUTER WORLD 字样。
实例代码	光盘\\chapter14\14-10\
知识要点	动画技术。

（1）编辑源代码

启动 Turbo C 3.0 编译器，在编辑区中输入代码。首先，输入如下代码：

```
#include<graphics.h>
#include<conio.h>
#include<stdlib.h>
#include<stdio.h>
#define SIZE 10
```

接着，定义画卫星和画星星的函数，如代码 14-14 所示。

代码 14-14 画卫星和画星星的函数

```
01. void draw_image(int x,int y)//画卫星
02. {
03.     moveto(x+10,y);
04.     setcolor(14);
05.     setfillstyle(1,4);//设置填充模式和填充颜色
06.     //画卫星天线
07.     linerel(-3*10,2*10);
08.     moveto(x+10,y);
09.     linerel(-3*10,-2*10);
10.     moveto(x+25,y);
11.     linerel(-5*10,0);
12.     fillellipse(x+13,y,8,8);//画本体
13. }
```

```
14.
15. void putpixel_demo(void)//画星星
16. {
17.
18.     int i,dotx,doty,h,w,color,maxcolor;
19.     maxcolor=getmaxcolor();//得到当前最大颜色数
20.     w=getmaxx();//获得 x 轴和 y 轴的最大值
21.     h=getmaxy();
22.     for(i=0;i<5000;++i)
23.     {
24.         dotx=1+random(w-1);
25.         doty=1+random(h-1);
26.         color=random(maxcolor);
27.         putpixel(dotx,doty,color);//用点表示星星
28.     }
29. }
```

代码执行步骤如下。

① 代码第 1 行定义了一个函数，该函数名称为 draw_image，函数形参为两个整型变量，其函数体为代码第 3 行到第 12 行。

② 代码第 3、4、5 行是将画笔移动到指定的（x+10, y）位置，然后设置画笔颜色为黄色，以及设置以红色实填模式。

③ 代码第 7 行到第 11 行是为卫星画天线。然后代码第 12 行画卫星的本体，一圆型并用红色填充。

④ 代码第 15 行定义了一个函数，该函数名称为 putpixel_demo，函数形参为两个整型变量，其函数体为代码第 18 行到第 28 行。

⑤ 代码第 18 行是定义相关的整型变量。

⑥ 代码第 19、20、21 行是获得当前最大颜色值，以及 x 轴和 y 轴的最大值。

⑦ 代码第 22 行到第 28 行是用 for 循环实现在不同位置输出不同颜色点。该循环先是随机获得坐标 x 和 y 的值，以及颜色值，然后以随机的颜色和坐标画点。

编者手记

上面用到了 random 函数（随机函数），这里对其简单说明一下。C 语言中常用的随机函数有：

```
int random(int num); //返回一个 0 到 num-1 之间的随机整数
int rand(void); //返回从 0 到 RAND_MAX(0x7fff) 之间的随机数
void srand(unsigned seed);//产生随机数的起始发生数据，和 rand 函数配合使用
```

它们在头文件 stdlib.h 中，所以在使用它们时，要在程序的开头输入下面命令：

```
#include " stdlib.h "或#include < stdlib.h >
```

