

刘路放 著



# C语言的窗口式 图形界面设计

— 自带汉字环境的应用软件编程

西安交通大学出版社



# C 语言的窗口式图形界面设计

——自带汉字环境的应用软件编程

刘 路 放 著



西安交通大学出版社

## 内 容 提 要

本书介绍了如何为使用C语言编写的应用程序设计一个“自带汉字环境”的窗口式图形用户界面。书中系统地介绍了VGA显示卡、鼠标、键盘和扩充存储器等设备的编程方法,并在此基础上建立了一个内含式汉字编程环境HANENV,包扩一个头文件和200多个库函数。将HANENV加入Turbo C之后,就可以在编写应用程序时直接引用其库函数。这些库函数的功能包括汉字的输入、存储和显示;时钟、定时器和闪烁光标;全屏幕数据录入和画图;窗口、滚动条、按键式菜单以及代码表等窗口式界面部件等。同时在HANENV系统中还提供了计算器、调色板、文件目录窗口以及屏幕平滑移动等功能,使用HANENV编写的软件经编译后可以直接在西文DOS环境下运行而具有强大的汉字处理能力。因此HANENV系统是使用C语言编写华丽而实用的用户界面应用程序的最佳编程环境。

本书配有两张同名软盘,软盘中除了HANENV系统的所有头文件、库函数及其源程序以外,还包括若干用于字库编辑、造字的工具软件和一个使用HANENV编写的窗口式文本编辑程序。

以上两张软盘(1#,2#)随本书一起发行,每套定价60.00元。除此之外,本书还配有一套扩展字库盘,内容包括32×32点阵、36×40点阵的宋、仿宋、黑、楷体汉字库(系HANENV系统专用的横排显示汉字库,与普通中文操作系统中用于打印的竖排字库不同),每套4张软盘(3#,4#,5#,6#)共80.00元,有兴趣的读者可直接向西安交通大学出版社邮购。(邮费另加15%)

汇款请寄:西安交通大学出版社发行科

邮 编:710049

户 名:西安交通大学出版社

开户银行:西安市工商银行互助路分理处

帐 号:235-14260-90801

(陕)新登字 007 号

### C 语言的窗口式图形界面设计 ——自带汉字环境的应用软件编程

刘 路 放 著

责任编辑 林全

\*

西安交通大学出版社出版发行

(西安市咸宁西路28号 邮政编码710049)

西安电子科技大学印刷厂印装

各地新华书店经销

\*

开本:787×1092 1/16 印张:25.25 字数:775千字

1996年6月第1版 1996年6月第1次印刷

印数:1—5000

ISBN7-5605-0831-6/TP·121 配套程序软盘,书总定价:95.00元

## 前 言

用户界面是现代应用程序中最重要的组成部分。封面是否新颖醒目,版面是否美观大方,操作是否方便易学,这些都是决定应用软件成败的关键之处。编写过应用程序的程序员们都知道,用户界面部分的编程工作量最大,有时可达整个程序工作量的1/2以上。因此,如何编写出性能优良、省工省时的界面程序,就成为应用软件设计者们共同关心的课题之一。

近来窗口式界面随着 WINDOWS 的推广而流行起来。窗口式界面摆脱了字符显示方式的限制,充分利用了图形模式的高分辨率和丰富的色彩,使得应用程序的面貌为之一新。同时窗口式界面大多采用鼠标器作为辅助输入设备,大大方便了用户的操作。但遗憾的是,目前窗口式界面主要由 WINDOWS 平台提供支持,DOS 下的应用程序如果要采用窗口式用户界面,就得自己编写所有的支撑环境,工作量相当庞大。

另外,我国的应用程序设计人员在进行应用程序界面设计时还要解决汉字的输入输出问题。通常的选择是利用某种现有的中文操作系统。这样做的优点是汉字处理独立于应用软件,所有的汉字处理工作均由中文操作系统解决,应用程序人员的负担较轻。但是要为自己的应用程序选择一个合适的中文操作系统也是一个令人头痛的问题:汉字系统的处理速度、显示和打印的效果、可否显示用于封面和标题的大尺寸汉字、汉字输入方法的选择、所用的程序设计语言和汉字操作系统之间的兼容性,以及各种汉字系统的用户群和正在开发的应用软件的用户群是否吻合等因素都应该考虑。因此,近来在应用软件开发厂商中兴起了一种被称为“自带汉字环境”的应用软件开发技术,即把中文操作系统中的汉字处理技术和应用软件有机地结合起来,使得编写出的应用软件本身就具有汉字处理的能力,在西文操作系统下就能直接处理汉字信息的输入输出工作。这种应用软件具有使用方便、体积小和运行速度快的优点,很受用户欢迎。然而要编写出性能优良的“自带汉字环境”的应用软件需要有丰富的设计中文操作系统的经验,因此这种技术目前主要掌握在一些规模较大的软件公司手中,并且多数使用汇编语言作为其编程工具。在目前,对于一般使用高级语言的应用程序设计人员来说,从零开始开发用于汉字处理的各种子程序不但是一件非常麻烦的工作,而且很难找到有关的参考资料。

本书旨在为使用 C 语言的应用程序设计人员和具有一定水平的程序爱好者们介绍有关开发“自带汉字环境”的应用软件的各种技术,包括汉字的输入输出、汉字库的组织以及开发性能良好的汉字窗口式应用软件界面。利用这些技术,我们设计了一个 C 语言的汉字处理软件包——HANENV 系统。HANENV 系统由一个头文件 HANENV.H 和近 200 个函数、若干汉字库文件组成,另外还带有若干工具软件。利用 HANENV 系统开发自带汉字环境的应用软件就像利用 C 语言的图形函数库开发图形软件一样方便。

HANENV 系统为应用软件编程人员提供的支持包括显示各种点阵的汉字或字符、区位、拼音和通用码表式汉字输入法模块(目前支持五笔字型输入法)、方便灵活的正文工作状态和闪烁光标以及多种窗口式用户界面部件,如多功能按键式菜单和图标式菜单、滚动条、代码表、全屏幕数据编辑函数和调色板等,特别是提供了设置大于实际屏幕尺寸的逻辑屏幕的功能,此时实际屏幕变为活动视窗,可以上下左右平滑移动,特别适合于设计大幅面的应用软件界面。HANENV 系统还为应用程序提供了全面的鼠标支持和访问扩充存储器的支持。另外,HANENV 系统还提供了一定的绘图能力。把这些功能结合起来,就可以设计出风格各异、性能优良的用户界面。

HANENV 系统是使用 Turbo C 2.0 开发的,也可以和 Borland C++ 联合使用。在编写 HANENV 系统的各个库函数时我们没有采用 Turbo C 的图形函数库。但是 HANENV 系统和 Turbo C 的 GRAPHICS.LIB 并不矛盾,在开发应用程序时可以同时使用以上两者编写程序。

应该说明的是,由于我们采用了直接写屏技术并充分利用了扩充存储器,所以利用 HANENV 系统编写的应用程序的汉字处理速度超过了许多中文系统的速度。

应用软件的汉字编程环境 HANENV 系统的全套库函数及其源程序、头文件、汉字库以及若干工具软件均作为本书的配套软件发行。

HANENV 系统是一个开放的软件系统,所有库函数的源程序均对读者开放。因此,读者可以根据自己的具体情况对其进行修改和扩充。我们衷心欢迎读者利用 HANENV 系统设计、编写自己的应用程序,并在自己的编程实践中对 HANENV 系统进行检验、修改和补充。我们希望读者能够就自己在使用 HANENV 系统时发现的问题或者对其所做的补充修改,例如新开发的库函数和作者进行交流,以使其更加完善。

在本书的撰写以及 HANENV 系统的设计过程中,得到了诸多老师和同志们的鼓励和帮助,在此表示衷心的感谢。在构思和写作本书的一年多里,我的父亲刘清阳教授对本书的写作提出了许多中肯的意见,李燕梅同志为本书的出版做了许多辅助性的工作,没有他们的共同努力本书就不可能问世得这样快。本书的责任编辑林全同志为本书的出版做了大量的工作,在此一并表示谢意。

刘 路 放

1996 年 6 月于西安交通大学计算机教育中心

## 绪 论

---

计算机是使用拼音文字的西方人发明的。现代计算机的典型形象就是一台显示器和一个由西文打字机演变而来的键盘。就计算机的使用而言,拼音文字的好处是显而易见的:只需使用几十个,至多百十个不同的键便足以直接输入任何文章或指令,显示时也只要考虑为数不多的符号即可。因此,无论是计算机的设计还是使用都比较简单。然而,在将计算机这一现代科技革命的结晶引入中国以后,立刻就产生了怎样将我们古老的建立在表形和表意基础上的方块汉字和计算机相结合的问题,具体说来,就是怎样解决汉字信息在计算机中的存储、交换以及汉字的输出与输入等问题。

首先,要将所有的汉字一一制作成键盘上的按键就成了一个非常困难的问题,而且即使我们造成了这种键盘(体积肯定相当庞大)之后,要想在上面很快地找出自己所需要的汉字也是相当困难的。目前解决这一问题的主要方法是限制计算机所能处理的汉字数目。80年代初制定的国家标准《信息交换用汉字编码字符集·基本集》(GB2312-80)中一共收录了6,763个常用和次常用汉字以及一些其他图形符号。考虑到《康熙字典》收录了42 174个汉字,《辞海》收录了16 534个汉字,而现在一般估计古今汉字累计已达六万多个,因此可以说上述国家标准规定的汉字集规模是太小了。因此,近年来汉字编码集的研究工作还在继续,其中比较重要的成果有国际标准化组织制定的多八位编码体系(ISO-10646),在该体系中汉字和世界各国文字统一编码。上述国家(国际)标准和编码体系提供了汉字的编码,同时也解决了汉字在计算机内部如何存放的问题。就国内目前的大多数汉字系统而言,基本上都是采用了在GB2312-80所规定的两字节汉字编码(称为国标码)上加以某种转换(为了兼容处理西文符号)而产生的变形码作为汉字机内码。

其次,由于西文的字符种类少、图象简单,为了提高显示或打印的速度,通常在设计计算机时,都把所有字符的图象转换成二值矩阵(通常称为点阵)后固化在计算机或打印机的只读存储器中。但是由于汉字的数量多、结构复杂,其点阵占用的存储量很大,将其固化在只读存储器中的方法代价较高,目前主要用于汉字打印机中。在计算机的主机中,必须采用多种灵活的手段处理汉字库的存储和点阵的读取,才能达到较高的输出速度和质量。

第三,即使采用了经过挑选的常用汉字集(如国标汉字集),对于制作汉字专用键盘来说其符号的数目还是太多了。因此,实际上在输入汉字时要采用间接的手段,使用某种输入码(码长 $>1$ )在标准的西文键盘上进行汉字的输入工作。目前已有数百以至上千种汉字输入法问世,其中最常用的也有区位、拼音、五笔字型等十几种。

最后,人们希望计算机在处理汉字信息的同时仍保留其对西文的处理能力,即所谓“中西文兼容”,因为目前有大量的计算机软件是用西文写成的。

目前国内的计算机界基本上都是采用汉字操作系统来解决上述问题。汉字操作系统相当于一个“二传手”,由键盘输入的汉字输入码经过汉字操作系统转换成汉字机内码,再将其送入各个应用程序去处理。处理的结果再由汉字操作系统变换成汉字的点阵显示或打印出来,如图 0-1 所示。

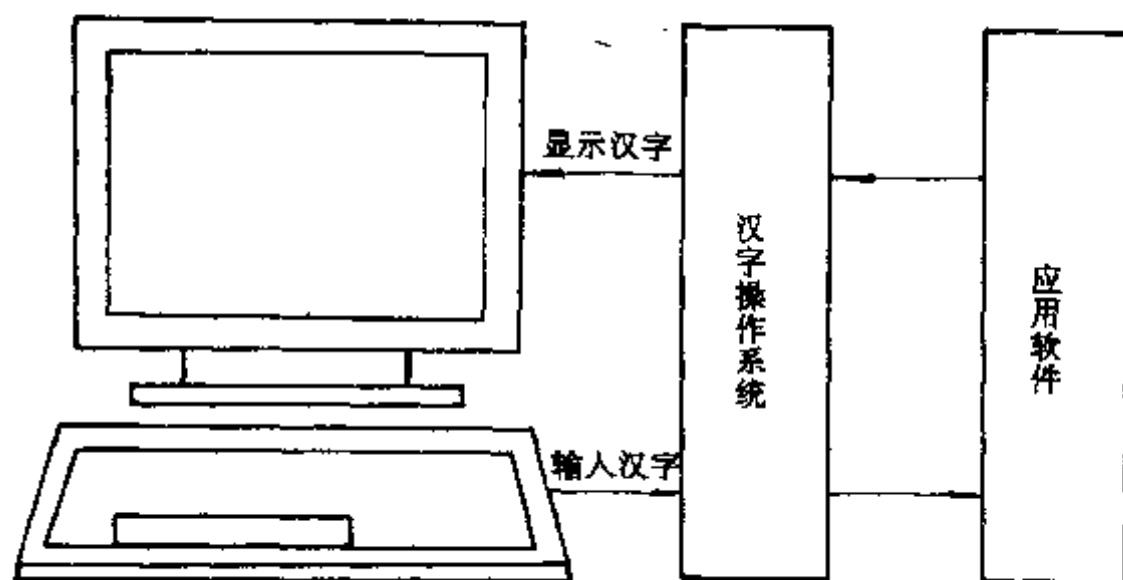


图 0-1 汉字操作系统

这样做的好处是汉字操作系统独立于应用软件。理论上说,一台计算机上只要装上一套汉字操作系统就应该可以供所有需要显示汉字的应用软件使用。但是由于汉字处理工作的复杂性,特别是目前微型计算机处理西文软件的特殊性,能够做到既兼容西文原版软件,汉字显示速度又高的汉字操作系统还很少见。现在市面上的汉字操作系统很多,各有各的优缺点和适用范围,各有各的用户群。所以,在计算机中装有几套不同的汉字操作系统供不同的应用软件使用的情况很普遍,这就给用户带来了许多麻烦。

另一方面,从应用软件开发人员的角度来看,如何为自己的应用软件选择一个合适的汉字操作系统也不容易。因此,近来兴起了一种被称为“自带汉字环境”的应用软件开发技术,即把汉字操作系统中的汉字处理技术和应用软件有机地结合起来,使得编写出的应用软件本身就具有汉字处理的能力,在西文操作系统下就能处理汉字信息。但是要编写出性能优良的“自带汉字环境”的应用软件需要丰富的设计汉字操作系统的经验,因此这种技术目前主要掌握在一些规模较大的软件公司手中,并且多数使用汇编语言编程。本书的目地就是为使用 C 语言的应用程序设计人员和具有一定水平的程序爱好者介绍有关开发“自带汉字环境”的应用软件的各种技术,包括汉字的显示、汉字库的存储、汉字的输入以及怎样开发性能良好的汉字应用软件界面。

## 二

要使自己编写的应用软件能够有效地处理汉字信息,必须解决好三个问题:汉字编码、汉字输出和汉字输入。一般来说,解决汉字编码问题不是应用程序设计人员要考虑的问题,而是国家标准设计委员会的任务。我们只需像大多数汉字系统的设计者一样,选用一种异形国标码即可。在本书介绍的 HANENV 系统中我们选用将国标码的两个字节的最高位都



置为 1 的汉字机内码,这也是绝大多数汉字操作系统所采用的格式,如图 0-2 所示。

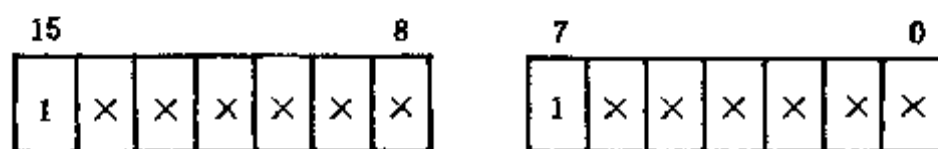


图 0-2 汉字机内码

汉字的输出又可以根据输出设备的不同分为显示、打印以及由绘图机输出等。由于绘图机等的汉字输出工作大多有专用的软件包支持,而现在的大多数打印机又都装有硬汉字库,支持汉字的直接打印输出,所以在 HANENV 系统中我们主要考虑汉字的显示问题(同时为了方便应用程序设计人员拷贝屏幕图形的需要,我们还为 HANENV 系统设计了一个屏幕图形拷贝函数,见 11.5)。为了提高汉字显示速度,我们采用直接写屏的方法显示汉字,因此利用 HANENV 编写的应用程序的汉字显示速度超过了许多汉字操作系统。在设计 HANENV 系统时我们特别考虑了现代应用程序的图形用户界面对汉字显示的要求,不但可以显示标准的  $16 \times 16$  点阵汉字,还可以显示  $24 \times 24$ 、 $32 \times 32$ 、 $40 \times 36$  或者任意其他尺寸的汉字和图形符号,而且在显示时还可以分别指定所要显示的汉字或字符的横向和纵向放大倍数、颜色以及背景等参数。

