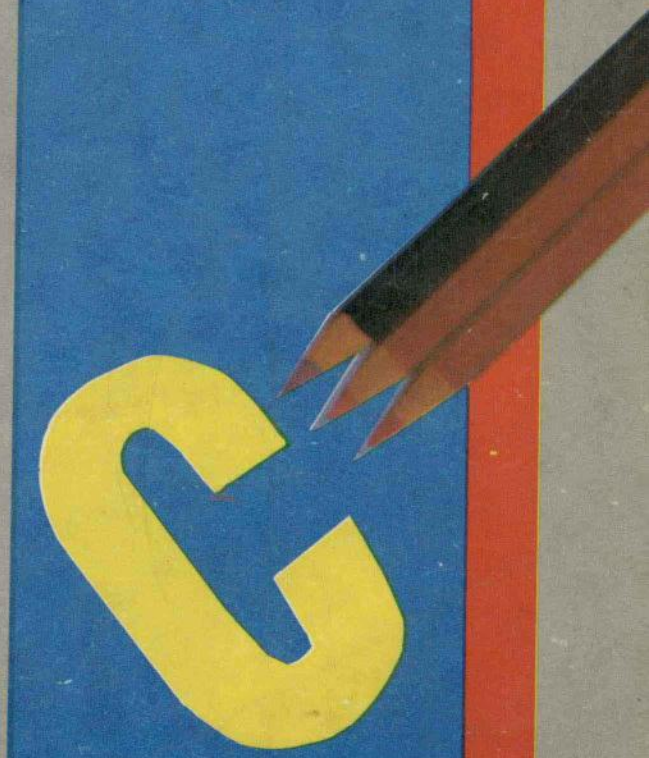
计算机知识普及系列丛书

实用C语言编程艺术和范例

尤　里
丁　东　编写
梁　水



学苑出版社

计算机知识普及系列丛书

实用C语言编程艺术和范例

创立自己一流程序的秘密和捷径

尤　里　　金　雷　　梁　水
丁　东　　郭　力　　王　英　编写
刘　凯　　钟文峰

韩　卫　　审校

学苑出版社
1993.

(京)新登字 151 号

内 容 提 要

全书共分简介、前言、正文、附录、参考文献和编后语六部分。其中正文部分就具体应用划分了若干章节，每章都演示了 C 程序设计问题，并给出了相关应用的完整代码；附录则是对相应正文的补充或建议，它们与参考文献结合起来，将是读者进一步回顾或学习的有价值的资料。

欲购本书的用户，请直接与北京 8721 信箱联系，电话 2562329，邮码 100080。

计算机知识普及应系列丛书

实用 C 语言编程艺术和范例

编　　写：尤　里　金　雷　梁　水　丁　东
郭　力　王　英　刘　凯　钟文锋
审　　校：韩　卫
责任编辑：甄国宪
出版发行：学苑出版社　　**邮政编码**：100032
社　　址：北京市西城区成方街 33 号
印　　刷：兰空印刷厂
开　　本：787×1092　1/16
印　　张：21.25　　**字　数**：492 千字
印　　数：1～3000 册
版　　次：1993 年 12 月北京第 1 版第 1 次
ISBN7－5077－0821－7/TP・19
本册定价：19.00 元

目　　录

第一章　弹出式和下拉式菜单

软件专业编写人员开发的程序给人最深刻的印象是有一个好的菜单系统，因而会使使用者有一种轻松、活泼的感觉。当然，这个问题在概念上是很简单的，但要真正地编写出风格优美的弹出式和下拉式菜单，没有专门的知识和经验是不行的。

开发弹出式和下拉式菜单时，你首先要明确怎样才能使你的程序直接控制视频。实际上，对屏幕的直接控制，并不意味着需要你直接动手去操纵机器硬件或者操作系统，这项任务是由一些小的程序实现的，这些程序中用到了C语言的标准I／O控制函数。本章中编写的屏幕控制函数适用于任何使用DOS操作系统的微机，但是请注意，你的计算机还应该具有与IBM兼容的BIOS。本书采用DOS操作系统的原因在于它是目前世界上使用最广泛的操作系统，当然，你也可以把在这里使用的基本方法推广到其它的操作系统上去。

