HOPE

HOPE COMPUTER COMPANY LTD.

C 语言高级程序员编程指南

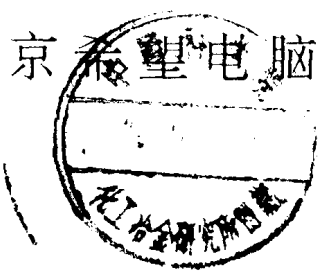
鸿健 编

北京希望电脑公司

C 语言高级程序员
编程指南

鸿 健 编

北京希望电脑公司



前　言

如果您想用 C 编写具有世界一流水平的程序，那么这本书将使您如愿以偿！

我叫 Herb Schildt，用 C 编程已有十多年，算是一名身经百战的老程序员了。在我着手编写此书前，我查阅了大量的成功软件产品，试图确定它们所具有的、而不太成功的程序所没有的特点。我想知道促使一件产品比另一件类似产品更成功的原因。经过一段时间的研究之后，我找到了一条线索。成功的产品是由这样的人设计出来的，他不仅充分了解某项应用，而且完全熟悉整个计算机的环境，包括操作系统和硬件本身。只有完全掌握了这些的程序员，才能够编写出漂亮的用户接口，可以高效地执行并向用户提供最大的灵活性。

本书揭示了老练的程序员所采用的、可以达到专业水平的许多秘诀。本书中，你将探求让程序发声的技术和方法。这样，你可以编写需要引起人们注意的程序。本书内容包括：

- ☐ 用于快速屏幕显示的直接视频存贮器访问。
- ☐ 弹出式(Pop-up)和下拉式(Pull-down)菜单
- ☐ 弹出式窗口例程
- ☐ 终止并驻留程序
- ☐ 鼠标器接口
- ☐ 图形函数，其中包括物体的旋转
- ☐ 语言解释器
- ☐ 通过串行口传送文件

本书是为所有的 C 程序员编写的，从初学者到专业人员均可。即使你是一名初学者，你仍可以使用本书提供的函数及程序，而不必了解它们的全部操作细节。水平稍高的读者可以将这些例行程序用作为其应用的基础。

除使用了一些 PC 专用函数的地方之外，本书中的源程序符合建议中的 ANSI 标准。因此，所有的例行程序可以在任何支持 ANSI 标准的编译器上编译。为了有利于进一步的开发，这里采用 Turbo C 和 Microsoft C。

注：凡需本书源程序的用户，可直接向希望公司索取，共 1 张高密软盘，工本费 14 元。

目　录

第九章　绘制商业用直方图

第十章　与鼠标器的接口

第一章　弹出式和下拉式菜单

由专业人员编写的程序，其最显著的标志是使用弹出式(pop-up)或下拉(pull-down)菜单。若能正确地实现，这种菜单将会提供给用户那种所希望的生动感。虽然这种菜单概念简单，但它们却呈现出很强的编程需要。

产生弹出式和下拉式菜单，要求对屏幕进行直接控制。由于实际的菜单程序可移植性极强，但访问屏幕的程序都与硬件及操作系统有内在联系，因而我们必须避免调用 C 语言中普通的控制台 I/O 函数。本书开发的视频访问例行程序，可工作于任何使用 DOS 的计算机，并采用 IBM 兼容的 ROM－BIOS 作为它的操作系统。之所以选择 DOS－BIOS，是因为它使用最广，但我们也可以将这些基本概念推广到别的系统中去。

即使现在你对弹出式或下拉式菜单不感兴趣，你仍应该阅读本章关于视频适配器部分，因为许多基本概念对理解以后各章很有必要。

1.1　什么是弹出式和下拉式菜单

了解弹出式和下拉式菜单，以及它与普通菜单的区别是很重要的。当我们使用普通菜单时，屏幕要么被清除，要么滚动，而后菜单才出现，若我们选了另一项，屏幕再次被清除或滚动，然后程序继续下去。我们采用数字编号或选项的英文字头进行菜单选择。

当弹出式或下拉式菜单被激活后，就覆盖了当前屏幕上的显示。选项完成后，屏幕就恢复成原来的状态。从弹出式和下拉式菜单中选一个项，可以采用以下两种方式中的一种：(1) 按下一个“热”键(hot key)，它是一个与各种菜单选项相关的字母或数字；或(2) 用箭头键移动醒目部分，直到所要的选项，然后按下 ENTER 键。一般地，醒目的选项是以反显出现的。