不同的应用软件对汉字处理的要求也不相同,大体上可以分为以下几种:一种是只需要显示少量在编写程序时已经确定的汉字,例如各种控制软件的菜单、用户界面等;第二种是在编写程序时无法预知需要显示哪些汉字,但在程序运行时并不需要由键盘输入汉字的应用场合,例如文本阅读器;第三种是汉字的输入输出功能都需要,如各种文字编辑软件、数据库和管理信息系统等。对于以上三种情况,使用 HANENV 系统编写程序时可以采用不同的策略:只需要显示少量汉字时可以使用小汉字库,这样程序结构简单、显示速度最快;如果不需要输入汉字就可以不挂接汉字输入模块,以节省内存和缩小程序体积;如果需要制作比较考究的软件封面,还可以使用 HANENV 系统提供的工具软件制作特大点阵(最大可达  $96 \times 96$ )的汉字字模,加工成小字库或者 C 语言的全局数组的源程序供应用程序调用。在 HANENV 系统中汉字库可以根据机器的配置情况选择装载在扩展存储器中或硬盘上,小字库还可以直接编译成目标模块而和程序连为一体,装入内存。

HANENV 系统目前配有区位、全拼、双拼、图形符号等基本汉字输入法和一个通用汉字输入法接口,只要配上不同的输入码表就可以实现各种汉字输入法(目前配有五笔字型词组输入码表)。为了方便编写 WINDOWS 风格的用户界面, HANENV 系统提供了全面的鼠标支持,包括鼠标控制的屏幕平滑移动、虚拟小键盘、按键式菜单、滚动条(scroll box)和一批基本鼠标应用函数。

除此而外, HANENV 系统还包括诸如按键式多功能菜单、时钟、定时器和代码表等较复杂的功能模块的库函数。

在开发管理信息系统等与数据库处理有关的应用软件时, HANENV 系统还可以与图文数据库系统 C-BASE(参看刘路放编著,《C 语言的汉字处理与图文数据库技术》一书,西安交通大学出版社出版)联合使用编程,以取得最佳效果。



### 三

HANENV 系统使用 Turbo C 2.0 开发,也可以和 Borland C++ 联合使用。为了提高汉字输出的速度,我们没有使用 Turbo C 的图形函数库。但是 HANENV 系统和 Turbo C 的 GRAPHICS.LIB 并不矛盾,在开发应用程序时可以同时使用以上两者编写程序(参看 10.1)。

同样,为了提高汉字的显示速度,我们也没有采用调用 BIOS 的显示中断的方法,而是通过直接对 VGA 显示卡的各寄存器和显示存储器(VRAM)操作来显示汉字,即所谓直接写屏。直接写屏的速度很快,但操作起来比较复杂。本书第 1 章对 VGA 显示卡的结构和编程有比较详细的说明。

一般说来,在 C 语言中调用 BIOS 中断或 DOS 功能调用可以有如下几种方法:

1. Turbo C 中有相应的处理函数,例如使用函数 kbhit 调用键盘中断,putch 调用显示器中断等;
2. 使用 Turbo C 的通用中断调用函数 int86、int86x、intdos 和 intdosx;
3. 使用伪变量和中断调用函数 getinterrupt;
4. 使用行间汇编。

第一种方法最省事,但不是所有的功能调用都有相应的处理函数;后三种方法的效果差不多,处理速度一种比一种快,但使用方便程度却是一种比一种差。作为在速度和方便之间作均衡的结果,我们选用了第 3 种方法,即使用伪变量和中断调用函数 getinterrupt 的方法处理 BIOS 中断和 DOS 的功能调用,例如鼠标器编程,调用 XMS 规范等。所谓伪变量就是 80x86 处理器芯片中的寄存器,在 Turbo C 中规定了一套符号与之对应,例如通用寄存器 ax、bx、cx 和 dx 分别使用了伪变量—AX、—BX、—CX 和—DX 表示。因此使用伪变量可以直接对寄存器操作,大大提高了程序的效率。

对于一些对处理速度要求特别高,或者要求直接对某些硬件操作的函数,例如画点函数和处理屏幕平滑移动的有关函数,我们则采用了行间汇编编写程序。在整个 HANENV 系统中,这类函数为数甚少,因为我们发现,只要恰当地设计程序的结构,使用 C 语言一样能够设计出高效率的程序来,而且程序结构清晰,易于修改和交流,可移植性好。

### 四

本书共 15 章,分为四篇。第 1 篇“环境与设备”包括第 1 章至第 5 章,主要介绍 HANENV 的编程基础,内容有 VGA 显示卡的工作原理和编程方法、中断 33H 及鼠标驱动程序设计、键盘缓冲区操作、扩展存储器规范(XMS)的使用以及扩展存储器的直接调用,包括相应的函数设计,以及 HANENV 系统的初始化与装配;第 2 篇“汉字处理”包括第 6 章至第 9 章,介绍各种显示汉字的函数设计、汉字库的组织 and 调用方法、汉字输入模块的设计以及光标、时钟与定时器等函数模块的设计;第 3 篇“用户界面程序设计”包括第 10 章和第 13 章,介绍如何设计 WINDOW 风格的用户界面及一批通用编程部件;第 4 篇“HANENV 系统应用”包括第 14 章至第 15 章,系统地介绍 HANENV 系统的头文件和 HANENV 系统中所有函数的使用方法。

应用软件的汉字编程环境 HANENV 系统的全套库函数及其源程序、头文件、汉字库以

---

及若干工具软件均作为本书的配套软件发行。关于 HANENV 系统的装配可以参看 5.2 节。由于 HANENV 系统的大部分函数的源程序已经作为本书各章的例子出现, 所以如果读者没有购买到本书的配套软盘, 也可以根据本书各章所介绍的原理自行编写其余函数。

HANENV 系统是一个开放的软件系统, 所有库函数的源程序均对读者开放。因此读者可以根据自己的具体情况对其进行修改和扩充。

# 目 录

## 绪论

## 第 1 篇 环境与设备

第1章 EGA/VGA 显示卡的基本工作原理及编程	(2)
1.1 EGA/VGA 卡的显示模式	(2)
1.2 VGA 卡的结构	(4)
1.3 BIOS 的显示器中断	(6)
1.4 VGA 卡的寄存器	(12)
1.5 EGA/VGA 卡使用小结	(26)
第2章 鼠标应用程序设计	(27)
2.1 鼠标的初始化	(28)
2.2 自制鼠标光标驱动	(30)
2.3 测试和设置鼠标状态	(40)
2.4 HANENV 系统中关于鼠标应用的其他设置	(47)
第3章 键盘操作	(48)
3.1 Turbo C 的键盘操作库函数	(48)
3.2 DOS 系统功能调用	(49)
3.3 使用 BIOS 的键盘中断编程	(50)
3.4 键盘缓冲区	(58)
第4章 扩充存储器编程	(62)
4.1 扩充存储器与扩充存储器调用规范 XMS	(62)
4.2 直接访问扩充存储器	(64)
4.3 利用扩充存储器管理规范 XMS 访问扩充存储器	(70)
第5章 HANENV 系统的初始化与装配	(80)
5.1 HANENV 系统的初始化	(80)
5.2 HANENV 系统的装配	(89)
5.3 HANENV 系统的工具软件	(92)

## 第 2 篇 汉字处理

第6章 汉字显示	(103)
6.1 显示一个像素点	(105)
6.2 显示一个字模点阵	(109)
6.3 以更快的速度显示汉字	(115)
6.4 汉字点阵的放大	(116)



6.5 HANENV 系统中的汉字(字符)显示函数族 .....	(121)
<b>第7章 汉字库的组织</b> .....	(123)
7.1 HANENV 的汉字库结构 .....	(123)
7.2 从汉字库中取字模点阵 .....	(128)
7.3 HANENV 系统的字库及其生成 .....	(130)
<b>第8章 汉字输入模块的设计</b> .....	(135)
8.1 汉字输入函数 gethan .....	(135)
8.2 输入法模块及其装入 .....	(141)
8.3 拼音输入法模块的设计 .....	(152)
8.4 通用输入法模块的设计 .....	(159)
<b>第9章 光标、时钟和定时器</b> .....	(174)
9.1 BIOS 的时钟中断 1CH .....	(176)
9.2 光标与汉字的文本工作方式设计 .....	(181)
9.3 正文工作方式 .....	(181)
9.4 时钟与定时器 .....	(188)
9.5 鼠标、光标和时钟函数应用小结 .....	(194)

### 第3篇 用户界面程序设计

<b>第10章 屏幕作图</b> .....	(197)
10.1 HANENV 系统和 Turbo C 的图形库联合编程 .....	(197)
10.2 横竖线、框和矩形块 .....	(198)
10.3 构造对话框和按键 .....	(206)
10.4 直线和曲线 .....	(210)
<b>第11章 图形用户界面设计</b> .....	(218)
11.1 提示和对话框 .....	(218)
11.2 滚动条 .....	(225)
11.3 代码表 .....	(235)
11.4 调色板 .....	(251)
<b>第12章 全屏幕数据编辑</b> .....	(260)
12.1 基本数据编辑函数 .....	(260)
12.2 各种类型数据字段的编辑函数 .....	(266)
12.3 设计一个全屏幕数据录入、编辑界面程序 .....	(269)
<b>第13章 菜单程序设计</b> .....	(278)
13.1 多功能按键式菜单及其应用 .....	(278)
13.2 图标式菜单 .....	(301)

### 第4篇 HANENV 系统应用

<b>第14章 HANENV 系统的头文件</b> .....	(311)
<b>第15章 HANENV 系统的库函数</b> .....	(336)
<b>参考文献</b>	

# 第 1 篇

## 环境与设备

# EGA/VGA 显示卡的基本工作原理及编程

微型计算机的显示系统由监视器和显示卡组成,在台式机中监视器是一个独立的外部设备,而在笔记本、笔式机中则与主机装配在一起。显示卡是插在主机中主板上的一块插卡(在某些计算机中显示卡的有关线路被直接设计在主板上,因此没有独立的显示卡),计算机对显示屏幕的所有操作都是通过显示卡进行的。显示卡和监视器是配套使用的,一种显示卡只能配用一种监视器。

显示卡的种类很多,例如 MDA 卡、CGA 卡、EGA 卡、MCGA 卡以及 VGA 卡等等。但目前流行得最广的是 VGA 卡或 VGA 的扩充兼容卡(例如 SVGA 卡、TVGA\*卡等),因为 VGA 卡不仅显示功能强大,而且还可以兼容 MDA 卡、CGA 卡、EGA 卡的所有显示模式。HANENV 系统的汉字显示就是针对 VGA 显示卡设计的。应该说明的是,由于 VGA 卡高度兼容 EGA 卡,所以在大多数资料中是将这两种显示卡放在一起介绍的,通称为 EGA/VGA 显示卡。本书介绍的 HANENV 系统的绝大部分内容只需略加修改(主要是在屏幕尺寸方面)也可以应用于 EGA 显示卡。

## 1.1 EGA/VGA 卡的显示模式

EGA/VGA 显示卡的显示模式可以分为字符显示模式和图形显示模式两大类,共 15 种,见表 1-1。

字符显示模式是为了快速显示西文字符而设计的,不宜用来直接显示汉字。通常显示汉字需要在图形模式下将汉字的字模作为图形块逐点画出。从表 1-1 中可以看出,在 VGA 的各种图形显示模式中模式 12H 最适合于显示汉字。模式 12H 的分辨率最高,可达  $640 \times 480$  点,如果使用  $16 \times 16$  点阵的汉字库,行间距取两点,显示 25 行汉字一共需要  $(16+2) \times 25 = 450$  线,还剩下 30 线可以用作汉字输入的提示行。每行可以显示  $640 \div 16 = 40$  个汉字,或者 80 个西文(半角)字符,和常用的字符显示模式(如模式 03H)正好相同。模式 12H 允许在 262 114(256K)种颜色中选择 16 种颜色同屏显示,色彩比较丰富。因此和大多数汉字操作系统一样,在设计 HANENV 系统时我们也选用 VGA 的模式 12H 作为工作模式。但应注意 12H 为 VGA 显示卡所独有的显示模式。因此如果需要在 EGA 显示卡的计算机上



实现汉字显示,可以选用图形模式 10H。该模式和模式 12H 最接近,只是分辨率略低。在模式 10H 下如果仍然选用  $16 \times 16$  点阵的汉字字模和两个象素的行间距,那就只能显示 19 行汉字了,除去一行提示行,只能显示 18 行正文。

表 1-1 VGA 卡的显示模式

模式	类型	分辨率	颜色数	可用显示卡				显示地址
00H	字符	$40 \times 25$	16	CGA	EGA	MCGA	VGA	B800
01H	字符	$40 \times 25$	16	CGA	EGA	MCGA	VGA	B800
02H	字符	$80 \times 25$	16	CGA	EGA	MCGA	VGA	B800
03H	字符	$80 \times 25$	16	CGA	EGA	MCGA	VGA	B800
04H	图形	$320 \times 200$	4	CGA	EGA	MCGA	VGA	B800
05H	图形	$320 \times 200$	4	CGA	EGA	MCGA	VGA	B800
06H	图形	$640 \times 200$	2	CGA	EGA	MCGA	VGA	B800
07H	字符	$80 \times 25$	2	EGA	VGA			B800
0DH	图形	$320 \times 200$	16	EGA	VGA			A000
0EH	图形	$640 \times 200$	16	EGA	VGA			A000
0FH	图形	$640 \times 350$	2	EGA	VGA			A000
10H	图形	$640 \times 350$	16	EGA	VGA			A000
11H	图形	$640 \times 480$	2	MCGA	VGA			A000
12H	图形	$640 \times 480$	16	VGA				A000
13H	图形	$320 \times 200$	256	MCGA	VGA			A000

显示模式的设置可以通过调用 BIOS 的显示中断的 00H 号功能进行,其调用方法如下:

入口寄存器设置:

AH = 00H

AL = 模式号

返回寄存器:无

程序 1-1 设置 VGA 显示模式 12H 和 03H。

在 HANENV 系统的初始化函数 init\_hanenv() 中有如下程序段:

```

/* -----
   设置 VGA 模式 12H(640 行×480 列×16 色)
   ----- */
_AH = 0x00;
_AL = 0x12;
geninterrupt(0x10);

```

用于设置 VGA 的图形显示模式 12H。

在函数 close\_hanenv() 中有如下程序段:

```

/* -----

```

设置 VGA 模式 03H(25 行×80 列的字符显示模式)

```
----- */
_AH = 0x00;
_AL = 0x03;
geninterrupt(0x10);
```

用于恢复 VGA 的字符显示模式 03H。

## 1.2 VGA 卡的结构

下面以模式 12H 为例介绍 VGA 卡的基本结构。

显示存储器(VRAM)用来存放要在屏幕上显示的数据,由 256K 字节的动态随机存储器构成,分为 4 个颜色平面。这 4 个颜色平面各有 64K 大小,在内存中占用相同的地址空间,起始地址为 A000:0000。在显示时屏幕上的一个象素由 4 位表示,每个颜色平面 1 位,因此每个象素可以是 16 种颜色中的一种,显示存储器中的每个字节存放相邻的 8 个水平象素。颜色平面的映射原理如图 1-1 所示。

