C语言常用算法与子程序

尹彦芝 等 编著

清 华 大 学 出 版 社

目　录

第一章　Pop-up 和 Pull-down 菜单

在专业程序员编写的 C 语言程序中，经常使用 Pop-up 和 Pull-down 菜单。如果这些菜单使用得好，会给用户一种清晰明快而又神秘的感觉。从概念上来讲，Pop-up 和 Pull-down 菜单并不复杂，但真正实现起来并不是那么容易。

为了建立 Pop-up 和 Pull-down 菜单，需要对屏幕进行直接控制。虽然实际的菜单程序是非常容易移植的，但从本质上来讲，对屏幕进行操作的程序必然是与硬件和操作系统有关的，并必须不使用 C 的正常的控制台输入输出函数。这里开发的对屏幕进行操作的程序可以在任何使用 MS-DOS 的 IBM PC 及其兼容机上运行。其所以选择 MS-DOS 和 IBM PC 机，是因为这种操作系统和机器是使用最广的。但是，菜单的基本概念是一样的，可以很容易地推广到其它操作系统或其它计算机上。

即使你对 Pop-up 和 Pull-down 菜单不感兴趣，也应该读一读这一章，尤其是"显示适配器简介"这一节，因为这些内容是理解以后许多章节所必要的一些基本概念。

1.1　什么是 Pop-up 和 Pull-down 菜单

什么是 Pop-up 和 Pull-down 菜单？它们与标准菜单有什么区别？当使用标准菜单时，往往是首先清除屏幕或滚动屏幕，然后再把菜单显示到屏幕上。当用户选择菜单中的某一项以后，屏幕再次被清除或滚动。然后，或者是出现下一个菜单，或者是按用户的选择继续运行下去。用户的选择是通过打入选择项的号码或打入选择项的第一个字母来进行的。

当 Pop-up 或 Pull-down 菜单被激活时，它不是去清除屏幕，而是改写当前屏幕上的内容。在用户进行了选择以后，屏幕上被改写的部分又恢复到它的原来的内容。用户进行选择的办法除打入选择项的号码或选择项的第一个字母以外，还可以使用光标移动键来加亮用户所需要的选择项，然后再打入回车键。

标准菜单与 Pop-up 或 Pull-down 菜单之间的关键性的差别在于"激活"。标准菜单会清除屏幕上所有原来的内容，给用户的感觉似乎是停止了原来程序的运行；而激活一个 Pop-up 或 Pull-down 菜单，看起来只是暂时挂起原来的程序。从实际产生的效果来看，标准菜单会把用户注意力引向一个新的方面，而 Pop-up 和 Pull-down 菜单只是暂时中断一下，而用户的注意力仍集中在原来程序的运行上。

Pop-up 菜单和 Pull-down 菜单之间的差别则很简单。屏幕上任何时候只能出现一个 Pop-up 菜单。当菜单只有一级深度，也就是说菜单上每个选择项不再有子菜单时，就应使用 Pop-up 菜单。而 Pull-down 菜单则不同，屏幕上可以同时有好几个 Pull-down 菜单。当菜单的深度超过一级时，就应使用 Pull-down 菜单。例如，一个选择水果的程序，第一级菜单可能是各种水果名称，用户如果选定要苹果，第二级菜单可能是苹果的颜色。用户如果选定要红色，第三级菜单可能是苹果的产地，等等。

当然，可以把 Pop-up 菜单想像为没有任何子菜单的特殊的 Pull-down 菜单。但是把 Pop-up 和 Pull-down 菜单分别考虑还是比较合适的，这是因为 Pull-down 菜单比简单的 Pop-up 菜单需要更多的开销。

在本章开发的程序中，菜单中的每一个选择项在屏幕上占一行，第一项在最上面。

1.2 显示适配器简介

因为 Pop-up 和 Pull-down 菜单的建立需要对屏幕进行直接控制，所以必须对显示适配器有所了解。至今为止，有四种显示适配器是使用最广的，这就是：MDA、CGA、EGA、VGA。MDA 只有一种显示方式，如表 1-1 所示。本章开发的菜单程序都是使用每行 80 列的文本方式，这是一般应用程序使用最广的一种文本方式。因为选定每行 80 列，所以显示方式只能是 2，3 或 7。