假如你现在对弹出式和下拉式菜单不感兴趣，那么你还是应该读一下本章中与视频适配器讨论有关的部分。本章中涉及的许多基本概念在以后有关窗口和图形设计的章节中也会用到。

1.1什么是弹出式和下拉式菜单

明白弹出式和下拉式菜单的基本概念以及它们与标准菜单之间的差别是很重要的。标准菜单使用时，屏幕先被清除或者动滚，随后菜单出现在屏幕上。在你选中了菜单中的某一选择项以后，屏幕再次被清除或者滚动。然后你选择的命令就开始执行。标准式菜单中，命令的选择是通过使用一个数字或者某个选择项的第一个字母实现的。

当一个弹出式或下拉式菜单被激活时，它就覆盖了屏幕上先前存在的东西。你选择了需要的命令以后，屏幕又恢复到调用菜单以前的状态。弹出式和下拉式菜单中选择命令通常采用下面两种方法。第一种是按一个热键。热键是一个与你选择项有关的数字或字母。第二种方面是使用方键把光标移到你的选择项随后按回车键。光标选中的命令为反视频显示或者用另外一种颜色显示。

标准菜单与弹出式或下拉式菜单之间关键的差别在于：激活标准菜单迫使程序停止执行，激活弹出或下拉式菜单只是使正在执行的程序“悬挂”了起来。从用户的观点来看，使用标准菜单将分散用户的注意力。而使用弹出或下拉式菜单只是引起简单的中断，用户的注意力不会受到任何影响。

弹出和下拉式菜单之间的差别是比较小的。使用弹出式菜单时，同一时刻屏幕上只出现一个菜单。弹出式菜单适用于菜单只有一级深度的情况，也就是说，菜单中的选择项再没有子选择项的情况。另一方面，几个下拉式菜单可以被同时激活在屏幕上。其中有一个是一级菜单，其余是二级菜单、三级菜单等等。更深一层的菜单是为了进一步阐述上一级菜单的选择项的功能而设计的。例如，开发一个订购水果的菜单时，你可能会用到下拉式菜单。如果你从一级菜单中选择了“苹果”这一项。二级菜单就被激活显示苹果的颜色。你选择了事先确定的颜色之后，会激发第三个菜单，这个菜单显示的是对应于这种颜色的苹果品种。

你也可以认为弹出式菜单是没有更深层次的下拉式菜单，然而对这两种菜单设计不同的子例程的优点是显而易见的，这是因为设计下拉式菜单要付出比设计弹出式菜单更多的开支。

有许多的方法能使你在屏幕上建立一个菜单，但本章中采用的是最常见的菜单设计方法，这种方法的特点是菜单中的每个选择项都在新的一行上。下面显示的菜单是采用这种方法设计的，它的内容与邮政局通信有关：

1.2视频适配器的基本知识

设计弹出式和下拉式菜单需要直接控制屏幕，所以了解关于视频适配器的知识显得非常重要。适配器的四种最常见的形式是单色适配器、彩色图形适配器（CGA）、增强图形适配器(EGA)和视频图形陈列(VGA)。CGA，EGA和VGA有多种视频模式，如40列文本模式、80列文本模式、或者是图形模式。这些模式如表1-1所示。本章中设计的菜单子例程采用的是80列文本视频模式。这意味着系统的视频模式应该设置为模式2、模式3或者模式7。无论你采用以上三种模式中的哪一种，屏幕左角上的坐标都是0，0。