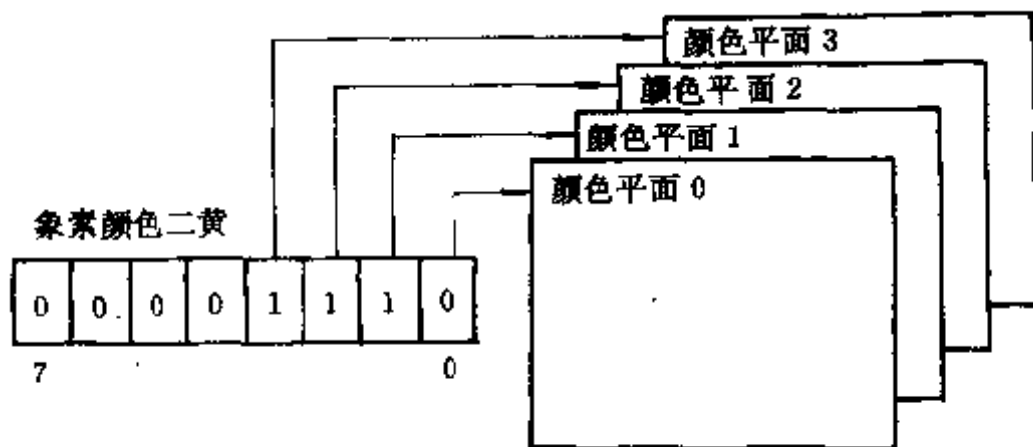


图 1-1 颜色平面的映射原理

具体说来,显示屏幕上的象素位置(X,Y)到显示存储器的位地址之间的转换可以按下列公式进行计算:

$$\text{字节地址} = Y \times 80 + X/8$$

$$\text{位地址} = 7 - (X \bmod 8)$$

存放在位平面中的象素颜色数据还要经过数模转换寄存器(DAC)的处理才能变为屏幕上的实际色彩,其原理如图 1-2 所示。

从图 1-2 中可以看出,VGA 显示卡通过显示存储器 4 个颜色平面的 4 位数据来选择调色板寄存器,调色板寄存器的低 4 位与由颜色选择寄存器的 C4、C5、C6 和 C7 位组成的高 4 位一起形成 DAC 颜色寄存器的八位地址,再由 DAC 颜色寄存器的值决定最终在屏幕上显示的象素的颜色。DAC 颜色寄存器的字长为 18 位,分别用其中的 6 位表示红、绿、蓝三种原色的分量。在设置模式 12H 时,16 个调色板寄存器和 256 个颜色寄存器分别被置为系统省缺值,见表 1-2。

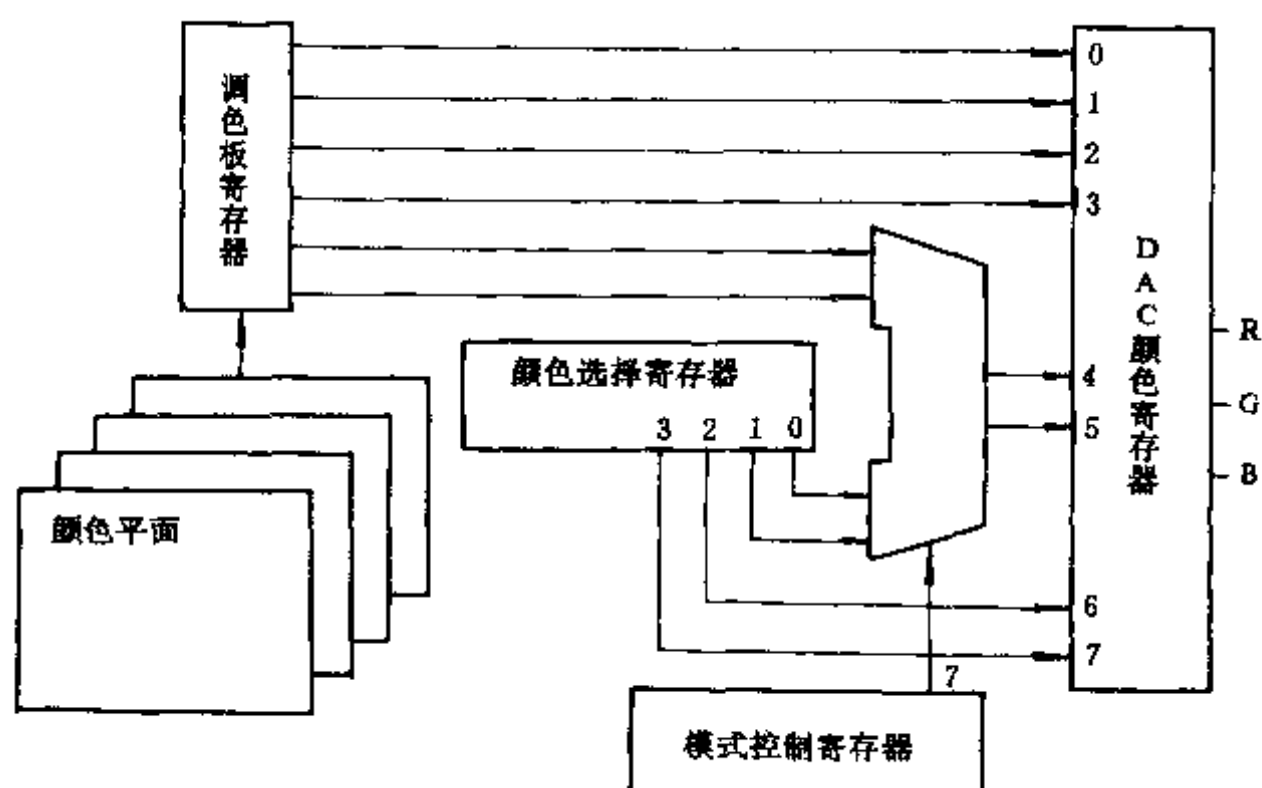


图 1-2 VGA 卡的颜色控制原理

表 1-2 模式 12H 的颜色省缺值

颜色号	颜色名称	调色板寄存器的值	红	绿	蓝
0	BLACK(黑)	0	0	0	0
1	BLUE(蓝)	1	0	0	42
2	GREEN(绿)	2	0	42	0
3	CYAN(青)	3	0	42	42
4	RED(红)	4	42	0	0
5	MAGENTA(洋红)	5	42	0	42
6	BROWN(棕)	6	42	42	0
7	LIGHTGRAY(浅灰)	20	42	42	42
8	DARKGRAY(深灰)	56	0	0	21
9	LIGHTBLUE(亮蓝)	57	0	0	63
10	LIGHTGREEN(亮绿)	58	0	42	21
11	LIGHTCYAN(亮青)	59	0	42	63
12	LIGHTRED(亮红)	60	42	0	21
13	LIGHTMAGENTA(亮洋红)	61	42	0	63
14	YELLOW(黄)	62	42	42	21
15	WHITE(白)	63	42	42	63

除了显示存储器以外,VGA 卡中还有 CRT 控制器、串并转换器、属性控制器、时序发生器 etc 等部件,其结构见图 1-3。



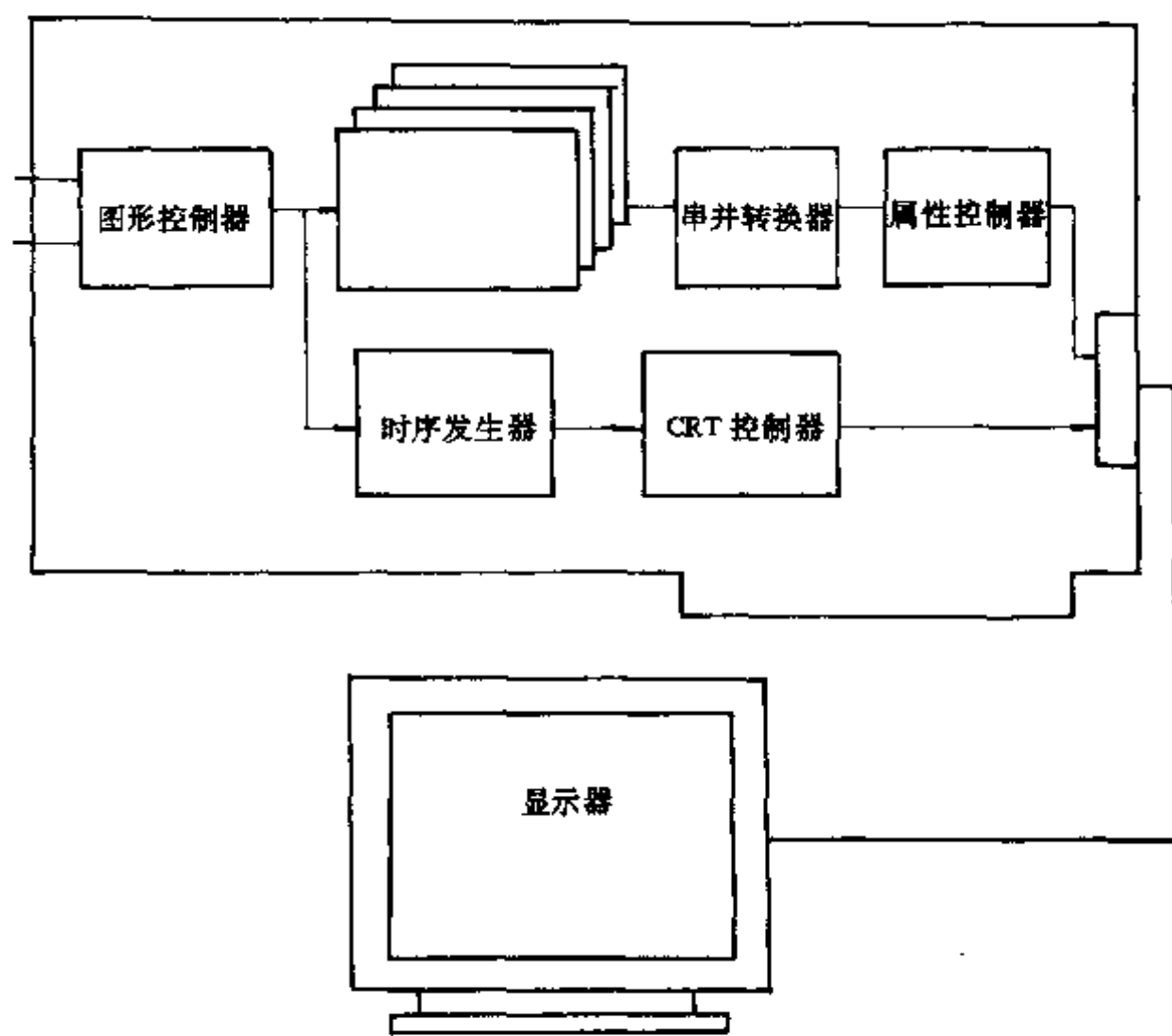


图 1-3 VGA 显示卡的结构

### 1.3 BIOS 的显示器中断

BIOS 的显示中断 10H 一般读者都很熟悉,利用该中断的各功能调用可以很方便地控制 VGA 显示卡的各种功能,但其缺点是速度稍慢。因此,在设计 HANENV 系统中的汉字显示、清屏、画点、线、块等需频繁调用的函数时我们不采用调用 BIOS 中断 10H 的方法,而是直接对 VGA 显示卡上的各寄存器和显示存储器 VRAM 进行编程,以提高显示速度。但对有些无需反复调用的函数来说,利用 BIOS 中断进行设计就比较合适。表 1-3 列出了 BIOS 的显示器中断(10H)的常用功能调用。

表 1-3 BIOS 的显示器中断的常用功能调用

功能号	功 能	备 注
00H	设置显示模式(0—13H)	见程序 1-1
01H	设置光标大小	
02H	设置光标位置	
03H	查询光标大小和位置	
04H	查询光笔位置	
05H	选择活动页	

功能号	功 能	备 注
06H	向上滚动正文窗口或使窗口空白	
07H	向下滚动正文窗口或使窗口空白	
08H	读光标所在处的字符及其属性	
09H	在光标所在处写字符及其属性	
0AH	在当前光标处写字符	
0BH	设置 CGA 调色板(方式 4—6)	
0CH	在屏幕上指定位置画点	速度太慢, HANENV 不采用
0DH	读屏幕上指定位置象素颜色	
0EH	写字符并移动光标	
0FH	查询当前显示方式	
10H	设置 EGA/VGA 调色板寄存器	包括若干子功能, 见表 1-4
11H	装入字符生成器	包括若干子功能
12H	查询 EGA/VGA 状态	包括若干子功能, 见程序 1-6
13H	在屏幕上指定位置写正文串	
1AH	读/写显示设备的配置信息	包括若干子功能
1BH	返回显示器的状态信息	
1CH	保存/恢复显示卡状态	包括若干子功能

BIOS 的显示器中断有几个功能中还包括若干子功能, 如功能 10H、11H、12H、1AH 和 1CH。其中 10H 在 HANENV 中应用较多。表 1-4 中列出了功能 10H 的子功能。

表 1-4 BIOS 的显示器中断的 10H 功能的子功能

子功能号	功 能	备 注
00H	设置单个调色板寄存器	参看程序 1-3
01H	设置屏幕边界颜色寄存器	参看程序 1-5
02H	设置所有调色板寄存器	
03H	闪烁/加亮属性控制	
07H	读单个调色板寄存器	参看程序 1-2 和 1-4
08H	读屏幕边界颜色寄存器	
09H	读所有调色板寄存器	
10H	设置单个数模转换寄存器(DAC)	参看程序 1-2 和 1-5
12H	设置一组数模转换寄存器(DAC)	
13H	选择颜色子集	
15H	读单个数模转换寄存器(DAC)	参看程序 1-4
17H	读一组数模转换寄存器(DAC)	
1AH	读颜色子集状态	
1BH	转换数模转换寄存器(DAC)的内容为灰度	

下面我们通过几个例子说明 BIOS 的显示器中断在 HANENV 系统中的应用。

#### 程序 1-2 改变屏幕显示颜色。

改变屏幕显示颜色的工作包括根据颜色号选择调色板寄存器和修改调色板寄存器两个

步骤。这两个步骤都要使用 BIOS 显示中断的 10H 号功能调用的子功能。根据颜色号选择调色板寄存器可以使用 INT10H 的 10H 号功能调用的 07H 号子功能。该子功能的调用方法如下：

入口寄存器设置：

AH = 10H  
AL = 07H  
BL = 颜色号(0—15)

返回寄存器：

BH = 调色板寄存器号(0—255)

而修改调色板寄存器可以使用 INT10H 的 10H 号功能调用的 10H 号子功能。该子功能的调用方法如下：

入口寄存器设置：

AH = 10H  
AL = 10H  
BX = 调色板寄存器号  
DH = 颜色中的红分量(0—63)  
CH = 颜色中的绿分量(0—63)  
CL = 颜色中的蓝分量(0—63)

返回寄存器：无

如果要设置的是屏幕边缘颜色，那么还需要使用 BIOS 显示中断的 10H 号功能调用的 01H 号子功能。该子功能将屏幕边缘颜色寄存器指向某个调色板寄存器(在省缺设置下，该寄存器指向屏幕背景色对应的调色板寄存器，即黑色)。其调用方法如下：

入口寄存器设置：

AH = 10H  
AL = 01H  
BH = 调色板寄存器号(0—255)

返回寄存器：无

```
/* -----
   函数 set_color : 改变颜色。
   ----- */

#include <dos.h>

void _Cdecl set_color(color_no, red, green, blue)
int color_no;      /* 颜色号 : 0—16, 可以使用颜色名称宏 */
int red;           /* 红分量(0—63) */
int green;         /* 绿分量(0—63) */
int blue;          /* 蓝分量(0—63) */
{
    /* —— 根据颜色号选择调色板寄存器 —— */
}
```



```

    if(color_no < 16)
    {
        _AX = 0x1007;
        _BL = color_no;
        geninterrupt(0x10);
        color_no = _BH;
    }
    else
        color_no = 0xff;

    /* -- 修改调色板寄存器 ----- */
    _DH = red;
    _CH = green;
    _CL = blue;
    _BX = color_no;
    _AX = 0x1010;
    geninterrupt(0x10);

    /* -- 处理屏幕边缘颜色 ----- */
    if(color_no == 0xff)
    {
        _BH = 0xff;
        _AX = 0x1001;
        geninterrupt(0x10);
    }
}