普通菜单和弹出式和下拉式菜单的主要区别在于：激活一个普通菜单将中止程序的运行；而弹出式和下拉式菜单却是“挂起”当前运行的程序。从用户角度看，普通菜单打断了他的注意力，而弹出式或下拉式菜单只轻微地打扰，不影响其注意力。

弹出式和下拉式这两种菜单的区别很简单。屏幕上任何时刻只可出现一个弹出式菜单。当菜单只有一级深度，即没有再次选择时，使用该菜单。相反，几个下拉式菜单可以同时显示。它们用于这种情况，即选择上一级菜单时，需要由下一级菜单来确定某些选项。例如，欲采用下拉式菜单设计一个订购水果的程序。如果用户选择了“苹果”，下一级菜单将显示苹果的颜色，第三级菜单将显示符合前面要求的苹果。

我们可以简单地把弹出式菜单，看作是一个没有子菜单的下拉式菜单。然而，将这两种菜单分别考虑有其优点，因为比起简单的弹出式菜单，编写下拉式菜单需要更多的开销。

在屏幕上构造菜单有许多方法，但本章开发的函数采用了最普遍的形式，也就是将各菜单项分别放在各行上，第一项在顶端。如图 1－1 所示的简单的发货单菜单。

1.2　视频适配器简介

由于产生弹出式和下拉式菜单需要对屏幕进行直接控制，因此很有必要了解一下视频

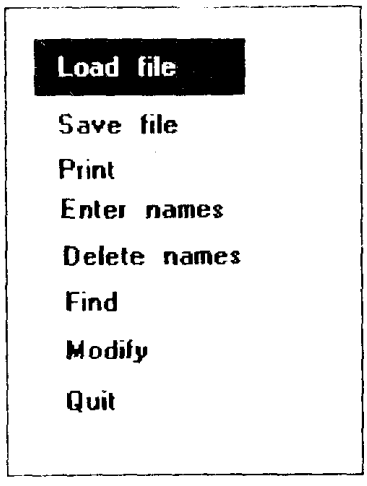


图 1－1　菜单示例

显示适配器。最常见的三种适配器为：单色适配器、彩色/图形适配器(CGA)以及增强型图形适配器(EGA)。其中，CGA 和 EGA 具有几种操作方式，包括 40 列或 80 列文本操作，以及图形操作。这些方式如表 1－1 所示。本章所编写的菜单程序用于 80 列文本方式，这是至今一般性应用中最普遍操作方式。因此，系统的视频方式必须是 2、3 或 7，但无论采用何种方式，左上角坐标均为(0,0)。

表 1.1　各种适配器的屏幕方式

方式	类型	尺寸	适配器
0	文本黑白	40×25	CGA,EGA
1	文本 16 色	40×25	CGA,EGA
2	文本黑白	80×25	CGA,EGA
3	文 216 色	80×25	CGA,EGA
4	图形 4 色	320×200	CGA,EGA
5	图形四灰度级	320×200	CGA,EGA
6	图形黑白	640×200	CGA,EGA
7	文本黑白	80×25	单色适配器
8	图形 16 色	160×200	PCjr
9	图形 16 色	320×200	PCjr
10	图形 4 色 PCjr，16 色 EGA	640×200	PCjr,EGA
13	图形 16 色	320×200	EGA
14	图形 16 色	640×200	EGA
15	图形 4 色	640×350	EGA

屏幕上显示的字符存于显示适配器的一些专用 RAM 中。对于单色适配器，视频存贮器起始地址为 B0000000H，而对于 CGA 和 EGA，视频存贮器的起始地址为 B8000000H(这样加以区分是为了能够分别使用图形和文本屏幕——但实际中却很少这样做)。虽然 CGA 和

EGA 在某些方式下功能不同，但在方式 2 或 3 下，其功能却一致。

屏幕上显示的各个字符在视频显示器中占两个字节。第一个字节放实际字符，第二个字节存放其屏幕属性。对于彩色适配器，属性字节的含义见表 1－2。CGA 和 EGA 的缺省操作方式为 3，且字符按属性字节值为 7 显示。于是，三种前景颜色都点亮，产生白色。若要产生相反的颜色，前景位均置零，而背景位均置 1，属性字节值为 70H。