Table 1-1

The Video Modes for the IBM Line of Microcomputers

Mode	Type	Dimensions	Adapters
0	Text, b/w	40x25	CGA, EGA, VGA
1	Text, 16 colors	40x25	CGA, EGA, VGA
2	Text, b/w	80x25	CGA, EGA, VGA
3	Text, 16 colors	80x25	CGA, EGA, VGA
4	Graphics, 4 colors	320x200	CGA, EGA, VGA
5	Graphics, 4 gray tones	320x200	CGA, EGA, VGA
6	Graphics, b/w	640x200	CGA, EGA, VGA
7	Text, b/w	80x25	Monochrome
8	Graphics, 16 colors	160x200	PCjr
9	Graphics, 16 colors	320x200	PCjr
10	Graphics, 4 colors	640x200	PCjr
11	Reserved		
12	Reserved		
13	Graphics, 16 colors	320x200	EGA, VGA
14	Graphics, 16 colors	640x200	EGA, VGA
15	Graphics, 4 colors	640x350	EGA, VGA
16	Graphics, 16 colors	640x350	VGA
17	Graphics, 2 colors	640x480	VGA
18	Graphics, 16 colors	640x480	VGA
19	Graphics, 256 colors	640x200	VGA

屏幕上显示的字符存放在专为显示适配器而保留的RAM区中。单色适配器视频RAM区的起始地址是B000:0000，CGA/EGA/VGA视频RAM是从B800:0000开始的。尽管为CGA和EGA设计的屏幕上控制函数在某些模式下不同，但是它们在模式2和模式3下都是相同的。

屏幕上显示的每个字符需要两字节的视频存储器，其中第一个字节存放字符本身，第二个字节用来存放字符的屏幕显示属性。对于彩色适配器，属性字节的每位的含义如表1-2所示。如果你已经有了CGA，EGA或者VGA，你应该知道它们默认的视频模式是3，默认的显示字符属性是7。这种情况下，三种前景颜色都处于开的位置，其结果是屏幕上显示的字符是白色的。为了产生反视频效果，属性字节中与前景颜色有关的几位都应处于闭的状态，与背景有关的三位置于开的状态，这时字符属性值为0x70。

单色适配器能使字符闪烁或者加亮字符显示，单色适配器的属性值为7，字符处于正常显示状态；属性字节为0x70，是反视频方式，属性字节值为1时，显示字符的下面有下划线。

Table 1-2

The Video Attribute Byte

Bit	Binary Value	Meaning When Set
0	1	Blue foreground
1	2	Green foreground
2	4	Red foreground
3	8	Low intensity
4	16	Blue background
5	32	Green background
6	64	Red background
7	128	Blinking character

视频适配器的保留内存是用于显示80列文本视频模式所用内存的4倍。这是由以下两个因素确定的。首先，图形显示需要更多的内存（当然不包括单色适配器）。其次，可以使视频RAM具有储存并显示多个屏幕的能力。需要的时候，可以很方便地使位于某一视频存储区域的画面出现在屏幕上。每个储存区域称为一个视频页，学会利用多个视频页是很有好处的。DOS初始化时，默认的视频页是第0页，而且在一般情况下用到的都是第0页。由于这个原因，本章中的子例程是采用0视频页编程的。当然，如果需要的话，也可以使用别的视频页。

控制视频适配器的方法有三种。第一种方法是通过调用DOS实现的。这种方法的速度是很慢的，不能适用于弹出式或下拉式菜单的设计。第二种方法是调用BIOS子例程完成这个任务。当设计的菜单较小，而且机器的运行速度较快时，这种方法是可行的。第三种方法是采用直接存取视频RAM的方法。这种方法速度最快，但它需要你做更多的工作。本章中，你将会看到两种不同风格的视频子例程，一种是使用BIOS编写的，另一种是通过直接存取视频RAM实现的。

1.3使用BIOS控制屏幕

设计弹出式和下拉式菜单，你面临的第一个问题首先要把菜单显示的区域保存起来，这样当你使用完了菜单以后，就可以接着把屏幕恢复到调用菜单以前的状态。为此，你一定要创建一个子例程，它能把菜单显示的部分暂时存起来。这节中你见到的储存和恢复部分屏幕的方法是基于调用两个内部BIOS函数实现的，其一从屏幕上读取字符，另一个是向屏幕上写字符。