表 1-1 显示方式

显示方式	显示类型	屏幕尺寸	字模	颜色数	页数	显示适配器
0	文本	200×320,25×40	8×8	16	8	C,E
1	文本	350×320,25×40	14×8	16／64	8	C,E
2	文本	200×640,25×80	8×8	16	8	C,E
3	文本	350×640,25×80	14×8	16／64	8	C,E
4	图形	200×320,25×40	8×8	4	1	C,E
5	图形	200×320,25×40	8×8	4	1	C,E
6	图形	200×640,25×80	14×8	2	1	C,E
7	文本	350×720,25×80	14×9		8	M
13	图形	200×640,25×40	8×8	16	2(64K)	E,V
					4(128K)	E,V
					8(256K)	E,V
14	图形	200×640,25×80	8×8	16	1(64K)	E,V
					2(128K)	E,V
					4(256K)	E,V
15	图形	350×640,25×80	14×8	4	1(64K)	E,V
					2(128K)	E,V
16	图形	350×640,25×80	14×8	4／64(64K)	1(128K)	E,V
				16／64(128K)	2(256K)	E,V
17	图形	480×640,30×80	16×8	2／256K	1	V
18	图形	480×640,30×80	16×8	16／256K	1	V
19	图形	200×320,25×40	8×8	256／256K	1	V

表 1-2 显示属性

位	7	6	5	4	3	2	1	0
二进制值	128	64	32	16	8	4	2	1
颜色	闪烁	红	绿	蓝	加亮	红	绿	蓝
	背景				前景			

屏幕上显示的字符都保存在显示适配器上的显示 RAM 中。在 MDA 中，显示 RAM 的起始地址是 B000：0000H，在 CGA 和 EGA 中，显示 RAM 的起始地址是 B800：0000H。虽然在某些显示方下，CGA 和 EGA 的功能有些不同，但在显示方式 2 和 3 下，CGA 和 EGA 是完全一样的。

在屏幕上显示的每一个字符要求在显示 RAM 中占两个字节。第 1 个字节用来保存字符的 ASC II 值，第 2 个字节用来保存字符显示属性。对于 CGA 和 EGA，属性字节各位的意义如表 1—2 所示。对于 CGA 和 EGA，缺省的显示方式是 3，缺省的显示属性是 7，这就是产生每行 80 列的黑底白字。为了加强显示效果，强调某些字符，则可以选用显示属性 70H，产生的结果是白底黑字。

MDA 不具备彩色，但却识别闪烁位（位 7）和加亮位（位 3），还能产生带下划线的字符（显示属性为 1）。幸运的是，MDA 也是把显示属性 7 解释为黑底白字，显示属性 70H 解释为白底黑字。

CGA 和 EGA 适配器上实际具有的显示 RAM 比保存 25×80 个字符所需要的 RAM 要多出好几倍，这是由于以下两方面的原因造成的。第一，适配器在图形方式下需要更多的 RAM。第二，显示 RAM 超出几倍，就可以多保存几个屏幕的信息，屏幕之间的切换就变得很简单。保存一屏幕字符所需要的 RAM 称为一页。缺省情况下，DOS 在启动时总是使用第 0 页，几乎所有的应用程序也都使用第 0 页，本章开发的各程序也都是使用第 0 页。然而，如果需要的话，也可以改为使用其它页。

对显示屏幕进行操作的办法有三种。第一种是通过 DOS 的系统调用，这种办法对 Pop-up 和 Pull-down 菜单实在是太慢了，不能采用。第二种办法是通过 ROM-BIOS，这种办法比较快，如果菜单比较小，在 286 或 386 这些速度较快的机器上用这种办法还是足可以的。第三种办法就是直接存取显示 RAM，这种办法最快，但编程相对困难一些。本章所列出的两个完整程序分别使用第 2 种和第 3 种办法。

1.3 通过 BIOS 对屏幕进行操作

因为 Pop-up 和 Pull-down 菜单必须首先把菜单将要占用的那一部分屏幕的内容保存起来，在用户进行了选择以后又必须把这一部分内容恢复到屏幕上，所以必须有一个子程序来做这个保存和恢复的工作，这一节先介绍通过 BIOS 来做保存和恢复工作的子程序。

通过 BIOS 对屏幕操作可能会很慢，但是这种办法不要求计算机的硬件百分之百地与 IBM PC 机兼容，而只要求它的 BIOS 与 IBM PC 机的 BIOS 兼容。因此，如果速度能够满足要求，则还是尽可能采取这种办法，而不要采取后面将要介绍的速度更快但对硬件兼容要求更高的直接存取显示 RAM 的办法。