视频属性字节

位	二进制值	意义
0	1	兰色前景
1	2	绿色前景
2	4	红色前景
3	8	低亮
4	16	兰色背景
5	32	绿色背景
6	64	红色背景
7	128	字符闪烁

单色适配器只识别闪烁和亮度位。但它能够将属性值 7 作为一般文本(黑底白字)，70H 作为反向显示。此外，属性值 1 可产生带下划线字符。

每个适配器具有的存贮空间，实际是显示 80 列方式的文本本所需空间的 4 倍。这里有两个原因。首先是将多余的存贮空间用于图形显示(单色适配器除外)，其次是为 能够将多个屏幕存于 RAM 中并在需要时取出(Switch in)。存贮器的每个区域称为一显示页(Video Page)。取出活动的显示页影响很大。在 DOS 初始化时，缺省使用第 0 页。实际上，所有的应用程序均使用第 0 页。因此本章将在例行程序中使用第 0 页，但也可以用其它页。

访问视频适配器有三种方式。第一种是通过 DOS 调用，但对于弹出式或下拉式菜单，这种方式太慢；第二种是通过 ROM－BIOS 例行程序，它们较快并且若菜单较小的话，在速度较快的机器上可能足够快，如 AT 或 PS/2 系列。第三种是通过直接读写视频存贮器，这种方式很快，但较复杂。本章给了两套独立的视频例行程序。其中之一是利用 ROM－BIOS，另一个是直接访问视频存贮器。

1.3 通过 BIOS 访问屏幕

由于弹出式或下拉式菜单函数在用户作出选择后，必须保存部分屏幕并恢复之，因此需要编写保存并装入部分屏幕的程序。本节采用的存贮和恢复部分屏幕的方法，是调用两个固化在 ROM－BIOS 中的读写屏幕上字符的函数。

众所周知，ROM－BIOS 调用速度很慢。但是，即使实际的显示器硬件不同，仍应保证它们工作于具有与 IBM 兼容的 ROM－BIOS 的计算机上。在后面的章节中，你将了解如何直接访问 IBM PC 机或其完全兼容机的视频存贮器，来增加其性能。然而，采用直接视频存贮器访问却在某种程序上削弱了程序的可移植性，因为它要求计算机与标准的 IBM PC 硬件完全兼容。因此，ROM－BIOS 菜单例行程序要用于要求最大可移植性的应用程序中。

1.3.1 使用 int86( )

采用软件中断的办法可以实现 ROM－BIOS 调用。RM－BIOS 具有几种用于不同目的的中断。其中，用于访问屏幕的是中断 16(10H)，用它可访问视频显示(如果你对访问 ROM－BIOS 的内容不太熟悉，可参阅原书作者的另一本书：《C：The Complete Reference》[Berkeley：Osborne/McGraw－Hill，1987]，从那里你将看到关于该问题的详细论述)。正如许多 ROM－BIOS 中断一样，中断 16 具有几种基于 AH 寄存器值的选择。如果函数有返回值，则需要其它寄存器。使用 C 函数 int86( )可以访问 ROM－BIOS 中断。有些编译器用其他名字称该函数，但 Microsoft C 及 Turbo C 均称之为 int86( )。下面的讨论是专门针对这些编译器的，但你应当能够推广它们)。

int86( )函数采取下面的一般形式：

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

返回值在 AX 寄存器中。类型 REGS 由开头的 DOS. H 提供。这里给出的类型 REGS 是由 Turbo C 定义的。但它与由 Microsoft C 及其他编译器定义的相似。

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

可见，REGS 是两个结构的联合。采用 WORDREGS 结构能够访问 16 位 CPU 寄存器，而 BYTEREGS 结构能访问 8 位寄存器。例如，访问中断 16、功能 5，可以利用下列编码序列：

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

1.3.2 保存部分屏幕

要保存屏幕上的内容，必须读取并存贮各屏幕位置上的当前值。用中断 16、功能 8 从某一屏幕位置读一个字符，将返回该字符及其在当前光标处的属性。因此，从屏幕的某一部分读字符需要确定光标位置。尽管一些 C 编译器提供了这一函数，但多数却没有提供。因此，可使用这里给出的函数 goto_xy( )。该函数使用中断 16、功能 2，纵坐标存于 DL 中，横坐标在 DH 中。显示页号在 BH 中指定(缺省页号为 0)。

```
/* 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 16、功能 8 返回当前光标位置处的字符(在 AL 中)及其属性(在 AH 上)。下面给出的函数 save_video( ),先读取部分屏幕,将其中的信息存入缓冲区,然后清除该部分屏幕。