```

函数 `set_color` 可以用来将省缺的 16 种屏幕显示颜色和屏幕边缘颜色修改为程序设计人员指定的其他颜色,其选择范围可达  $262\,144(256K)$  种,为制作漂亮新颖的屏幕界面提供了重要的手段。不过,要调配出令人满意的色彩来也不是一件很容易的事。在 HANENV 系统中还有一个调色板函数 `palette`(参看 12.5),可以用于实时修改屏幕色彩。而在 HANENV 系统提供的实用工具 HANTOOL 中有一个调色板图标,只要使用鼠标左键点选该图标即可调出一个用鼠标控制的调色板,可以很方便地调配出各种颜色并给出每种颜色的构成供 `set_color` 函数使用。

使用上面的程序,只要取参数 `color_no` 的值为 `EDGE_COLOR(16)`,就可以设置屏幕边缘的颜色。我们首先将一不常用的调色板寄存器(在上面的程序中是第 255 号调色板寄存器)的值设置为所需要的屏幕边缘颜色,然后再将屏幕边缘颜色寄存器指向该调色板寄存器。这样做的原因是在设置 VGA 的模式 12H 时屏幕边缘颜色寄存器的省缺值被设置为与调色板寄存器组的 0 号寄存器的值相同,而调色板的 0 号寄存器的值同时也是屏幕背景颜色(省缺值为黑色),如果直接修改便会影响到屏幕显示效果。而使用上述方法设置屏幕的边缘颜色,还可以达到同屏显示 17 种颜色的效果(包括屏幕边缘颜色)。

### 程序 1-3 改变省缺的 16 种颜色设置。

改变省缺的 16 种颜色设置可以使用 BIOS 显示中断的 10H 号功能调用的 00H 号子功能。该子功能的调用方法如下：

入口寄存器设置：

AH = 10H  
AL = 00H  
BL = 颜色号(0—15)  
BH = 调色板寄存器号(0—255)

返回寄存器：无

```
/* -----
   函数 change_color : 改变省缺的 16 种颜色设置
   ----- */
#include <dos.h>
void _Cdecl change_color(color,new_color)
int color;          /* 颜色号(0—15)          */
int new_color;      /* 调色板寄存器号(0—255)      */
{
    _AX = 0x1000;
    _BL = color;
    _BH = new_color;
    geninterrupt(0x10);
}
```

使用函数 change\_color 可以改变屏幕的背景色。在 VGA 图形模式 12H 的初始设置中，屏幕背景色被设置为调色板的第 0 号寄存器(黑色)。如果要将其变为红色，可以调用函数 change\_color：

```
change_color(BLACK,RED);
```

### 程序 1-4 取当前屏幕各颜色的构成。

取当前屏幕各颜色的构成可以使用 BIOS 显示中断的 10H 号功能调用的 15H 号子功能。该子功能的调用方法如下：

入口寄存器设置：

AH = 10H  
AL = 15H  
BX = 调色板寄存器号(0—255)

返回寄存器：

\_DH = 颜色中的红分量(0—63)  
\_CH = 颜色中的绿分量(0—63)

`_CL` = 颜色中的蓝分量(0—63)

在使用本功能调用取某颜色的构成时应该首先使用 10H 号功能调用的 07H 号子功能找到某颜色对应的调色板寄存器号。

```
/* -----
   函数 get_color : 分析颜色的构成
   ----- */
#include <dos.h>

void _Cdecl get_color(color_no, red, green, blue)
int color_no;      /* 颜色号 : 0—16, 可以使用颜色名称宏 */
int *red;          /* 红分量(0—63) */
int *green;        /* 绿分量(0—63) */
int *blue;         /* 蓝分量(0—63) */
{
    /* -- 根据颜色号选择调色板寄存器 ----- */
    if(color_no < 16)
    {
        _AX = 0x1007;
        _BL = color_no;
        geninterrupt(0x10);
        color_no = _BH;
    }
    else
        color_no = 0xff;

    /* -- 取指定颜色的红、绿、蓝分量 ----- */
    _AX = 0x1015;
    _BX = color_no;
    geninterrupt(0x10);
    *red = _DH;
    *green = _CH;
    *blue = _CL;
}
```

和使用函数 `set_color` 时相同,只要取参数 `color_no` 的值为 `EDGE_COLOR`,就可以查询屏幕边缘颜色。不过,这得在使用了 `set_color` 函数设置了屏幕边缘颜色之后才行,否则屏幕边缘颜色和第 0 号颜色(黑色)相同。

#### 程序 1-5 关闭和打开屏幕显示。

关闭和打开屏幕显示这两个函数使用了 BIOS 显示器中断的 12H 号功能调用的 36H

号子功能。其调用方法如下:

入口寄存器设置:

AH = 12H

AL = 开关屏幕显示(0 = 开,1=关)

BL = 36H

返回寄存器:无

```

/* -----
   函数 close_display : 关闭屏幕显示
----- */
#include <dos.h>

void _Cdecl close_display(void)
{
    _BL = 0x36;
    _AH = 0x12;
    _AL = 0x01;
    geninterrupt(0x10);
}

/* -----
   函数 open_display : 打开屏幕显示
----- */
#include <dos.h>

void _Cdecl open_display(void)
{
    _BL = 0x36;
    _AH = 0x12;
    _AL = 0x00;
    geninterrupt(0x10);
}

```

使用这两个函数可以实现整屏弹出的效果。如果我们需要在屏幕上画出比较复杂的图形版面或写出尺寸较大的汉字而又不想让用户看到画图过程,就可以在画图前关闭屏幕显示,待画图过程结束后再恢复显示,即可达到整屏图像立即更新的效果。

## 1.4 VGA 卡的寄存器

EGA/VGA 显示卡共包括了 5 组共约 60 个寄存器,其使用相当复杂。一般来说,EGA 的寄存器大多只能写,而 VGA 的寄存器大多数既能写也能读。EGA/VGA 的寄存器虽然



多,但其中只有部分与汉字显示直接有关,下面我们分别介绍之。

#### 1.4.1 时序发生器

时序发生器控制所有 VGA 功能的总定时信号,并完成某些存储地址译码工作。时序发生器包括 5 个寄存器,复用两个 I/O 地址(3C4 和 3C5)。索引寄存器地址为 3C4,用来选择当前寄存器;数据通过 I/O 地址 3C5 输出到选中的寄存器。时序发生器中的寄存器可参看表 1-5。在时序发生器的 5 个寄存器中,最常用的是颜色平面允许写寄存器。除此而外,在 HANENV 中还用到了时钟方式寄存器。

表 1-5 时序发生器的寄存器

索引号	寄存器名称
00	复位寄存器
01	时钟方式寄存器
02	颜色平面允许写寄存器
03	字符发生器选择寄存器
04	存储方式寄存器

**时钟方式寄存器(索引号 01H)** 配置时序发生器的定时线路。该寄存器的内容在 BIOS 初始化时确定,一般情况下应用程序不必修改该寄存器的内容。该寄存器的第 4 位确定 EGA/VGA 用字节方式编址还是用字方式编址。在实现屏幕的平滑滚动时我们要用到这个参数。

**颜色平面允许写寄存器(索引号 02H)** 这是一个对图形绘制非常重要的寄存器,控制是否允许处理器写颜色平面。通过有选择地允许写某些颜色平面,就可以将特定的图形显示在屏幕上。该寄存器的位定义为

省缺设置:0x0f(即所有平面均允许写)

第 7—4 位:保留

第 3 位:允许写平面 3(1=允许)

第 2 位:允许写平面 2(1=允许)

第 1 位:允许写平面 1(1=允许)

第 0 位:允许写平面 0(1=允许)

#### 程序 1-6 将保存在内存中的屏幕映象块恢复到屏幕上。

该函数的关键是将内存中的数据直接写入显示存储器 VRAM。在显示存储器中 4 个颜色平面复用同一段地址空间,而在常规内存中我们将同一颜色平面的数据放在一起,4 个颜色平面的数据分别占用 4 块连续的常规内存区域,通过使用 EGA/VGA 的时序发生器中的颜色平面允许写寄存器来控制将数据分别写入 4 个颜色平面。

```
/* -----
   函数 putblock : 恢复屏幕映象块。
   ----- */
```

```

#include <hanenv.h>
#include <alloc.h>

void putblock(col,line,width,high,block)
int col;          /* 屏幕图象块左上角列坐标(以字节为单位) */
int line;         /* 屏幕图象块左上角行坐标(以像素为单位) */
int width;        /* 屏幕图象块宽度(以字节为单位) */
int high;         /* 屏幕图象块高度(以像素为单位) */
char ** block;    /* 屏幕图象块存储缓冲区 */
{
    register unsigned i,j,k;
    char * p;

    if(block)
    {
        /* -- 设置显示寄存器操作标志 ----- */
        _VideoBusy = YES;
        for(i=1,j=0;i<9;i*=2,j++)
        {
            p = block[j];
            outportb(0x3c4,2); /* 选择颜色平面允许写寄存器 */
            outportb(0x3c5,i); /* 允许写第i个平面 */
            for(k=0;k<high;k++)
            {
                movedata(FP_SEG(p),FP_OFF(p),0xa000,
                        (line+k+_ScreenTop)*_ScreenWidth+col,width);
                p += width;
            }
            free(block[j]);
        }
        outportb(0x3c4,2); /* 恢复允许写所有位平面 */
        outportb(0x3c5,0x0f);
        _VideoBusy = NO;
        free(block);
    }
}

```

该函数和 getblock(见程序 1-8)配合使用以完成屏幕图形块的读写。getblock 将屏幕上的一块指定矩形区域的图象数据存放在常规内存中由指针 block 指向的存储块数组中。putblock 用来将存储在 block 中的图形块重新写回屏幕。该函数中值得注意的是参数 block 的定义。block 实际上是一个动态二维数组,由 4 个一维数组构成,每个一维数组分别对应一个颜色平面。之所以这样设计的理由是 C 语言规定每个数据块的大小不能超过 64K。由于一

个位平面的大小不超过 64K, 因此 getblock 和 putblock 可以拷贝屏幕上任意大小的图象块。该函数关于坐标和尺寸的参数的单位也是容易弄错的地方。参数 line 和 high 的单位为像素, 而参数 col 和 width 的单位为字节(8 个像素), 这当然是由于 VGA 显示模式 12H 的显示存储器的组织结构而引起的。HANENV 中有许多函数的位置、尺寸参数都是这样定义的。但是 HANENV 中还有一些函数的列坐标参数和宽度参数的单位也是像素, 和行坐标和高的单位相同。因此在使用一个函数时首先应该弄清其参数的单位。这可以通过查阅第 15 章“HANENV 系统的库函数”来解决。实际上还有另一种更简单的方法, 即通过函数的参数变量名来判断。如果一个函数的坐标参数被写为(col, line), 其列坐标和宽度的单位就是字节, 而如果写为(x, y), 单位就是像素。

#### 1.4.2 图形控制器

图形控制器可以对要写入显示存储器的数据进行逻辑运算(例如与、或、异或等)。图形控制器包括 9 个内部寄存器, 复用两个 I/O 地址(3CE 和 3CF)。索引寄存器使用 I/O 地址 3CE, 用来选择图形控制器的各个寄存器, 通过地址 3CF 对选中的寄存器进行读写。图形控制器的各寄存器见表 1-6。在执行任何绘图操作之前, 还应通过设置时序发生器的颜色平面允许写寄存器选择颜色平面。

表 1-6 图形控制器的寄存器

索引号	寄存器名称
00	置位/复位寄存器
01	允许置位/复位寄存器
02	颜色比较寄存器
03	数据移位和函数选择寄存器
04	读平面选择寄存器
05	方式寄存器
06	混合寄存器
07	颜色忽略寄存器
08	位屏蔽寄存器

**置位/复位寄存器(索引号 00H)** 用来存放写向显示存储器的填充数据。在 HANENV 系统的许多函数中, 我们使用该寄存器决定写入屏幕的字符、汉字或图形的颜色。该寄存器应该和允许置位/复位寄存器联合使用。该寄存器的位定义为

省缺设置: 所有方式下均为 0。

第 7—4 位: 保留

第 3 位: 平面 3 的填充数据

第 2 位: 平面 2 的填充数据

第 1 位: 平面 1 的填充数据

第 0 位: 平面 0 的填充数据

**允许置位/复位寄存器(索引号 01H)** 决定哪些存储平面接收来自置位/复位寄存器的填充数据。任何被禁止置位/复位的平面将用通常的处理器输出数据填写。该寄存器的位

定义为

省缺设置:所有方式下均为0。

第7—4位:保留

第3位:允许对平面3置位/复位(1=允许)

第2位:允许对平面2置位/复位

第1位:允许对平面1置位/复位

第0位:允许对平面0置位/复位

**颜色比较寄存器(索引号 02H)** 可以用来在显示存储器中快速查找指定颜色的象素。颜色比较寄存器可以在单个读存储周期内将所有4个显示平面的数据与参考颜色比较,并报告每个象素位置的颜色是否匹配。颜色比较寄存器在图形区域填充算法中用于边界查找特别有效。该寄存器必需和方式寄存器联合使用。该寄存器的位定义为

省缺设置:所有方式下均为0。

第7—4位:保留

第3位:平面3的颜色比较值

第2位:平面2的颜色比较值

第1位:平面1的颜色比较值

第0位:平面0的颜色比较值

**数据移位和函数选择寄存器(索引号 03H)** 控制两个独立的功能:写数据移位和对写数据的逻辑操作。在写周期中数据可以被移位0~7位。必须选择写方式0以允许移位。函数选择位为在显示存储器上的读写操作提供了基本的硬件支持。该寄存器的位定义为

省缺设置:所有方式下均为0。

第7—5位:保留

第4—3位:函数选择位: =0——不修改写数据

=1——写数据 AND(与)锁存器

=2——写数据 OR(或)锁存器

=3——写数据 XOR(异或)锁存器

第2—0位:数据移位位数

如果移位和逻辑操作都允许,则移位在逻辑操作之前完成。每个颜色平面都有一个对应的一字节宽的锁存器,因此在一次显示存储器读周期中32位数据可同时装入锁存器内。

**读平面选择寄存器(索引号 04H)** 决定处理器可以读哪个颜色平面(除颜色比较方式以外)。该寄存器的位定义为

省缺设置:0

第7—2位:保留

第1—0位:定义可读的颜色平面(0~3)

**方式寄存器(索引号 05H)** 中的大多数位不应由应用程序修改。但有两位是有用的,即写方式位和颜色比较方式允许位。写方式位控制处理器数据如何写进显示存储器(参看颜色比较寄存器)。

如果应用程序要修改方式寄存器,则必须注意保护第7—4位的状态,以免适配器的操作方式被破坏。该寄存器的位定义为

- 第 7 位 : 保留(0)
- 第 6 位 : 256 种颜色方式(仅对 VGA)
- 第 5 位 : 移位寄存器方式
- 第 4 位 : 奇/偶方式
- 第 3 位 : 颜色比较方式可工作(1=可工作)
- 第 2 位 : 保留
- 第 1—0 位 : 写方式: 0——处理器直接写(可能使用数据移位, 置位/复位)
  - 1——锁存器内容作为写数据
  - 2——用处理器数据的第  $n(0-3)$  位值填写第  $n$  个色平面
  - 3——没有使用

在写方式位的应用中, 通常使用的是写方式 0(直接写), 它也是图形控制器的省缺状态。该方式允许处理器数据直接写进显示存储器, 同时允许使用置位/复位、移位、屏蔽、与(AND)、或(OR)和异或(XOR)等功能。

写方式 1(处理器锁存器内容作为写数据) 可用来快速地把数据块从显示存储器的一个位置复制到另一个位置。处理器对显示存储器的一次读操作, 将从 4 个颜色平面中的每一个读一个字节的的数据, 并将这 32 位数据锁存到处理器的锁存器中。处理器向显示存储器的一次写操作将把 4 个字节的数据同时写回显示存储器的另一个地址(只要所有 4 个颜色平面都允许写)。该方式可以用于构造屏幕块快速复制或移动函数(参看程序 1-9)。