1.3.1 使用 INT86( ) 函数

在 Microsoft C 和 Turbo C 中调用 ROM-BIOS 中软中断的办法是通过 INT86( ) 函数。在其它 C 编译中，也都提供了类似的函数。

INT86( ) 函数的一般格式如下：

```
int int86(num,inregs,outregs)
```

```
int num; /* the interrupt number */
union REGS *inregs; /* the input register values */
union REGS *outregs; /* the output register values */
```

INT86() 函数返回的是 AX 寄存器中的值。在上面的格式中，联合 REGS 的说明是在文件 DOS.H 中。以 Turbo C 为例（Microsoft C 和其它 C 编译是类似的），REGS 的定义如下：

```
struct WORDREGS
  {
  unigned int ax,bx,cx,dx,si,di,cflag;
  };
struct BYTEREGS
  {
  unsiged char al,ah,bl,bh,cl,ch,dl,dh;
  };
union REGS {
  struct WORDREGS x;
  struct BYTEREGS h;
  };
```

从这个定义中可以看出，REGS 是两个结构的联合。WORDREGS 结构是把 CPU 的寄存器按 16 位进行操作，BYTEREGS 结构是把 CPU 寄存器按 8 位进行操作。例如，为了调用软中断 10H 中的子功能号 5，可以用下列几条语句。

```
union REGS in,out;
in.h.ah=5;
int86(16,&in,&out);
```

1.3.2 保存屏幕

为了把屏幕的内容保存下来，必须读入每一个屏幕位置的当前值并存储起来。中断 10H 的子功能号 8 可以读入在当前光标位置处的字符的 ASCII 码及其显示属性。因此，为了读入屏幕上某一特定位置处的一个字符，必须首先把光标移到这位置。虽然某些 C 编译提供了这样一个移动光标的函数，但大部分 C 编译却不提供。为了方便移动光标的工作，这里提供了一个 goto-xy() 函数。这个函数使用了中断 10H 的子功能号 2，调用时要求寄存器 DL 存有光标的列号，DH 存有光标的行号，BH 存有显示页号。

```
* send the cursor to x,y */
void goto_xy(x,y)
```

```
int x,y;
{
  union REGS r;

  r.h.ah=2; /* cursor addressing function */
  r.h.dl=y; /* column coordinate */
  r.h.dh=x; /* row coordinate */
  r.h.bh=0; /* video page */
  int86(0x10,&r,&r);
}
```

中断 10H 的子功能号 8 返回当前光标位置下字符的 ASCII 码(在 AL 寄存器中) 和显示属性 (在 AH 寄存器中)。下面的函数 Save_video ( )把屏幕的一部分读入内存的一个缓冲区中，并清除这部分屏幕。

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

  for(i=starty; i<endy; i++)
    for(j=startx;j<endx; j++) {
      goto_xy(j, i);
      r.h.ah = 8; /* read character function */
      r.h.bh = 0; /* assume active disoplay page is 0 */
      * buf_ptr++ = int86(0x10, &r, &r);
      putchar(' '); /* clear the screen */
  }
}
```

1.3.3 恢复屏幕

一旦用户对菜单进行了选择，就需要恢复屏幕。所谓恢复屏幕，就是把原先保存下来的信息再写回到显示 RAM 中去。中断 10H 的子功能号 9 可以用来做这件事情。在调用它之前，要求把字符的 ASCII 码放在 AL 寄存器内，字符的显示属性放在 BL 内，显示页号放在 BH 内，重复次数放在 CX 内。下面的函数 restore_video 把由指针 buf_ptr 所指向的内存中的内容写到屏幕上去，从指定的屏幕的左上角开始到指定的右

下角结束。

```
/ * restore a portion of the screen * /
void restore_video(startx, endx,starty, endy,buf_ptr)
int startx, endx,starty, endy;
unsigned char * buf_ptr;
{
  union REGS r;
  register int i, j;
  for(i = starty; i <= endy; i++)
    for(j = startx; j < endx; j++) {
      goto_xy(j, i);
      r.h.ah = 9; / * write character function * /
      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 建立 Pop-up 菜单