如你所知，通过调用BIOS把视频RAM映象到屏幕上的方法是比较慢的。然而使用这种方法的一个好处在于只要你的计算机的BIOS与IBM兼容，即使实际使用的屏幕显示硬件不同，你也可以放心地调用BIOS函数达到你的目的。所以，BIOS调用的方法适合于编写移植较为广泛的程序（实际上，当今几乎所有的基于DOS操作系统的计算机与IBM是100%兼容的，当然，许多早期的系统并非如此）。

1 使用int86( )

利用软中断的方法能够实现BIOS调用。对于不同的任务，BIOS具有相对应的中断。用于屏幕控制的中断是INT10，它可以控制视频显示器（如果你对使用BIOS还不太熟悉的话，请参考下面这本书中有关BIOS调用部分的内容。这本书是：《C：The Complete Refernce》(2d.ed.BerKeley：Osborne/McGraw—Hill，1990。）像许多BIOS中断一样，INT10可以实现多种任务。至于具体使用那一种功能是由你向寄存器AH中输入的值确定的。如果调用的函数有返值，这个返值通常是存放在寄存器AL中返回的；当你从一个返值时，其它的寄存器也会被用到。为了获取BIOS中断，你需要使用一个叫做int86( )的C函数。有些编译器中这个函数的名称与此不一样，但是在Microsoft C和Turbo C中，它都被称为int86( )。下面的讨论是适用于Microsoft C和Turbo C的，但你应该明白如何在其它编译器上使用这个函数。

int86( )函数的一般形式是：

```
int int86(imt num, /* the interrupt number */
          union REGS *inregs, /* the input register values */
          union REGS *outregs) /* the output register values */
```

int86( )的返值在寄存器AX中。头文件BIOS.H中有联合体REGS的原型。这里所示的联合体REGS是在Turbo C的头文件中定义的，它与在Microsoft C的头文件中定义的联合体REGS的形式是相似的。

```
/*
    copyright (c) Borland International 1987, 1988, 1990, 1991
    All Rights Reserved.
    */
struct WORDREGS {
    unsigned int    ax, bx, cx, dx, si, di, cflag, flags;
};
struct BYTEREGS {
    unsigned char    al, ah, bl, bh, cl, ch, dl, dh;
};
```

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

如你所见，联合体REGS中包含有两个结构体：WORDREGS和BYTER-EGS。使用结构体WORDREGS，你能控制的CPU寄存器是16位的，如AX，BX等；BYTEREGS结构体则能让你使用的CPU寄存器是8位的，如AH，AL，BH，BL等。例如，为了获取INT10的第5号功能，你可以使用这样的代码序列：

```
union REGS in, out;
in.h.ah=5;
int86(0×10, &in, &out);
```

2 保存部分屏幕

为了保存部分屏幕，你就要把在该区域中当前位置的字符读出与储存起来。从屏幕上指定位置读取一个字符，使用的是INT10的第8号功能，它返回光标所在位置的字符本身及其属性。所以，为了从屏幕上指定位置读取字符，首先你要有确定光标位置的方法。虽然大多数C编译系统中有实现这个功能的函数，但个别的还是没有。函数goto_xy( )就可以满足那些无此函数的编译系统的需要。这个函数使用BIOS INT10的第2号功能，寄存器DL中放的是列坐标，DH中存放的是行坐标，视频页存放在BH中（通常使用默认的第0页）。

```
/*Send the cursor to specified X, Y coordinates. */
void goto_xy(int x, int y)
{
  union REGS r;
  r.h.ah=2; /* cursor addressing function */
  r.h.dl=x; /* column coordinate */
  r.h.dh=y; /* row coordinate */
  r.h.bh=0; /* video page */
  int86(0×10, &r, &r);
}
```

使用INT10的第8号功能时，把需要使用的视频页放在BH中。调用这个中断以后，AL中返回的是屏幕上光标当前位置的字符本身，AH中返回的是字符属性。函数save_video( )读取由其调用参数确定的部分屏幕，把读取的信息存放在一个缓冲区中，同时清除这部分屏幕。函数save_video如下所示：

```
/* Save a portion of the screen. */
void save_video(int startx, int endx, int starty, int endy,
                unsigned int *buf_ptr)
{
  union REGS r;
  register int i, j;
  for(i=startx; i<endx; i++)
    for(j=starty; j<endy; j++){
```

```
        goto_xy(i, j);
        r.h.ah=8; /* read character */
        r.h.bh=0; /* assume active display page is 0 */
        *buf_ptr++=int86(0x10, &r, &r); /* save in buffer */
        putchar=' '); /* clear the acreen */
      }
  }
```