写方式 2(用第  $n$  位填写平面  $n$ ) 与位屏蔽寄存器一起使用时, 将把压缩的像素转换成平面像素并写进颜色平面。

**混合寄存器(索引号 06H)** 由 BIOS 方式选择操作初始化后, 通常不应该由应用程序修改。

**颜色忽略寄存器(索引号 07H)** 应与颜色比较方式配合使用。该寄存器在颜色比较周期中屏蔽特定的平面不被测试。该寄存器的位定义为

- 第 7—4 位 : 保留
- 第 3 位 : 忽略平面 3
- 第 2 位 : 忽略平面 2
- 第 1 位 : 忽略平面 1
- 第 0 位 : 忽略平面 0

**位屏蔽寄存器(索引号 08H)** 用来屏蔽某些位, 使其在读写周期中不被修改。位屏蔽寄存器的某位为 0 表示在显示器的一次处理器写期间, 该位的数据将来自处理器锁存器, 而不是来自处理器的输出数据。为了起到屏蔽功能, 在进行写操作之前必须经过一次读操作使锁存器装入正确的数据。

图形控制器的各寄存器对屏幕操作的影响最大, 下面我们举几个 HANENV 中的应用例子。

#### 程序 1-7 将屏幕映象块保存到内存中。

本程序的要点是直接将显示存储器 VRAM 中的数据读入内存。为了实现该功能, 我们应用了 EGA/VGA 的图形控制器中的读平面选择寄存器来逐一读取 4 个颜色平面的数据。

```

/* -----
   函数 getblock : 保存屏幕映象块。
   ----- */
#include <hanenv.h>
#include <alloc.h>

char ** getblock(col,line,width,high)
int col;          /* 屏幕图象块左上角列坐标(以字节为单位) */
int line;         /* 屏幕图象块左上角行坐标(以像素为单位) */
int width;        /* 屏幕图象块宽度(以字节为单位) */
int high;         /* 屏幕图象块高度(以像素为单位) */
{
    char ** block; /* 屏幕图象块存储缓冲区 */
    unsigned i,j;
    unsigned blocksize = high * width;
    char * p;

    /* --- 如果内存空间不够则退出 --- */
    if(coreleft() < blocksize * 4 + 16)
        return NULL;

    /* --- 为缓冲区分配空间 --- */
    block = (char ** )malloc(sizeof(char *) * 4);

    /* --- 设置显示寄存器操作标志 --- */
    _VideoBusy = YES;

    /* --- 分别拷贝屏幕图象块的 4 个颜色平面 --- */
    for(i=0;i < 4;i++)
    {
        /* --- 为屏幕映象块的每个位平面申请存储 --- */
        block[i] = (char *)malloc(blocksize);
        p = block[i];

        /* --- 选择颜色平面 i --- */
        outportb(0x3ce,4);
        outportb(0x3cf,i);
        for(j=0;j < high;j++)
        {
            movedata(0xa000,(line+j+_ScreenTop) * _ScreenWidth + col,
                    FP_SEG(p),FP_OFF(p),width);
            p += width;
        }
    }
}

```



```

    }
}

/* ——恢复读平面选择寄存器为省缺值 —— */
outportb(0x3ce,4);
outportb(0x3cf,0);
_VideoBusy = NO;
return block;
}

```

该函数的说明见程序 1-6。

#### 程序 1-8 在显示存储器中移动屏幕映象块。

EGA/VGA 的写方式 1 允许在显示存储器中直接传输数据。由于这种传输是 4 个位平面同时进行的,即一次传输 32 字节的数据,所以速度很快。设置写方式使用图形控制器中的方式寄存器。

```

/* —— */
    函数 _MoveImage : 在显示存储器中移动屏幕映象块。
    —— */
#include <hanenv.h>

void _Cdecl _MoveImage(col1,line1,width,high,col2,line2)
int col1;          /* 屏幕图象块左上角列坐标(以字节为单位) */
int line1;         /* 屏幕图象块左上角行坐标(以像素为单位) */
int width;         /* 屏幕图象块宽度(以字节为单位) */
int high;          /* 屏幕图象块高度(以像素为单位) */
int col2;          /* 屏幕图象块新位置列坐标(以字节为单位) */
int line2;         /* 屏幕图象块新位置行坐标(以像素为单位) */
{
    register int i,j;
    unsigned char tmpreg;
    unsigned char far *sour;
    unsigned char far *dest;

    /* —— 设置显示寄存器操作标志 —— */
    _VideoBusy = YES;

    /* —— 确定源屏幕块和目标屏幕块的首地址 —— */
    sour      = MK_FP(0xa000,(line1+_ScreenTop)*_ScreenWidth+col1);
    dest      = MK_FP(0xa000,(line2+_ScreenTop)*_ScreenWidth+col2);
}

```

```

/* ----- 修改方式寄存器内容为写方式1 ----- */
outportb(0x3ce,0x05);
tmpreg   = inportb(0x3cf);
tmpreg   |= 0x01;
outportb(0x3cf,tmpreg);

/* ----- 如果源块在目标块之后,正向拷贝 ----- */
if(sour > dest)
{
    for(i=0;i < high;i++)
    {
        for(j=0;j < width;j++)
            *(dest+j) = *(sour+j);
        dest += _ScreenWidth;
        sour += _ScreenWidth;
    }
}

/* ----- 如果源块在目标块之前,反向拷贝 ----- */
else
{
    sour += (high-1) * _ScreenWidth + width - 1;
    dest += (high-1) * _ScreenWidth + width - 1;
    for(i=high-1;i >= 0;i--)
    {
        for(j=0;j < width;j++)
            *(dest-j) = *(sour-j);
        dest -= _ScreenWidth;
        sour -= _ScreenWidth;
    }
}

/* ----- 恢复方式寄存器内容为写方式0 ----- */
tmpreg &= 0xfc;
outportb(0x3ce,0x05);
outportb(0x3cf,tmpreg);
_VideoBusy = NO;
}

```

该函数用于移动屏幕上的矩形区域。为了使得在移动前后矩形区域有部分重合时也能正确地操作,在程序中分别考虑了两种拷贝方向。应该说明的是该函数的操作范围可以覆盖整个显示存储区,并不只限于屏幕的可见部分(0—479行)。因此,我们可以使用该函数

利用 450—818 行暂时存储屏幕块,这样做比使用 getblock() 和 putblock() 速度要快,而且不占用内存。但要注意的是如果使用了屏幕平滑移动功能后,屏幕可见部分会超过 0—479 行。有关屏幕平滑移动功能的介绍请参看 2.2。

### 程序 1-9 设置写模式。

所谓写模式是指数据写入显示存储器时与显示存储器原来数据的逻辑关系,共有 4 种:

覆盖:新数据代替原数据

与:新数据和原数据做与操作

或:新数据和原数据做或操作

异或:新数据和原数据做异或操作

其中覆盖模式为省缺模式,应用最多。但有时我们希望达到这样一种效果:无论屏幕上原来的内容是什么,新写上去的内容都应可见。显然,如果屏幕上原来内容的颜色和新写数据的颜色相同时采用覆盖模式是不行的。这时可以采用异或模式。异或模式的特点是只要新数据和原数据均不为 0,则其结果既不是原数据的颜色,也不是新数据的颜色。

```
/* -----
   函数 _SetWriteMode: 设置写模式。
   ----- */
#include <dos.h>

#define PUT          0
#define AND_MODE    0x08
#define OR_MODE     0x10
#define XOR_MODE    0x18

void _Cdecl _SetWriteMode(write_mode)
int write_mode;      /* 写模式 */
{
    outportb(0x3ce, 3);
    outportb(0x3cf, write_mode);
}
```

关于使用该函数的例子可以参看第 9 章有关写光标的部分。

### 程序 1-10 清屏幕块。

使用置位/复位和允许置位/复位寄存器可以实现在屏幕上画一矩形块。由程序中可以看出,4 个位平面上的同一地址的数据是同时处理的,因而有较快的处理速度。

```
/* -----
   函数 _Block: 在屏幕上指定位置显示一矩形块。
   ----- */
#include <hanenv.h>
```

```

void _Cdecl _Block(col,line,width,high,color)
int col;          /* 屏幕矩形块左上角列坐标(以字节为单位) */
int line;         /* 屏幕矩形块左上角行坐标(以像素为单位) */
int width;        /* 屏幕矩形块宽度(以字节为单位) */
int high;         /* 屏幕矩形块高度(以像素为单位) */
int color;        /* 屏幕矩形块颜色 */
{
    register char far * addr;
    register i,j;

    /* --- 设置显示寄存器操作标志 --- */
    _VideoBusy = YES;

    /* --- 确定块在显示存储器(VRAM)中的地址 --- */
    addr = (char far *)0xa0000000+(line+_ScreenTop)*_ScreenWidth+col;

    /* --- 设置块颜色 --- */
    outportb(0x3ce,0);
    outportb(0x3cf,color);

    /* --- 颜色数据由置位/复位寄存器提供 --- */
    outportb(0x3ce,1);
    outportb(0x3cf,0x0f);

    for(i=0;i<high;i++)
    {
        for(j=0;j<width;j++)
            *(addr+j)%4 = 0xff; /* --- 先读后写 --- */
        addr += _ScreenWidth;
    }

    /* --- 恢复置位/复位和允许置位/复位寄存器 --- */
    outportb(0x3ce,1);
    outportb(0x3cf,0);
    outportb(0x3ce,0);
    outportb(0x3cf,0x0f);
    _VideoBusy = NO;
}

```

该函数的功能为在屏幕上的指定位置显示一矩形块。该函数的特点是运行速度快,但其列坐标和宽度只能使用8的倍数。因此该函数常用来快速清除屏幕上的矩形区域,如汉字

或字符的背景。对于那些需要使用精确位置和宽度的应用场合,可以选用另一函数 Bar。

### 1.4.3 CRT 控制器

CRT 控制器通过产生各种定义显示光栅的同步和消隐信号来控制 CRT。CRT 控制器也定义屏幕上显示数据的格式。CRT 控制器的寄存器组使用两个 I/O 地址(单色模式是 3B4 和 3B5,彩色模式是 3D4 和 3D5)。对彩色模式来说,索引寄存器地址为 3D4,用来选择 CRT 控制器的 24 个内部寄存器,而通过地址 3D5 可以对选中的寄存器进行读写操作。CRT 控制器的各寄存器见表 1-7。

表 1-7 CRT 控制器的寄存器

索引号	寄存器名称
00	* 水平总数寄存器
01	* 允许水平显示结尾寄存器
02	* 水平消隐信号起点寄存器
03	* 水平消隐信号结尾寄存器
04	* 水平回扫起点寄存器
05	* 水平回扫结尾寄存器
06	* 垂直总数寄存器
07	* 溢出寄存器
08	预置行扫描寄存器
09	最大扫描线数目寄存器
0A	光标起始寄存器
0B	光标中止寄存器
0C	起始地址寄存器(高字节)
0D	起始地址寄存器(低字节)
0E	光标位置寄存器(高字节)
0F	光标位置寄存器(低字节)
10	* 垂直回扫起点寄存器
11	* 垂直回扫结尾寄存器
12	* 允许垂直显示结尾寄存器
13	逻辑屏幕宽度/偏移量寄存器
14	下划线位置寄存器
15	* 垂直消隐信号起点寄存器
16	* 垂直消隐信号结尾寄存器
17	* 方式控制寄存器
18	行比较寄存器

要特别注意的是 CRT 控制器的多数寄存器在应用程序中修改是有危险的,可能损坏 CRT 显示器,所以要特别谨慎。在表 1-7 中我们将有危险的寄存器用“\*”号标记出来。然而不幸的是其中的溢出寄存器还是一个特别有用的寄存器。在 HANENV 系统中,我们通过该寄存器和预置行扫描寄存器、起始地址寄存器、逻辑屏幕宽度寄存器、行比较寄存器、最大扫描线数目寄存器以及其他寄存器组的一些寄存一起实现屏幕分割和屏幕平滑移动功能。下面我们简单介绍一下这些寄存器的调用方法。

**溢出寄存器(索引号 07H)** 在大多数情况下可以忽略。但 EGA/VGA 的几个 CRT 控制器的定时寄存器是 9 位或 10 位寄存器,它们的最高位就存放在溢出寄存器中。该寄存器的位定义为

- 第 7 位:垂直回扫起点寄存器的第 9 位(仅对 VGA)
- 第 6 位:允许垂直显示结尾寄存器的第 9 位(仅对 VGA)
- 第 5 位:垂直总数寄存器的第 9 位(仅对 VGA)
- 第 4 位:行比较寄存器的第 8 位
- 第 3 位:垂直消隐信号起点寄存器的第 8 位
- 第 2 位:垂直回扫起点寄存器的第 8 位
- 第 1 位:允许垂直显示结尾寄存器的第 8 位
- 第 0 位:垂直总数寄存器的第 8 位

在修改溢出寄存器某位时要特别小心,要保证所有其他位的值受到保护。

**最大扫描线数目寄存器(索引号 09H)** 的第 6 位存放着行比较寄存器的第 9 位(仅对 VGA 显示卡)。

**起始地址寄存器(索引号 0CH 和 0DH)** 共 16 位,存放第一条扫描线在显示存储器 VRAM 中的地址。通过修改该寄存器的值,可以移动屏幕上显示内容的位置(以字节为单位)。在移动量小于 8 位时应修改属性控制器中的水平移动寄存器的内容。

**逻辑屏幕宽度寄存器(索引号 13H)** 确定每一逻辑行的宽度(可以大于实际屏幕宽度)所占用的显示存储器字数(1 字=2 字节)。在默认方式下对于 80 列(640 点)模式此值为 28H(40)。在实现屏幕平滑滚动时利用该寄存器确定逻辑屏幕的宽度。

**行比较寄存器(索引号 18H,省缺设置为 FFH)** 与起始地址寄存器配合使用可以实现屏幕分割功能。此时屏幕被分为上、下两个部分,上面的部分显示由起始地址寄存器指向的内容,可以平滑滚屏,下面的部分显示在显示存储器(VRAM)起始地址开始的内容,不能进行平滑滚屏。在 VGA 显示卡上该寄存器实际上有 10 位,第 08H 位存放在溢出寄存器的第 4 位,而最高位(09H 位)在最大扫描线数目寄存器(索引号 09H)的第 6 位中。

CRT 控制器中的寄存器在 HANENV 中用得不多,主要集中在实现屏幕分割、逻辑屏幕和平滑滚动等功能方面。

#### 程序 1-12 设置逻辑屏幕宽度。

设置逻辑屏幕宽度可以使用 CRT 控制器中的逻辑屏幕宽度/偏移量寄存器。注意此时逻辑屏幕宽度是以字为单位的,即如果设置逻辑屏幕宽度为 80 各字节,则应向口地址 0x3d5 中送入数据 40(28H)。同时要注意的是在内存的 40:00 处开始的 256 个字节中存放着 BIOS 的一些重要数据,其中由 40:4A 开始的一个字中存放着屏幕的宽度数据,供 BIOS 使用。因此我们修改逻辑屏幕宽度时也要同时修改这个数据。

在 HANENV 系统的 init\_hanenv()函数中有如下程序段用于设置逻辑屏幕宽度。

```
/* -----
   设置逻辑屏幕宽度。
   ----- */
```



```

outportb(0x3d4, 0x13);
outportb(0x3d5, _ScreenWidth/2);
poke(0, 0x44a, _ScreenWidth);

```

#### 1.4.4 属性控制器

属性控制器控制显示的属性。就图形模式而言,颜色是唯一可能进行控制的属性。而对于字符模式来说,还可能需要控制诸如闪烁、下划线之类的属性。

属性控制器内部包括 20 个输出寄存器,复用 1 个 I/O 地址,这一点和 EGA/VGA 其他部分的寄存器有所不同。在属性控制器中索引寄存器和数据寄存器轮流使用 I/O 地址 3C0H。实际上,在属性控制器中有一个内部触发器每执行一次写操作即被触发一次,因此可以轮流选择索引寄存器和数据寄存器。对 I/O 地址 3DAH(彩色方式)或者 3BAH(单色方式)进行一次写操作可以将该触发器初始化为指向索引寄存器。

属性控制器的索引寄存器的位定义为

第 7—6 位:保留

第 5 位: 0 = 可以修改调色板,显示消失

1 = 不能修改调色板,可以显示

第 4—0 位:属性控制器的寄存器号(00H — 13H)