```
/* save a portion of the screen */
void save_video(startx, endx, starty, endy, buf_ptr)
int startx, endx, starty, endy;
unsigned int *buf_ptr;  /*buffer where  screen will be store */
{
  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 display page is 0 */
         *buf_ptr++ = int86(0x10, &r,&r);
         putchar(' '); /* clear the screen */
       }
}
```

Save_Video 函数的头四个参数指出所保存区域的左上角及右下角的 X、Y 坐标。参数 buf_ptr 是一个整型指针。它指向保存信息的存贮区,该区域必须足够大,以保存从屏幕上读取的信息。

本章中的程序采用动态分配缓冲区,但如果对你的某项应用更有意义的话,也可以采取其他方案。但是要记住,缓冲区必须保留到屏幕恢复原来依态为止。函数 Save_Video 也通过在每个位置上逐个写空格,来清除指定区域。

1.3.3 恢复屏幕

菜单选择完毕,恢复屏幕只不过是将以前保存的信息写回到视频存贮器中。恢复屏幕,我们使用中断 16、功能 9,它要求字符放在 AL 中,其属性值在 BL 中,显示页号在 BH 中,写字符的次数在 CX 中(在本例中 1)。下面给出的 restore_video( )函数按照给出的起始和终止 X、Y 坐标,将 buf_ptr 指定的缓冲区中的信息放到屏幕上。

```
/* restore a portion of the sceen */
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++) {
     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 timrs to write the character */
     r.h.al = *buf_ptr++; /* chatacter */
     r.h.bl = *buf_ptr++; /* attribute */
     int86(0x10, &r, &r);
  }
}
```

1.4 产生弹出式菜单

有若干条信息必须传递给产生弹出式菜单的函数。首先是一个菜单选项表。由于菜单项是欲显示的字符串，因此将一字符串表传递给一个函数的最简单的办法，是将字符串放在一个二维数组中，传递指向该数组的指针。正如前面所提到的，选择一个菜单项，可以通过将醒目部分移到欲选项上，并按下 ENTER 键，或者直接按下对应该项的键。为使函数知道哪些"热"键，以及它们的含义，必须将这些键的名字传递给该函数。最好的办法是，按照菜单字符串一样的顺序，传递包含热键字符的字符串。

pop_up( )函数还需知道菜单中选项的个数，所以需要将项数传递给它。此外，该函数还需知道将菜单放到何处，因此需要 X、Y 坐标值。最后，在某些情况下，最好给菜单加一个边框，因而还要传递边框开关值。在开发 popup( )函数之前，先说明如下：

```
/* 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 */
```

popup( )函数须完成下述任务：

- ☐ 保存菜单所用的部分屏幕
- ☐ 按要求显示边框
- ☐ 显示菜单
- ☐ 输入用户的响应
- ☐ 将屏幕恢复为原始状态

其中两个目标由上节讨论过的函数 save_video( )和 restore_video( )完成。现在我们看一下如何实现其余三个。

1.4.1 菜单显示

记住，菜单显示的关键，是将指向字符串指针数组的指针传递给 popup( )。为了显示单个字符串，只要将指针象数组一样编成索引。数组的每项均为指向相应菜单项的指针。下列函数 display_menu( )，就使用了这种方法显示每个菜单项。

```
/* 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]);
   }
}
```

可见，须将欲显示字符串数组的指针、起始显示位置的 X、Y 坐标值以及菜单中的项数传递给该函数。

生成存放菜单选项字符的二维数组的最简单方法，是采用以下一般形式产生变量：

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

该变量说明自动地使 C 编译器将字符串放入运行串表中。然后，该变量指向表中第一个字符串的首字符。例如，该说明产生一个称为 fruit 的变量，它指向"Apple"的"A"。

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

1.4.2 边框显示

如果需要边框的话，下面的例行程序可用来给一个已知左上角和右下角坐标的菜单加边框。它使用了一些画线字符，这些字符是 PC 机及其兼容机标准字符集的一部分。如果愿

意的话，也可以用其他字符代替。