Save_Video( )的前4个参数传递屏幕需要保存部分的左上角和右下角坐标。参数buf-ptr是一个指向容纳屏幕当前信息的存储区域的指针。这个存储区域应该足够大，能够容纳从屏幕上读取的信息。

本章中的程序使用的是动态分配内存的方法，但你可以用其它的分配内存的方法——如果它能使特定的应用更有意义的话，请记住，在屏幕恢复以前，你一定要保留已经分配好的缓冲区。函数Save_Video( )通过向每个位置写空格字符使屏幕清除。

3 幕屏复恢

菜单命令被选择以后，只要把先前保留的屏幕信息写回到视频RAM，就可以使屏幕得到恢复。BIOS INT10的第9号功能实现这个任务。使用时需要把字符本身装入AL，字符属性装入BL，视频页放入BH，字符写的次数放入CX（本章中一律取1）。函数restore_video( )把指针buf-ptr指向的缓冲区的信息映象到给定了起点和终点坐标的屏幕区域上：

```
/* Restore a portion of the screen. */
Void restore_video(int startx, int endx, int starty, int endy, unsigned char *buf_ptr)
{
  union REGS r;
  register int i, j;
  for(i=startx; i<endx; i++)
    for(j=starty; j<endy; j++){
      goto_xy(i, j);
      r.h.ah=9; /* write character */
      r.h.bh=0; /* assume active display page is 0 */
      r.x.cx=1; number of times to write the character */
      r.h.al=*buf_ptr++; /* character */
      r.h.bl=*buf_ptr++; /* attribute */
      int86(0x10, &r, &r, );
  }
}
```

1.4生成弹出式菜单

对于一个生成弹出式菜单的函数，需要你向它传递这么几条信息。首先是菜单选择项表。菜单选择项是要被显示的字符串，所以最简单的传递菜单选择项表的方法是把菜单中的各个选择项放入一个二维数组中，并且定义一个指针，使其指向这个数组。正如前面已经讲过的那样，下面两种方法中的任何一个都可以确定菜单的选择项。一种方法是把光标移动到选择项，然后按回车键；另一种方法是按一个热键。为使这个函数明白那些键是“热”的及

其含义，也需要把你事先定义的热键传递给这个函数。最好的办法是把这些热键字符组成的字符串传递给这个函数，其中热键字符串的排列顺序与菜单选择项表中字符串的排列顺序应是一致的。

pop_up( )函数需要知道菜单选择项表中有几个选择项，所以反映选择项条数多少的数值也需要向它传递。pop_up( )函数也需要知道把这个菜单放在屏幕上的什么位置，因此菜单的左、上角位置的X和Y坐标是需要的。最后，创建一个菜单边框在某些情况下也是需要的，当然，有的情况下是不需要这样一个边框的。所以，反映边框关/闭信息的参数也应该传向函数pop_up( )。从上面的分析可以看出，函数pop_up( )具有如下的原型：

```
int popup(
  char *menu[], /*menu text*/
  char *keys, /*hot keys*/
  int count, /*number of menu items*/
  int x, int y, /*X, Y coordinates of left-hand corner*/
  int border /*no border if 0*/
)
```

函数pop_up( )应该实现下列功能：

- 储存被菜单占用的屏幕
- 如果需要的话，显示菜单边框
- 显示菜单内容
- 输入用户的响应
- 把屏幕恢复到调用菜单以前的状态

这5项功能中的前两项在前述部分已经实现了，即用于保存屏幕的函数save_video( )和用于恢复屏幕的函数restore_video( )分别完成了前两项任务。下面让我们看看其它各项功能是如何实现的，