属性控制器的寄存器见表 1-8。

表 1-8 属性控制器的寄存器

索引号	寄存器名称
00—0F	调色板寄存器 0—15
10	方式控制寄存器
11	屏幕边缘颜色寄存器
12	允许颜色平面寄存器
13	水平移动寄存器
14	颜色选择寄存器

在 HANENV 系统中我们采用调用 BIOS 中断的方式操纵调色板寄存器和屏幕边缘颜色寄存器,参看程序 1-2、程序 1-3、程序 1-4 和程序 1-5。在实现屏幕平滑滚动时要用到水平移动寄存器,其位定义为

省缺设置:00H

第 7—4 位:保留

第 3—0 位:水平移动的像素数(对 VGA 模式 12H 来说可以是 0—7 位)

水平移动寄存器允许在水平方向上一次移动 1—7 个像素点,而 CRT 控制器中的起始地址寄存器允许显示器每次移动 8 的倍数个像素。将两者结合起来就可以实现任意数量的水平平滑移动。需要注意的是对属性控制器的寄存器的所有操作都必须在垂直回扫期间进行(参看 1.4.5)。

#### 1.4.5 外部寄存器

在 EGA/VGA 的图形控制器、时序发生器、CRT 控制器和属性控制器这几个主要功能模块之外还有一些寄存器,通称外部寄存器。在外部寄存器中,我们只介绍一个输入状态寄存器。输入状态寄存器的 I/O 地址为 3BAH(单色方式)/3DAH(彩色方式),其位定义为

第 7—6 位:没用

第 5—4 位:诊断

第 3 位:垂直回扫

第 2—1 位:光笔开关和选通(仅对 EGA)

第 0 位:显示器可工作

该寄存器的第 3 位为垂直回扫位,该位为 1 时表示处于垂直回扫期间。EGA/VGA 的某些操作(如修改属性控制器中的寄存器)必须在垂直回扫期间进行,我们可以通过测试该位来确定合适的操作时机。

### 1.5 EGA/VGA 卡使用小结

EGA/VGA 显示器适配卡是微机上结构最复杂的设备之一。为了适应不同应用问题的需要,为 EGA/VGA 卡设计了若干种显示模式。通常 DOS 工作在字符模式之下。但是,在字符模式下可以显示的字符集是固定的(西文 ASCII 字符集),很难直接用于显示汉字。因此在几乎所有的中文操作系统(当然也包括 HANENV 系统)中都是通过使用图形显示模式模拟字符显示方式的方法来达到显示汉字的目的。而在 VGA 卡的众多图形模式中,模式 12H 以其较高的分辨率和色彩数获得了最广泛的使用。

从操作的角度来看,对显示器的操作也就是对显示存储器 VRAM、EGA/VGA 寄存器和 BIOS 的 10H 号中断的操作。对 BIOS 中断的操作使用方便、安全,但速度稍慢,最适用于实现使用频度较低的功能,例如显示模式的转换、调色板的设置等。如果将其用于显示汉字则效果不佳。

使用 EGA/VGA 的寄存器来控制对屏幕的读写操作是最基本、同时也是最有效的手段。尤其是用图形控制器中的寄存器和对显示存储器的读写相配合来实现汉字的显示,可以达到最佳效果。但是要特别注意的是对有些 EGA/VGA 的寄存器的操作是有危险的,使用不当可能破坏显示器的工作状态,甚至可能损坏显示器。

## 鼠标应用程序设计

随着图形用户界面的流行,鼠标已经成为微型计算机的标准设备。目前市面上流行的鼠标有多种类型,从结构上来看有机械式、光电式和机械光电结合式等几种,从工作方式来看主要有 IBM 公司的三按钮 5 字节鼠标和 Microsoft 公司的两按钮 3 字节鼠标两大类。通常鼠标通过串行口和主机连接。在把鼠标接到主机上之后,还要执行相应的鼠标驱动程序之后方可使用。鼠标驱动程序有两种形式:一种是作为设备驱动程序,文件名的后缀为 .SYS,必须将其加入 config.sys 才能使用。另一种是可执行文件,文件名后缀为 .COM,既可以直接在命令行运行,也可以将其加入批处理文件 autoexec.bat 中。关于鼠标的安装细节,请参看鼠标的说明书。

Microsoft 公司为在 PC 机上使用的鼠标提供了一个标准的功能调用软件中断接口,只要加载了支持该标准的鼠标驱动程序,应用程序即可无需过问鼠标的类型和工作方式,直接通过该软件接口对鼠标进行操纵。Microsoft 把 INT33H 定义为鼠标软件中断,其使用方法和其他 BIOS 的中断调用完全相同。鼠标中断 INT33H 的主要功能调用可参看表 2-1。

表 2-1 鼠标中断 INT33H 的主要功能调用

功能号	功 能 调 用
00H	初始化鼠标
01H	显示鼠标光标
02H	关闭鼠标光标
03H	读取按钮状态及鼠标位置
04H	设置鼠标光标位置
05H	读取按钮按下信息
06H	读取按钮释放信息
07H	设置鼠标光标的横向移动范围
08H	设置鼠标光标的纵向移动范围
09H	设置图形模式的鼠标光标图形
0AH	设置字符模式的鼠标光标图形
0BH	读取鼠标移动距离
0CH	设置用户定义的事件处理子程序

功能号	功 能 调 用
0DH	打开光笔仿真开关
0EH	关闭光笔仿真开关
0FH	设置鼠标移动速率
10H	设置鼠标隐藏条件
14H	交换用户定义的事件处理子程序
15H	读取存储鼠标驱动程序状态所需存储字节数
16H	存储鼠标驱动程序状态
17H	恢复鼠标驱动程序状态
18H	设置替换用的用户定义的事件处理子程序
19H	读用户定义的事件处理子程序的地址
1AH	设置鼠标灵敏度
1BH	读取鼠标灵敏度
1CH	设置鼠标中断速度
1DH	设置显示页码
1EH	读取显示页码
1FH	关闭鼠标驱动程序
20H	开放鼠标驱动程序
21H	软件初始化
24H	读取鼠标驱动程序版本号、鼠标类型和中断号
25H	读取鼠标驱动程序有关信息

在以下各节中,我们结合 HANENV 系统中的鼠标应用函数族的设计介绍 INT33H 的几个常用子功能的调用方法。

## 2.1 鼠标的初始化

初始化鼠标驱动程序使用 INT33H 的 00H 号子功能。该子功能的调用方法如下:

入口寄存器设置:

AX = 00H

返回寄存器

AX = 鼠标是否安装成功(=0:安装失败,=1:安装成功)

BX = 鼠标按钮数目(对于使用 Microsoft 鼠标驱动程序的鼠标来说,不管实际上鼠标有几个按钮,此值总是 2)

### 程序 2-1 初始化鼠标驱动程序。

```

/* -----
   函数 reset_mouse : 初始化鼠标
   ----- */
#include <hanenv.h>

```

```

int _HaveMouse = NO;                                /* 鼠标装载成功标记 */

void reset _mouse(void)
{
    int mouse_off = FP_OFF(_MouseHandle); /* 光标处理子程序偏移量 */
    int mouse_seg = FP_SEG(_MouseHandle); /* 光标处理子程序段地址 */

    _AX = 0;                                           /* 初始化鼠标 */
    geninterrupt(0x33);
    _HaveMouse = _AX;                                  /* 鼠标安装是否成功 */
    if(_HaveMouse)                                    /* 安装光标处理子程序 */
    {
        _CX = 1;
        _DX = mouse_off;
        _BX = mouse_seg;
        _ES = _BX;
        _AX = 0x0c;                                   /* 调鼠标中断 0CH 号功能 */
        geninterrupt(0x33);
    }
}

```

本函数的后半部分为调用 INT33H 的 0CH 号功能安装了一个能被鼠标驱动程序识别的特殊处理子程序。该子程序的处理功能可以对应于鼠标按钮的按下与释放、光标位置的变化以及这些事件的组合。实际上,该子程序应该被设计为中断子程序,而一旦设置成功以后,便由鼠标驱动程序接管而执行。INT33H 的 0CH 号子功能的调用方法如下:

入口寄存器设置:

AX = 0CH

CX = 调用掩码

ES:DX = 用户定义的事件处理子程序的地址

返回寄存器: 无

其中调用掩码字节中的各位为 1 的定义如下:

第 7—5 位: 未用

第 4 位: 右按钮释放

第 3 位: 右按钮按下

第 2 位: 左按钮释放

第 1 位: 左按钮按下

第 0 位: 鼠标移动了

在 HANENV 系统中我们使用这种方法重新定义了自己的鼠标光标驱动程序,即函数 \_MouseHandle。在安装 \_MouseHandle 时我们使用了掩码 01H,即激活该鼠标光标驱动程序的条件为“鼠标移动了”。

应该注意的是在 HANENV 的 init\_hanenv 函数中已经加入了对本函数的调用,所以一

般情况下在应用程序中无须再调用函数 `reset_mouse`。

## 2.2 自制鼠标光标驱动

我们在设计 HANENV 系统的屏幕显示操作函数时发现 INT33H 中的鼠标光标有几个缺点。首先,INT33H 为了实现鼠标光标的快速移动,使用了图形控制器的写方式 01H(即数据在显示存储器内移动,参看 1.4.2),并将鼠标光标的缓冲区定在显示存储器 VRAM 的中段(约 510 行左右)。这样就妨碍了我们将整个显示存储器 VRAM 提供给应用程序设计人员以实现屏幕内容的快速拷贝、分屏显示和屏幕平滑滚动等功能。其次,在 1.4.3 中我们介绍过可以设置一个比屏幕实际宽度要宽的逻辑屏幕,以之作为屏幕横向平滑滚动的基础。但是 INT33H 的鼠标光标在这种显示方式下无法正常显示。因此我们重新编写了一个鼠标光标驱动函数 `_MouseHandle`,通过 INT33H 的设置用户自定义的事件处理子程序的子功能(0CH)将其装入鼠标驱动程序,以代替原来的鼠标光标(参看程序 2-1)。此外,使用 `_MouseHandle` 代替原来的鼠标光标驱动还有可以改变鼠标光标的颜色等优点。

使用 INT33H 的 0CH 号子功能安装好用户定义的事件处理子程序后,每当该子程序被触发后,各寄存器的内容被设置为

- AX = 调用掩码中的事件引发位
- BX = 按钮状态
- CX = 鼠标光标的水平坐标
- DX = 鼠标光标的垂直坐标
- DI = 鼠标光标的水平方向移动量(以 mickey 为单位)
- SI = 鼠标光标的垂直方向移动量(以 mickey 为单位)

在子程序中可以通过检测这些寄存器的值了解当前的工作环境。

### 程序 2-2 自定义鼠标光标驱动子程序

我们设计的函数 `_MouseHandle` 用来代替 INT33H 的图形光标,以支持逻辑屏幕、平滑滚动等功能。该函数的处理结构为

```
如果鼠标移动了,则
{
    如果鼠标新位置已经超出当前屏幕范围,则
        在逻辑屏幕的范围内移动屏幕图象;
    恢复鼠标光标下原来的图象(抹去鼠标光标);
    存储鼠标光标新位置下的图象;
    在新位置上画出鼠标光标;
}
```

由于该函数是由鼠标驱动程序根据我们设置的鼠标事件(鼠标移动)激活的,所以该程序的执行条件就是鼠标移动。在进入该函数后,我们首先要记下寄存器 CX 和 DX 中的新的



鼠标位置(原来的鼠标位置在全局变量 `_MouseCol` 和 `_MouseRow` 中)。在逻辑屏幕的范围内移动屏幕图象、恢复鼠标光标下原来的图象、存储鼠标光标新位置下的图象以及画出鼠标光标分别由函数 `_MouseScroll`、`_SaveMouseBk`、`_LoadMouseBk` 和 `drawxyh` 完成。最后还要更新鼠标位置。

函数 `_MouseHandle` 实际上是接管了 BIOS 的时钟中断 1CH, 因此我们还要考虑两个问题: 中断函数的重复调用和现场保护。为了防止中断函数的重复调用, 我们设置了一个全局变量 `_MouseBusy`。如果在画鼠标光标的过程中由于中断再次引发对本进程的调用, 此时将不做任何操作而立即返回, 以免影响画鼠标光标的过程。这一点很象某些驻留程序的设计。

C 语言的中断类型函数已经处理了常规的现场保护工作, 如各寄存器值的存储与恢复均是自动进行的。但是 HANENV 中的中断函数还要考虑更多的现场保护工作。由于 HANENV 中有许多直接对 EGA/VGA 显示卡寄存器组的操作, 如果在这些操作正在进行时被中断程序打断, 将会使屏幕显示出现异常。因此我们设置了一个全局变量 `_VideoBusy`, 初值为 0, 当进行显示寄存器的操作时将其置为 1, 操作完毕再恢复为 0。在中断程序(除了 `_MouseHandle` 之外, 还有光标和时钟显示函数)中如果发现该变量的值非零(显示寄存器忙)则退出。

全局变量 `_MouseLight` 表示当前鼠标光标是否显示。

```
/* -----
函数 _MouseHandle : 鼠标光标驱动函数
说明: 自定义的鼠标事件处理程序在编写时应注意的几个问题:
1. 鼠标中断在调用本程序时修改了数据地址, 使之指向了鼠标
   中断自己的数据。由于在我们编写的鼠标光标驱动函数中需
   要使用全局变量, 所以在程序开始处必须恢复原来的数据段
   地址;
2. 该程序需要使用 RETF 指令返回, 需要注意的是低版本的
   汇编程序不支持该指令;
3. 在编写该程序时需要设置防止重新进入的标志, 这一点很象
   某些驻留程序。
----- */
unsigned char _MouseFonts[] =
{
    /* -- 鼠标光标边缘图象点阵 ----- */
    0xFFC0, 0x00, 0xFFA0, 0x00, 0xFF90, 0x00, 0xFF88, 0x00,
    0xFF84, 0x00, 0xFF82, 0x00, 0xFF81, 0x00, 0xFF80, 0xFF80,
    0xFF80, 0x40, 0xFF83, 0xFFE0, 0xFF92, 0x00, 0xFFA9, 0x00,
    0xFFC9, 0x00, 0x04, 0xFF80, 0x04, 0xFF80, 0x03, 0x00,

    /* -- 鼠标光标图象点阵 ----- */
    0x00, 0x00, 0x40, 0x00, 0x60, 0x00, 0x70, 0x00,
    0x78, 0x00, 0x7C, 0x00, 0x7E, 0x00, 0x7F, 0x00,

```

```

0x7F,0xFF80,0x7C,0x00,0x6C,0x00,0x46,0x00,
0x06,0x00,0x03,0x00,0x03,0x00,0x00,0x00
};

extern void _MouseScroll(int x,int y); /* 鼠标控制移动屏幕 */
extern void _SaveMouseBk(void); /* 存储鼠标光标位置的背景 */
extern void _LoadMouseBk(void); /* 恢复鼠标光标位置的背景 */

/* --- 中断函数 _MouseHandle : 鼠标光标驱动函数 --- */
void interrupt _MouseHandle(void)
{
    unsigned char old03;
    int newcol,newrow;
    int cursorlight;

    asm push ds
    asm push ax
    asm mov ax, _DATA
    asm mov ds,ax
    asm pop ax
    newcol = _CX;
    newrow = _DX;
    cursorlight = iscursorlight();

    /* --- 如果鼠标光标亮且显示寄存器组不忙 --- */
    if(_MouseLight && !_MouseBusy && !_VideoBusy)
    {
        /* --- 设置防止重复调用标志 --- */
        _VideoBusy = YES;

        /* --- 如果正文光标亮,关闭正文光标 --- */
        if(cursorlight)
            delightcursor();
        disable();

        /* --- 在逻辑屏幕的范围内移动屏幕图象 --- */
        _MouseScroll(newcol,newrow);

        /* --- 恢复鼠标光标下原来的图象 --- */
        _LoadMouseBk();

        /* --- 更新鼠标位置 --- */
        _MouseCol = newcol;