一般来说，一个建立 Pop-up 菜单的函数必须能接受下列几个参数：菜单在屏幕上的位置，菜单的内容，菜单项数，选择菜单的热键，是否需要在菜单上加边框，等等。菜单的内容一般总是几个字符串，因此把这些字符串处理成一个二维数组并把指向这个数组的指针传递给建立 Pop-up 菜单的函数是最方便的。前面已经说过，用户选择一个菜单项通常有两种办法，一种办法是用移动光标键来加亮所需要的选择项并打入回车键，另一种办法是打入与那个菜单项相对应的单个字母键，这个键就称为热键。每个菜单项有一个热键，这些热键合起来就构成一个字符串，把指向这个字符串的指针传给建立 Pop- up 菜单的函数就可以了。下面就是这个函数及其参数的说明。

```
/ * display a pop_up menu and return selection * /
int popup(menu, keys, count, x, y, border)
char * menu[ ]; / * menu text * /
char * keys; / * hot keys * /
int count; / * number of menu items * /
int x, y; / * x, y coordinates of left hand corner * /
int border; / * no border if 0 * /
```

概括起来讲，建立 Pop—up 菜单的函数必须做如下几件事情：

①把菜单将要用到的那一部分屏幕内容保存起来。

②如果需要，显示菜单的边框。

③显示菜单的内容。

④接受用户的选择。

⑤把屏幕恢复到原来的情况。

在这 5 步中，第 1 步和第 5 步所要用到的程序在上一节中已经讨论过了，下面接着讨论实现其它 3 步的办法。

1.4.1 显示一个菜单

如上所述，传递给 Pop—up 函数的是一个指向字符串指针数组的指针。即，这个指针是指向一个数组，这个数组的每一个元素又是一个指针，分别指向构成菜单内容的各个字符串。因此，为了显示某一个字符串，就可以像处理数组元素一样，先从数组中找到指向这个字符串的指针，然后调用 printf 函数去显示就可以了，如下所示。

```
/ * display the menu in its proper location * /
void display_menu(menu, x, y, count)
char * menu[ ];
int x, y, count;
{
  register int i;

  for(i = 0; i < count; i++, x++) {
    goto xy(x,y);
    printf(menu[i]);
  }
}
```

在 C 语言中，用一个二维数组来存储多个字符串的最简单的办法就是用如下这种定义形式：

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

凭这条说明语句，C 编译就会把这些字符串放在它的实时字符串表中，并建立一个变量 fruit，这个变量指向第一个字符串（“Apple”）的第一个字符（“A”）。

1.4.2 显示菜单的边框

如果需要，可以在菜单的四周加上一个边框。下面这个程序就是用于画边框的。这个程序是用 IBM PC 及其兼容机所支持的一些特殊形状的图形字符来画边框的，而不是用图形方式来画边框的。

```
void draw_border(startx, starty, endx, endy)
int startx, starty, endx, endy;
{
  register int i;
  for(i = startx+1; i < endx; i++) {
    goto_xy(i, starty);

    putchar(179);
    goto_xy(i, endy);
    putchar(179);
  }

  for(i = starty+1; i < endy; i++) {
    goto_xy(startx, i);
    putchar(196);
    goto_xy(endx, i);
    putchar(196);
  }
  goto_xy(startx, startx); putchar(218);
  goto_xy(startx, endy); putchar(191);
  goto_xy(endx,starty); putchar(192);
  goto_xy(endx, endy); putchar(217);
}
```

1.4.3 接受用户的选择

如上所述，用户可以通过两种办法对菜单进行选择。一种办法是上下移动光标键，加亮所需要的选择项，再打入回车键表示认可。所谓加亮，本书中指的都是反像显示，即把黑底白字改为白底黑字。用户也可以用空格键来上下移动光标，第二种办法是通过打入热键来进行选择。下面这个 get_resp 函数同时实现了这两种办法。

```
/ * input user's selection * /
get_resp(x, y, count, menu, keys)
```

```
int x, y, count;
char * menu[ ];
char * keys;
{
  union inkey {
    char ch[2];
    int i;
  } c;
  int arrow_choice=0, key_choice;
  y++;
  / * highlight the first selection * /
  goto_xy(x, y,);
  write_video(x, y, menu[0],REV_VID); / * reverse video * /
  for(;;) {
    while(!bioskey(1)); / * wait for key stroke * /
    c.i = bioskey(0); / * read the key * /
    / * reset the selection to normal video * /
    goto_xy(x+arrow_choice, y);
    write_video(x+arrow_choice, y,
      menu[arrow_choice],NORM_VID); / * redisplay * /
    if(c.ch[0]) { / * is normal key * /
      / * see if it is a hot key * /
      key_choice = is_in(keys, tolower(c.ch[0]));
      if(key_choice) return key_choice-1;
      / * check for ENTER or space bar * /
      switch(c.ch[0]) {
        case '\r': return arrow_choice;
        case ' ': arrow_choice++;
          berak;
        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;
      }
```