要注意，random 函数并不是 ANSI C 标准函数，它不能在 gcc，vc 等编译器下编译通过。

然后，在 main 函数中使用相关的函数和变量，实现动画效果。如代码 14-15 所示。

代码 14-15　主函数代码

```
01. void main()
02. {
03.     int gmode,gdriver=DETECT;
04.     void *pt_addr;
05.     int x,y,maxx,maxy,midx,midy,i;
06.     unsigned int size;
07.     initgraph(&gdriver,&gmode,"c:\\tc\\bgi");
08.     maxx=getmaxx();//取允许的最大 x 值
09.     maxy=getmaxy();//取允许的最大 y 值
10.     midx=maxx/2;//获得 x 轴的中间值
11.     x=0;
12.     midy=maxy/2;//获得 y 轴的中间值
13.     y=maxy/2;
14.     setcolor(YELLOW);//设置画笔颜色为黄色
15.     settextstyle(TRIPLEX_FONT,HORIZ_DIR,4);//设置文本输出的字形、方向和大小
16.     settextjustify(CENTER_TEXT,CENTER_TEXT);//设置文本输出的位置
17.     outtextxy(midx,400,"COMPUTER WORLD");//输出文本
18.     setbkcolor(BLACK);//设置背景色
19.     setcolor(LIGHTBLUE);//设置画笔颜色
20.     for(i=0;i<13;i++)//画地球
21.     {
22.         ellipse(midx,midy,0,360,100,100-8*i);
23.         ellipse(midx,midy,0,360,100-8*i,100);
24.     }
25.     draw_image(x,y);//画卫星
26.     size=imagesize(x,y-SIZE,x+(4*SIZE),y+SIZE);//获得存储卫星图像所需字节数
27.     pt_addr=malloc(size);//开辟一块内存区域
28.     getimage(x,y-SIZE,x+(4*SIZE),y+SIZE,pt_addr);//保存卫星图像
29.     putpixel_demo();//画星星
30.     setcolor(WHITE);//设置画笔颜色
31.     setlinestyle(SOLID_LINE,0,NORM_WIDTH);//设置线型
32.     rectangle(0,0,maxx,maxy);//画一矩形框
33.     while(!kbhit())//移动卫星图像
34.     {
35.         putimage(x,y-SIZE,pt_addr,XOR_PUT);
36.         x=x+5;
37.         if(x>maxx)
38.             x=0;
39.         putimage(x,y-SIZE,pt_addr,XOR_PUT);
40.         delay(100);//延时
41.     }
42.     free(pt_addr);//释放内存空间
43.     closegraph();//退出图形模式
44. }
```

代码执行步骤如下。

① 代码第 3 行到第 7 行是定义相关的变量以及初始化图形模式。

② 代码第 8 行到第 13 行先取允许的最大 x、y 值，然后获得 x 轴和 y 轴上的中间值，并

为变量 x 和 y 赋值。

③ 代码第 14 行到第 17 行以设定好的颜色、文本字型、方向和大小输出文本 COMPUTER WORLD。

④ 代码第 18 行到第 24 行是设置屏幕背景色为黑色，设置画笔颜色为淡蓝色，然后用 for 循环，画一个由经纬线组成的蓝色地球。

⑤ 代码第 25 行到第 28 行调用 draw_image 函数画一卫星，接着调用 imagesize 函数获得该卫星图像所占的字节数，并在内存中开辟一个同样的存储区域，然后将卫星图像存储在该内存区中。

⑥ 代码第 29 行调用 putpixel_demo 函数，在图形屏幕上画点，即星星。

⑦ 代码第 30、31、32 行以设定好的颜色和线型在图形屏幕上画一个最大的矩形框。

⑧ 代码第 33 行到第 41 行是 while 循环。该循环先是判断当前是否有键按下，若没有则调用 putimage()函数将卫星图像输出到另一个位置，然后将 x 坐标的值加 5；接着判断 x 的值是否大于 x 轴的最大值，若是则将 x 的值赋为 0；接着将再调用 putimage()函数将卫星图像输出到另一个位置；然后延时 100ms，继续下一轮循环。

⑨ 代码第 42、43 行是释放之前开辟的内存空间，然后退出图形模式。

（2）保存源文件

按【F2】键，在弹出的保存文件窗口中，选择或输入文件保存到的目录并修改该文件名为 Ex14-10.c。

（3）编译、链接、运行程序

按【F9】键，编译并链接代码 14-10，然后按【Ctrl+F9】快捷键运行该程序。结果如图 14-18 所示。

图 14-18　实例 14-10 的运行结果

14.5　本章小结

本章主要介绍了 C 语言的高级应用，包括文本的屏幕输出和键盘输入、TC 图形处理、菜单设计以及动画技术。通过理论知识和实际例子的结合，向读者详细介绍了相关的内容。