```

```

    _MouseRow = newrow - _ScreenTop;

    /* ----- 存储鼠标光标新位置下的图象 ----- */
    _SaveMouseBk();

    /* ----- 设置直接写模式 ----- */
    outportb(0x3ce, 0x03);
    old_03 = inportb(0x3cf);
    outportb(0x3cf, 0x00);

    /* ----- 画出鼠标光标 ----- */
    _DrawF(_MouseCol, _MouseRow, 2, 16, _MouseBk, _MouseFonts);
    _DrawF(_MouseCol, _MouseRow, 2, 16, _MouseColor, _MouseFonts + 32);

    /* ----- 恢复写模式寄存器原来的值 ----- */
    outportb(0x3ce, 0x03);
    outportb(0x3cf, old03);

    /* ----- 如果原来正文光标亮, 点亮之 ----- */
    enable();
    if(cursorlight)
        lightcursor();

    /* ----- 撤消防止重复调用标志 ----- */
    asm pop ds
    asm retf
    _VideoBusy = NO;
}
}

```

在调用 INT33H 的 0H 号子功能初始化鼠标驱动程序时整个调用掩码将复位为 0。因此在应用程序中设置了用户定义的事件处理子程序之后, 在结束程序运行之前应使用 INT33H 的 0H 号子功能恢复鼠标驱动程序的初始状态。在 HANENV 系统的 close\_hanenv 函数中, 有如下程序段用于恢复鼠标驱动程序的初始状态:

```

/* -----
恢复鼠标驱动程序的初始状态
----- */

_AX = 0;
geninterrupt(0x33);

```

函数 \_MouseHandle 还调用了几个内部函数, 下面我们介绍其中几个函数的设计。

### 程序 2-3 存储鼠标光标位置下的图象。

为了迅速更新鼠标光标的图象,我们采用了 EGA/VGA 的写方式 01H(显示存储器内写,参看 1.1.2)将鼠标光标位置下的图象存储在显示存储器的最底端。由于对显示存储器 VRAM 的读写是以字节为单位的,而鼠标光标的移动却是以象素为单位的,所以我们在存储鼠标光标下的背景图象时便多存了一些,有 3 字节(24 象素)宽,16 象素高,共占用了 48 字节。因此显示存储器 VRAM 的最后 48 个字节不能移作其他用途。

```

/* -----
   函数 _SaveMouseBk : 存储鼠标光标的背景
   ----- */
#include "hanenv.h"

void _SaveMouseBk(void)
:
    /* ----- 鼠标当前位置在显示存储器中的地址 ----- */
    char far *sour = MK_FP(0xa000,(_MouseRow
                                +_ScreenTop)*_ScreenWidth+_MouseCol/8);

    /* ----- 显示存储器最后 48 个字节的地址 ----- */
    char far *dest = MK_FP(0xa000,0xffd0);
    int i;
    unsigned char tmpreg;

    /* ----- 设置写方式 01H ----- */
    outportb(0x3ce,0x05);
    tmpreg = inportb(0x3cf);
    tmpreg |= 0x01;
    outportb(0x3cf,tmpreg);

    /* ----- 显示存储器内数据拷贝 ----- */
    for(i=0;i<16;i++)
    {
        *dest++ = *sour;
        *dest++ = *(sour+1);
        *dest++ = *(sour+2);
        sour += _ScreenWidth;
    }

    /* ----- 恢复写方式 00H ----- */
    tmpreg &= 0xfc;
    outportb(0x3ce,0x05);

```

```
    outportb(0x3cf,tmpreg);
}
```

#### 程序 2-4 恢复鼠标光标位置下的图象。

该函数用于恢复由函数\_SaveMouseBk 保存的图象。

```
/* -----
   函数 _LoadMouseBk : 恢复鼠标光标的背景
   ----- */
#include <hanenv.h>

void _LoadMouseBk(void)
{
    /* ----- 显示存储器最后 48 个字节的地址 ----- */
    char far *sour = MK_FP(0xa000,0xfd0);

    /* ----- 鼠标当前位置在显示存储器中的地址 ----- */
    char far *dest = MK_FP(0xa000,(_MouseRow
                                   +_ScreenTop)*_ScreenWidth+_MouseCol/8);

    int i;
    unsigned char tmpreg;

    /* -- 设置写方式 01H ----- */
    outportb(0x3ce,0x05);
    tmpreg = inportb(0x3cf);
    tmpreg &= 0xfc;
    tmpreg |= 0x01;
    outportb(0x3cf,tmpreg);

    /* -- 显示存储器内数据拷贝 ----- */
    for(i=0;i<16;i++)
    {
        *dest      = *sour++;
        *(dest+1) = *sour++;
        *(dest+2) = *sour++;
        dest      += _ScreenWidth;
    }

    /* -- 恢复写方式 00H ----- */
    tmpreg &= 0xfc;
    outportb(0x3ce,0x05);
    outportb(0x3cf,tmpreg);
}
```

### 程序 2-5 鼠标控制的在逻辑屏幕的范围内移动屏幕图象函数。

该函数用于当鼠标的新位置已经超出当前屏幕范围时在逻辑屏幕的范围内移动屏幕图象。在 HANENV 中逻辑屏幕与实际屏幕窗口的关系如图 2-1 所示。由图 2-1 可以看出，逻辑屏幕宽度 `_ScreenWidth` 可以大于屏幕窗口的宽度 (640 个象素)，而逻辑屏幕高度 `_ScreenHigh` 的最大值可用下式计算：

$$\_ScreenHigh = (VRAMSIZE - 48) / \_ScreenWidth - \text{分屏后的固定显示区高}$$

式中 VRAMSIZE 为显示存储器的页面尺寸，通常为 64K，而鼠标光标背景存储区占用了显示存储器的最后 48 个字节。

屏幕显示窗口可以在逻辑屏幕中移动，其活动范围不能超过逻辑屏幕的宽度和高度。

利用 EGA/VGA 显示卡的分屏功能，我们可以将屏幕显示窗口分为两个部分，其一是屏幕下方的固定显示窗口，用来显示汉字输入提示行和固定的屏幕菜单；另一部分是位于屏幕上部的可移动屏幕显示窗口，两部分屏幕显示窗口之和的尺寸为  $640 \times 480$  象素。固定显示窗口的内容从显示存储器的起始地址开始，而可移动屏幕显示窗口的显示内容从第 `_WindowTop` 行开始。

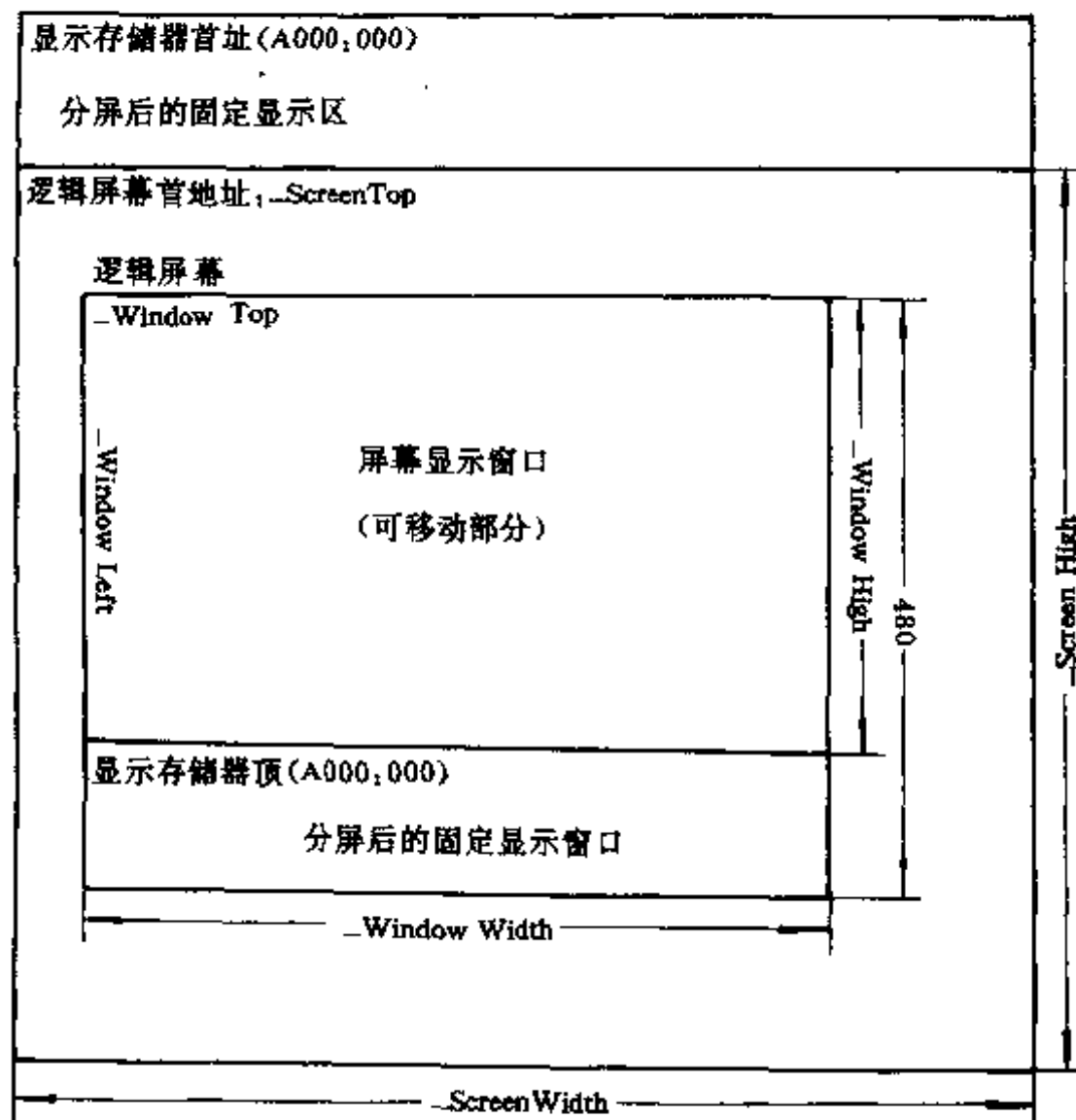


图 2-1 逻辑屏幕与实际屏幕窗口的关系

HANENV 系统的初始设置为



```

_ScreenTop    = 0
_ScreenWidth  = 80
_ScreenHigh   = 480
_WindowHigh   = 480
_WindowLeft   = 0
_WindowTop    = 0

```

即逻辑屏幕的尺寸和屏幕显示窗口的尺寸相同,屏幕显示窗口也没有进行分割。如果需要改变上述初始设置,可以在 `init_hanenv` 函数之前使用下列函数:

```

void set_screen_width(int width); /* 设置逻辑屏幕宽度 */
void set_screen_high(int high); /* 设置逻辑屏幕高度 */
void split_screen(int high); /* 设置分屏显示活动屏的高度 */

```

当逻辑屏幕的尺寸大于屏幕显示窗口的尺寸时,将鼠标光标移出屏幕显示窗口,即可在逻辑屏幕的范围内移动屏幕显示窗口。

```

/* -----
   函数 _MouseScroll : 在逻辑屏幕的范围内移动屏幕显示窗口
   ----- */
#include <hanenv.h>

/* -- 平滑移动屏幕显示窗口 ----- */
extern void _SmoothScroll(int x,int y);

void _MouseScroll(newcol,newrow)
int newcol;
int newrow;
{
    /* ----- 如果鼠标光标移出屏幕显示窗口 ----- */
    if(newcol < _WindowLeft || newcol > _WindowLeft+639 || newrow < _WindowTop
        || newrow > _WindowTop+_WindowHigh-1)
    {
        /* ----- 如果屏幕显示窗口应向左移 ----- */
        if(newcol < _WindowLeft)
        {
            _WindowLeft = newcol;
            if(_WindowLeft < 0)
                _WindowLeft = 0;
        }

        /* ----- 如果屏幕显示窗口应向右移 ----- */

```

```

if(newcol>_WindowLeft+639)
{
    _WindowLeft = newcol - 639;
    if(_WindowLeft>_ScreenWidth*8-640)
        _WindowLeft = _ScreenWidth*8-640;
}

/* ----- 如果屏幕显示窗口应向上移 ----- */
if(newrow<_WindowTop)
{
    _WindowTop = newrow;
    if(_WindowTop<_ScreenTop)
        _WindowTop = _ScreenTop;
}

/* ----- 如果屏幕显示窗口应向下移 ----- */
if(newrow>_WindowTop+_WindowHigh-1)
{
    _WindowTop = newrow-_WindowHigh;
    if(_WindowTop>_ScreenHigh-_WindowHigh+_ScreenTop)
        _WindowTop = _ScreenHigh-_WindowHigh+_ScreenTop;
}

/* ----- 移动屏幕显示窗口 ----- */
_SmoothScroll(_WindowLeft,_WindowTop);
}
}

```

由于我们使用了自己定义的鼠标光标驱动程序,所以也就不能直接使用 INT33H 的第 01H 和 02H 号子功能来打开和关闭鼠标光标了。程序 2-6 和程序 2-7 是配合自定义鼠标光标驱动程序的打开和关闭鼠标光标函数。

#### 程序 2-6 打开鼠标光标。

打开鼠标光标的过程是:首先,调用 INT33H 的 03H 号子功能查询鼠标的当前坐标;然后,存储相应位置上的背景图象,再画出鼠标光标;最后置鼠标光标标志为点燃状态。

```

/* -----
函数 light_mouse : 打开鼠标光标
----- */
#include <hanenv.h>

extern unsigned char _MouseFonts[];
extern void _SaveMouseBk(void); /* 存储鼠标光标位置的背景 */

```

```

void light _mouse(void)
{
    if(_HaveMouse && ! _MouseLight)
    {

        /* ----- 查当前鼠标坐标 ----- */
        _AX      = 3;
        geninterrupt(0x33);
        _MouseCol  = _CX;
        _MouseRow  = _DX;
        _MouseRow -= _ScreenTop;

        /* ----- 存储鼠标光标新位置下的图象 ----- */
        _VideoBusy = YES;
        _SaveMouseBk();
        _VideoBusy = NO;

        /* ----- 画出鼠标光标 ----- */
        _DrawF(_MouseCol, _MouseRow, 2, 16, _MouseBk, _MouseFonts);
        _DrawF(_MouseCol, _MouseRow, 2, 16, _MouseColor, _MouseFonts+32);

        /* ----- 设置鼠标光标点燃标志 ----- */
        _MouseLight = YES;
    }
}

```

#### 程序 2-7 关闭鼠标光标。

关闭鼠标光标的过程比较简单,只需先恢复鼠标光标下原来的背景图象,然后再置鼠标光标标志为熄灭即可。

```

/* -----
    函数 delight _mouse : 关闭鼠标光标
----- */
#include (hanenv.h)

extern void _LoadMouseBk(void);    /* 恢复鼠标光标位置的背景 */

void delight _mouse(void)
{
    if(_MouseLight)
    {

```

```

/* ----- 恢复鼠标光标下原来的图象 ----- */
_VideoBusy = YES;
_LoadMouseBk();
_VideoBusy = NO;
_MouseLight = NO;
}
}

```

## 2.3 测试和设置鼠标状态

本节我们介绍一组用于测试和设置鼠标状态的函数设计。在 HANENV 中，我们构造了自己的鼠标运行环境，包括一组全局变量和一组操作函数。因此，在设计鼠标操作函数时，不仅要考虑 INT33H 的有子功能，还要考虑到 HANENV 的具体情况，才能设计出结构简明、运行效率高的鼠标操作函数。

### 程序 2-8 测试鼠标按钮状态和鼠标光标位置。

测试鼠标按钮状态和鼠标光标位置可以使用 INT33H 的 03H 号子功能。该子功能的调用方法如下：

入口寄存器设置：

AX = 03H

返回寄存器

BX = 鼠标按钮状态

CX = 鼠标当前行坐标

DX = 鼠标当前列坐标

其中鼠标按钮状态各位的定义为

第 7 - 2 位：未用

第 1 位：右按钮状态(=1：按下)

第 0 位：左按钮状态(=1：按下)

在 HANENV 系统的头文件 hanenv.h 中，我们将鼠标按钮状态定义为一组宏：

```
#define LEFT_BUTTON 401
```

```
#define RIGHT_BUTTON 402
```

```
#define L_R_BUTTON 403
```