1.显示菜单内容

显示菜单内容的关键在于向一个字符串指针数组传递一个指针。为了显示单个字符串，你可以把这个指针指明为一个指针数组。这个数组中的每一项是指向相应的菜单项目的一个指针。下面所示的函数display_menu( )，就是采用这种方法显示每个菜单的内容的：

```
/*Display the menu in its proper location.*/
void display_menu(char *menu[], int x, int y, int count)
{
  register int i;
  for(i=0; i<count; i++, y++){
    goto_xy(x, y);
    printf(menu[i]);
  }
}
```

如你所见，这个函数把一个指针指向了被显示的字符串数组。X、Y是字符串左边起始位置的坐标，菜单中选择项的数目是通过参数Count传递的。

创建容纳菜单选择项字符串的二维数组的最容易的方法是定义一个全局变量，这个全局变量有如下的形式：

```
char * <menu-name>[]={
   "first selection" ,
   "second selection" ,
   .
   .
   .
   "Nth selection"
}
```

这种定义方式可自动地使C编译器在程序运行过程中，每当遇到该全局变量时，使其指向字符串表中的第一个字符串中的第一个字符。例如，下面的说明定义了一个叫fruit的指针变量，它指向“Apple”中的A：

```
char * fruit[]={
    "Apple" ,
   "Orange" ,
    "Pear" ,
   "Grape" ,
   "Raspberry" ,
   "Strawberry"
};
```

2.显示边框

如果需要一个边框的话，下面的函数能使一个给定了左上角坐标和右下角坐标的边框围住菜单。边框使用的是水平线字符和垂直线字符，它们是PC机或其兼容机里的标准字符。假如你喜欢的话，也可以选择其它字符做成你的边框。函数draw_border( )如下所示：

```
/* Draw a border around the menu. */
void draw_border( int startx, int starty, int endx, int endy )
{
  register int i;
/* draw vertical lines */
for( i=starty+1; i<endy; i++ ){
  goto_xy( startx, i );
  putchar( 179 );
  goto_xy( endx, i );
  putchar( 179 );
}
/* draw horizontal lines */
for( i=startx+1; i<endx; i++ ){
  goto_xy( i, starty );
  putchar( 196 );
  goto_xy( i, endy );
  putchar( 196 );
}
```

```
/* draw the corners */
goto_xy( startx, starty ); putchar( 218 );
goto_xy( startx, endy ); putchar( 192 );
goto_xy( endx, starty ); putchar( 191 );
goto_xy( endx, endy ); putchar( 217 );
}
```

3 输入用户响应

用户可以采用两种方法中的任何一种输入响应。第一种方法是使用 UP ARROW 和 DOWN ARROW 键移动光标到指定位置，然后按回车键，这样用户的响应就被输入了进去。(这里设计的菜单能使被光标选中的菜单选择项反视频显示)。SPACEBAR 键也可以用来移动光标。第二种方法是按与菜单选择项对应的一个热键。这里看到的函数 get_resp ( )，它的任务就是为了完成输入用户的相应这个要求的：

```
/* Input user's selection. */
get_resp( int x, int y, int count, char * menu[], char * keys )
{
  union inkey {
  char ch[2];
  int i;
} c;
int arrow_choice=0;
char * key_choice;
x++;
y++;
/* highlight the first selection */
goto_xy( x, y );
write_video( x, y, menu[0], REV_VID ); /* reverse video */
for( ; ; ) {
  c.i=readkey( ); /* read the key */
  /* reset the selection to normal video */
  goto_xy(x, y+arrow_choice);
  write_video( x, y+arrow_choice,
               menu[arrow_choice], NORM_VID ); /* redisplay */
  if( c.ch[0] ){/* is normal key */
    /* see if it is a hot key */
    key-choice=strchr( keys, tolower( c.ch[0] ) );
    if( key-choice )return key-choice-keys;
    /* check for Enter or spacebar */
    switch( c.ch[0] ){
      case '\r': return arrow_choice;
      case ' ': arrow_choice++;
        break;
      case ESC: return -1; /* cancel */
    }
  }
```

```
    else { /* is special key */
      switch( c.ch[1] ){
        case 72: arrow_choice--; /* up arrow */
          break;
        case 80: arrow_choice++; /* down arrow */
          break;
      }
    }
    if( arrow_choice==count ) arrow_choice=0;
    if( arrow_choice<0 ) arrow_choice=count-1;
      /* highlight the next selection */
      goto_xy( x, y+arrow_choice );
      write_video( x, y+arrow_choice, menu[arrow_choice], REV_VID );
    }
  }
```