```
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, starty); putchar(218);
  goto_xy(startx, endy); putchar(191);
  goto_xy(endx, starty); putchar(192);
  goto_xy(endx, endy); putchar(217);
}
```

1.4.3 输入用户的响应

前已述及，用户可以采用两种方法输入响应。第一个方法是，用向上和向下箭头将醒目部分移至欲选项，然后按下 ENTER 键，选中该项(本书中将菜单的醒目项以反显方式显示——这是一种常用的操作)也可以用空格键移动醒目部分。第二种方法是按下相应的热键进行选项。下面给出的 get_resp( )函数即实现了这些功能。

```
/* 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++;
             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+arrow_choice, y);
    write_video(x+arrow_choice,y, menu[arrow_choice], REV_VID);
   }
 }
```

当 get__resp( )开始执行的时候，第一个菜单选项变得醒目。程序中别的地方将宏 REV__VID 定义为 70H，NORM__VID 为 7。ESCAPE 键用于退出菜单，ESC 的值为 27。然后剩行程序进入等待用户响应的循环。该循环首先用 bioskey( )等待，直至有键按下。然后读取该键。函数 bioskey( )专用于 Turbo C。如果使用与此不同的 C 编译器，可以采用下面的描述：

```
/* emulate part of turbo C bioskey() function */
bioskey(c)
int c;
{
  switch(c) {
    case 0: return get_key();
    case 1: return kbhit();
  }
}
```

使用 bioskey( )而不用 getchar( )的原因，是例行程序必须能够读取由击键产生的满 16 位扫描码。如果按下的是字符键，则返回时该字符在低 8 位，而高 8 位为 0。但是，如果按下的是特殊键，如箭头键，则低 8 位为 0 而高字节为该键的位置码。向上箭头和向下箭头的位置码分别为 72 和 80。getchar( )等函数只返回字符代码，因此没有必要考虑它们而直接读取扫描码。

每当按下一个箭头键，则当前醒目的选项以正常显示再现，而下一项变为醒目。当醒目部分已经位于屏幕低端时，又按下向下箭头，则醒目部分返回到第一项。而当第一项已变为醒目时，按下向上箭头，则醒目部分跳到最底端项。

使用函数 write_video 是借助 get_resp( )在指定的 X、Y 位置处，用给定的属性向视频显示器写一字符串。这里给出的 write_video( )用于当选项为醒目时以反显方式显示菜单项，或者不醒目时返回到正常显示。

```
/* display a string with specified attribute */
void write_video(x, y, p, attrib)
int x, y;
char *p;
int attrib;
{
  union REGS r;
  register int i,j;

  for(i=y; *p; i++) {
        goto_xy(x, 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 = *p++; /* character */
     r.h.bl = attrib; /* attribute */
     int86(0x10, &r, &r);
    }
}
```

函数 is_in( )返回字符串中热键的位置。如果用户接下的是非热键，则返回 0。

```
is_in(s, c)
char *s, c;
{
  register int i;

  for(i=0; *s; i++) if(*s++==c) return i+1;
  return 0;
}
```

1.4.4 Popup( )函数

由于各项已全部产生了,Popup( )函数就可以如下编写:

```
/* display a pop_up menu and return selection
   returns -2 if menu cannot be constructed
   returns -1 if user hits escape key
   otherwise the item number is returned starting
   with 0 as the first (top most) entry
*/
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 */
{
    register int i,len;
    int endx, endy, choice;
    unsigned int *p;

  if((x>24) || (x<0) || (y>79) || (y<0)) {
  printf("range error");
  return -2;
  }

  /* compute the dimensions */
  len = 0;
  for(i=0; i<count; i++)
    if(strlen(menu[i]) > len) len = strlen(menu[i]);
  endy = len +2 + y;
  endx = count + 1 + x;
  if((endx+1>24) || (endy+1>79)) {
    printf("menu won't fit");
    return -2;
  }

  /* allocate enough memory for it */
  p = (unsigned int *) malloc((endx-x+1) * (endy-y+1));
  if(!p) exit(1);  /* install your own error handler here */

  /* save the current screen data */
  save_video(x, endx+1, y, endy+1, p);

  if(border) draw_border(x, y, endx, endy);

  /* display the menu */
  display_menu(menu, x+1, y+1, count,,
```