当get_resp( )开始执行时，第一个菜单选择项被点亮。宏REV_VID的值为0x70，NORM_VID的值为7。当第一个菜单被点亮时，这个函数进入循环并等待用户的响应。首先，它通过函数read key( )读入一个击键。函数read key( )如下所示：

```
/* Return the 16-bit scan code from the keyboard. */
readkey( void )
{
  union REGS r;
  r.h.ah=0;
  return int86( 0x16, &r, &r );
}
```

函数readkey( )使用BIOS中断INT 0x16的第0号功能。当某个键被敲击时，它将返回由键盘产生的16位扫描码（INT0x16是BIOS键盘服务的入口）。每次按一个键，产生两个8位代码，通过编码生成一个16位整数。如果击的是一个字符键，那么扫描码的低8位返回这个字符的键盘编码。如果敲击的是一个特殊键，例如一个ARROW键，那么低位字节的扫描码为0，高位字节存放的是它的位置编码。UP ARROW键和DOWN ARROW键的位置扫描码分别是72和80。像getchar( )这样的标准C函数只能返回字符扫描码，所以不采用它而用readkey()直接获取扫描码C是很有必要的(有些编译系统的库函数中提供了类似于readkey()的函数。例如，Turbo中类似的函数名叫做bioskey( )。如果你的编译器中已经提供了类似的函数，则使用起来是很方便的。)

每次按一个ARROW键，则先前被点亮的菜单选择项转为正常显示状态，而光标移到的项被点亮。当光标已经位于菜单的最下面的一个选择项，这时你按DOWN ARROW键，光标又将回到菜单的最上面的一个选择项。如果菜单最上面的选择项已经被点亮，这时你按UP ARROW键，光标将移到菜单最下面的一个选择项。按SPACEBAR键的效果与按DOWN ARROW键类似，但你还是要注意到它们之间是有差别的。按ESC键，则将退出菜单。ESC的值为27。

get_resp( )使用函数write_video( )向屏幕指定位置写一个具有特定属性的

字符串。write_video( )使被点亮的菜单选择项反视频显示，它也可以使菜单选择项呈正常显示方式。函数write_video( )的源代码如下所示：

```
/* Display a string with specified attribute. */
void write_video( int x, int y, char *p, int attrib )
{
  union REGS r;
  register int i;
  for( i=x; *p; i++ ){
    goto_xy( i, y );
    r.h.ah=9; /* write character */
    r.h.bh=0; /* assume active display page is 0 */
    r.x.cx=1; /* number of times to write the character */
    r.h.al=*p++; /* character */
    r.h.bl=attrib; /* attribute */
    int86( 0X10, &r, &r );
  }
}
```

4 popup( )函数

函数popup( )所要求的功能都已经实现了，把前面设计的各个函数集合在一起，就形成了一个完整的popup( )函数：

```
/* Display a pop-up menu and return selection.
   This function returns -2 if menu cannot be constructed;
   it returns -1 if user hits escape key;
   otherwise the item number is returned starting
   with 0 as the first ( topmost ) entry.
*/
int popup(
   char *menu[ ], /* menu text */
   char *keys, /* hot keys */
   int count, /* number of menu items */
   int x, int y, /* X, Ycoordinates of left-hand corner */
   int border  /* no border if 0 */
)
{
register int i, len;
int endx, endy, choice;
unsigned int *p;
if((y>24)||(y<0)||(x>79)||(x<0)){
  printf( "range error" );
  return -2;
}
/* make sure that the menu will fit */
len=0;
for( i=0; i<count; i++ )
```