即在 INT33H 号子功能的鼠标按钮状态上加上常数 400 而得。之所以要加上 400，是因为我们在 HANENV 系统中将鼠标按钮状态也看成是一种广义的扩充键盘代码，这样就可以统一处理鼠标按钮状态和键盘信息了。在设计函数 get\_mouse\_status 时我们没有使用 03H 号子功能提供的鼠标当前行坐标和列坐标，而是直接使用由自定义鼠标光标驱动程序控制的全局变量 \_MouseCol 和 \_MouseRow 的值，以避免可能的误差。

```

/* -----
   函数 get_mouse_status : 测试鼠标按钮状态和鼠标光标位置
   ----- */
#include <hanenv.h>

int get_mouse_status(x,y)
int *x;                /* 当前鼠标光标列坐标 */
int *y;                /* 当前鼠标光标行坐标 */
{
    int status = 0;

    if(_MouseLight)
    {
        /* ----- 测试鼠标按钮状态 ----- */
        _AX = 3;
        geninterrupt(0x33);
        status = _BX;

        /* ----- 取鼠标当前坐标 ----- */
        *x = _MouseCol;
        *y = _MouseRow;

        /* ----- 将鼠标按钮状态变换为扩充键盘代码 ----- */
        if(status)
            status += 400;
    }
    return status;
}

```

#### 程序 2-9 测试鼠标光标的移动信息。

测试鼠标光标的移动信息可以使用 INT33H 的 0BH 号子功能。该子功能给出自上次调用本子功能以来鼠标移动的增量。其调用方法如下：

入口寄存器设置：

AX = 0BH

返回寄存器

CX = 鼠标移动行增量(单位为 mickey)

DX = 鼠标移动列增量(单位为 mickey)

```

/* -----
   函数 mouse_moved : 测试鼠标光标的移动信息
   ----- */

```

```

#include <dos.h>

void mouse_moved(dx,dy)
int *dx;                /* 当前鼠标光标列坐标 */
int *dy;                /* 当前鼠标光标行坐标 */
{
    _AX = 0x0b;
    geninterrupt(0x33);
    *dx = _CX;
    *dy = _DX;
}

```

#### 程序 2-10 测试鼠标按钮的按下信息。

测试鼠标按钮的按下信息可以使用 INT33H 的 05H 号子功能。该子功能的调用方法如下：

入口寄存器设置：

AX = 05H  
BX = 测试哪个按钮

返回寄存器：

AX = 各按钮是否按下  
BX = 按钮按下次数  
CX = 最后一次按下时鼠标行坐标  
DX = 最后一次按下时鼠标列坐标

调用该函数时参数 which\_button 应使用前述之鼠标按钮的宏定义。

```

/* -----
   函数 mouse_pressed : 测试鼠标按钮的按下信息
   ----- */
#include <hanenv.h>

int mouse_pressed(which_button,x,y,count)
int which_button;      /* 被测按钮 */
int *x;                /* 最后一次按下时鼠标行坐标 */
int *y;                /* 最后一次按下时鼠标列坐标 */
int *count;            /* 按钮按下次数 */
{
    int status = 0;
    if(_HaveMouse)
    {
        /* -- 测试鼠标按钮的按下信息 ----- */
        _BX = which_button-400;

```

```

    _AX    = 0x05;
    geninterrupt(0x33);
    status = _AX;
    *count = _BX;
    /* ----- 将鼠标按钮状态变换为扩充键盘代码 ----- */
    status += 400;

    /* ----- 取鼠标当前坐标 ----- */
    *x      = _MouseCol;
    *y      = _MouseRow;
}
return status;
}

```

### 程序 2-11 测试鼠标按钮的释放信息。

测试鼠标按钮的释放信息可以使用 INT33H 的 06H 号子功能。该子功能的调用方法如下：

入口寄存器设置：

AX = 06H

BX = 测试哪个按钮

返回寄存器：

AX = 各按钮是否释放

BX = 按钮释放次数

CX = 最后一次释放时鼠标行坐标

DX = 最后一次释放时鼠标列坐标

调用该函数时参数 which\_button 应使用前述之鼠标按钮的宏定义。

```

/* -----
   函数 mouse_release : 测试鼠标按钮的释放信息
   ----- */
#include <hanenv.h>

int mouse_release(which_button, x, y, count)
int which_button;          /* 被测按钮 */
int *x;                    /* 最后一次释放时鼠标行坐标 */
int *y;                    /* 最后一次释放时鼠标列坐标 */
int *count;                /* 按钮释放次数 */
{
    int status = 0;
    if(_HaveMouse)
    {

```



```

/* —— 测试鼠标按钮的释放信息 —— */
_BX    = which_button - 400;
_AX    = 0x06;
geninterrupt(0x33);
status  = _AX;
*count  = _BX;

/* —— 将鼠标按钮状态变换为扩充键盘代码 —— */
status += 400;

/* —— 取鼠标当前坐标 —— */
*x      = _MouseCol;
*y      = _MouseRow - _ScreenTop;
}
return status;
}

```

下面我们再介绍一组用于设置鼠标状态的函数设计。和测试鼠标状态类似，我们还是应用鼠标中断来设计这组函数。

#### 程序 2-12 设置鼠标光标坐标。

设置鼠标光标坐标可以使用 INT33H 的 04H 号子功能。该子功能的调用方法如下：

入口寄存器设置：

AX = 04H

CX = 鼠标行坐标

DX = 鼠标列坐标

返回寄存器：无

```

/* —— 函数 set_mouse_pos : 设置鼠标光标坐标 —— */
#include <dos.h>

extern int _HaveMouse;          /* 鼠标装载成功标记 */
extern int _MouseLight;        /* 当前鼠标光标是否显示 */
extern int _ScreenTop;         /* 当前屏幕首行位置 */

void set_mouse_pos(x,y)
int x;                          /* 鼠标光标列坐标 */
int y;                          /* 鼠标光标行坐标 */
{

```

```

if( _HaveMouse)
{
    int mouselight = _MouseLight;

    /* ----- 关闭鼠标光标坐标 ----- */
    if(mouselight)
        delight _mouse();

    /* ----- 转换行坐标为逻辑屏幕行坐标 ----- */
    x += _ScreenTop;

    /* ----- 设置鼠标光标坐标 ----- */
    _CX = x;
    _DX = y;
    _AX = 0x04;
    geninterrupt(0x33);

    /* ----- 若原来鼠标光标打开则仍打开 ----- */
    if(mouselight)
        light _mouse();
}
}

```

### 程序 2-13 设置鼠标活动范围。

设置鼠标活动范围可以使用 INT33H 的 07H 号子功能和 08H 号子功能。07H 号子功能的调用方法如下：

入口寄存器设置：

AX = 07H  
CX = 鼠标列坐标最小值  
DX = 鼠标列坐标最大值

返回寄存器：无

08H 号子功能的调用方法如下：

入口寄存器设置：

AX = 08H  
CX = 鼠标行坐标最小值  
DX = 鼠标行坐标最大值

返回寄存器：无

```

/* -----
函数 set _mouse _range : 设置鼠标活动范围
----- */

```

```

#include <dos.h>

extern int _HaveMouse;          /* 鼠标装载成功标记      */
extern int _ScreenTop;          /* 当前屏幕首行位置      */
void set_mouse_range(left,top,right,bottom)
int left;                       /* 鼠标活动窗口左上角列坐标 */
int top;                        /* 鼠标活动窗口左上角行坐标 */
int right;                      /* 鼠标活动窗口右下角列坐标 */
int bottom;                     /* 鼠标活动窗口右下角行坐标 */
{
    if(_HaveMouse)
    {
        /* ----- 设置鼠标列坐标的最小值和最大值 ----- */
        _CX    = left;
        _DX    = right;
        _AX    = 7;
        geninterrupt(0x33);

        /* ----- 设置鼠标行坐标的最小值和最大值 ----- */
        top    += _ScreenTop;
        bottom += _ScreenTop;
        _CX    = top;
        _DX    = bottom;
        _AX    = 8;
        geninterrupt(0x33);
    }
}

```

#### 程序 2-14 等待鼠标按钮释放。

为了设计实用的鼠标控制的按键式菜单和鼠标按钮的释放和拖动等功能，我们设计了一个等待置鼠标按钮释放函数。该函数可以在某鼠标按钮按下后等待到该鼠标按钮释放。

```

/* -----
   函数 till_mouse_pop : 等待置鼠标按钮释放
   ----- */

#include <dos.h>

extern int _HaveMouse;          /* 鼠标装载成功标记      */

void till_mouse_pop(which_button)
int which_button;               /* 等待释放的鼠标按钮    */
{
    . . .

```

```
if(_HaveMouse)
{
    which_button == 400;
    do
    {
        _BX = which_button;
        _AX = 6;
        geninterrupt(0x33);
    }while(_AX == which_button);
}
```

## 2.4 HANENV 系统中关于鼠标应用的其他设置

本节介绍的鼠标应用程序族中,我们定义了一组全局变量用以标志和控制鼠标的运行。应用程序设计人员应该避免直接使用这些全局变量,特别是直接向这些全局变量赋值,以免破坏鼠标的工作状态。因此,我们在 HANENV 系统的头文件 `hanenv.h` 中定义了一组宏调用来操纵这些鼠标全局变量。在应用程序中,我们可以将这些宏调用当做库函数使用。其使用方法可参看本书的第 15 章“HANENV 系统的库函数”,定义可参看第 14 章“HANENV 系统的头文件”中的有关部分。

## 键盘操作

键盘是微型计算机最常用的输入设备。在 C 语言中对键盘的编程可以通过四条途径进行：C 语言库函数、DOS 系统功能调用、BIOS 中断和直接对键盘缓冲区操作。下面我们结合 HANENV 系统有关部分的设计分别介绍这四方面的内容。

### 3.1 Turbo C 的键盘操作库函数

1. getch, getchar, getche, getpass, gets... : 用于从标准输入流中读取数据。这些函数都是 C 语言中最常用的函数, 一般读者都很熟悉。这些函数的共同特点是读取数据的基本单位是字符, 因此不能直接用于输入汉字。在读取扩充键盘码时也要经过一定处理方可。

2. ungetch : 将一个字符退回键盘缓冲区。和上一组函数一样, 该函数的缺点也是无法有效地处理汉字和扩充键盘码。

3. kbhit : 检查当前是否有键按下。该函数的用法为

```
int kbhit(void);
```

当有键按下时该函数返回非零值, 否则返回零。

4. bioskey : BIOS 的键盘接口函数, 实际上是直接调用 BIOS 中断 INT16H 的功能。其用法为

```
int bioskey(int cmd);
```

参数 cmd 确定该函数的实际操作:

cmd = 0: 返回从键盘输入的下一键码。如果返回值的低 8 位非 0, 即为 ASCII 字符, 否则其高 8 位为扩充键盘码。

cmd = 1: 测试输入键是否有效。此时用法和 kbhit 函数相同。

cmd = 2: 查询当前各换档键的状态。此时该函数返回值各位的含义为

第 7 位 : ins 键是否被触发

第 6 位 : caps 键是否被触发

第 5 位 : Num Lock 键是否被触发

第 4 位 : Scroll Lock 键是否被触发

第3位：Alt 键是否被按下

第2位：Ctrl 键是否被按下

第1位：左 Shift 键是否被按下

第0位：右 Shift 键是否被按下

如果某对应换档键被触发或按下则返回值的对应位为1,否则为0。

## 3.2 DOS 系统功能调用

使用DOS系统功能调用INT21H的有关子功能也可以实现由键盘的输入。和使用BIOS中断相比,通常将使用DOS的系统功能调用称为高级调用。

### 3.2.1 从键盘输入一个字符并回显(01H)

在Turbo C中的调用方法为

```
int key;
```

```
key = bdos(1,0,0);
```

```
if(key == 0)
```

```
    key = bdos(1,0,0)+128;
```

即如果返回值为0,说明按下的是一个组合键,还需要再次调用本功能以读取其扫描码。

### 3.2.2 从键盘输入一个字符但不回显(08H)

该子功能可以用来设计口令输入功能。调用方法为

```
int key;
```

```
key = bdos(8,0,0);
```

### 3.2.3 从键盘输入一个字符串(0AH)

该子功能可以将从键盘输入的一个字符串送入应用程序指定的缓冲区中。该缓冲区的结构应符合如下条件:

1. 第1个字节指定缓冲区的容量;
2. 第2个字节供DOS返回实际键入的字符数(不包括结束字符串用的回车键);
3. 从第3个字节开始存放输入的字符串。

在Turbo C中调用该子功能可以使用如下程序段:

```
char keybuffer[128];
```

```
keybuffer[0] = 128;
```

```
bdosptr(0x0a,&keybuffer,0);
```

### 3.2.4 将键盘看作文件对其进行读操作(3FH)

我们可以将键盘看成一个文件,DOS引导后该文件即自动打开,并被赋予文件号0。因

此可以使用 INT21H 的 3FH 号子功能对该文件进行读操作。

程序 3-1 读键盘文件。

```
/* -----
   函数 read_key : 读键盘。
   ----- */

#include <dos.h>

int read_key(char *keybuffer,int size)
{
    union REGS r;
    struct SREGS s;

    r.h.ah = 0x3f;          /* 读文件          */
    r.x.bx = 0;             /* 键盘文件号      */
    r.x.cx = size;          /* 指定缓冲区大小  */
    r.x.dx = FP_OFF(keybuffer); /* 键盘缓冲区偏移量 */
    s.ds = FP_SEG(keybuffer); /* 键盘缓冲区段地址 */
    intdosx(&r,&r,&s);       /* 读文件          */
    return r.x.ax;          /* 返回读的字符个数 */
}
```

注意此时已经没有字符还是字符串的区别了。如果键入的内容超过了所提供的缓冲区的大小,则多出的部分自动被截去。

### 3.3 使用 BIOS 的键盘中断编程

PC 系列微型计算机的 BIOS 提供了两个键盘中断:INT09H 和 INT16H。INT09H 是硬中断,每当发生击键操作时,无论是键被按下还是放开,INT09H 都立即被调用。根据所按键的扫描码的不同,该中断分别进行三种工作:

- (1)如果是换档键(参看 3.1)则将其状态存入内存中的 0040:0017H 和 0040:0018H 单元处;
- (2)如果是普通字符键或组合键,将击键转换为 ASCII 码或扩充键盘码,然后写入键盘缓冲区;
- (3)如果是特殊组合键如 Ctrl+C(Ctrl+Break)、Ctrl+Alt+Del 和 Shift+Prtsc 等则直接执行相应的动作。

值得注意的是中断 09H 对组合键 Ctrl+C(或 Ctrl+Break)的处理。通常该组合键用于强行中断应用程序的执行而返回 DOS。在许多情况下这是必要的,使得用户可以在程序运行中发生严重错误时可以迅速中断程序的运行。但是有些应用程序为了实现一些特殊功能,在运行时修改了一些操作系统提供的资源如中断程序等,在程序正常运行结束前再将其恢



复原来的状态。显然,在这些程序运行时使用组合键 Ctrl\_C (Ctrl+Break) 强行中断时,这些被修改的内容得不到恢复,就有可能引起死机等现象。因此,许多比较考究的大型应用程序都采用修改中断 09H 的方法来禁止使用上述组合键。

在我们的 HANENV 系统中,为了显示汉字、实现屏幕平滑滚动和闪烁光标等功能,重新设置了显示模式,修改了鼠标中断 33H 和时钟中断 1CH,因此,使用 HANENV 系统编写的应用程序也必须对组合键 Ctrl+C (Ctrl+Break) 进行屏蔽。我们的方法是首先将处理强行结束程序运行的中断 23H 的处理程序替换为我们自己编写的处理程序(实际上是一个空的中断函数),然后再修改中断 09H 以取消该组合键的屏幕显示。具体的做法请参看 5.1 节“HANENV 系统的初始化”中的有关内容。

进入键盘缓冲区的扫描码可以通过中断 INT16H 进行处理。INT16H 是软中断,共有 4 个功能调用:

1. 读键盘缓冲区(AH=00H)。从键盘读取字符。如果读键盘缓冲区中有字符可供读取则将其读入 AX 寄存器,然后修改读键盘缓冲区指针,否则等待按键。

2. 读键盘缓冲区状态(AH=01H)。检查键盘缓冲区。如果键盘缓冲区中没有键值可供取出,则 AX=0,反之则将键码取到 AX 中,但不清除键盘缓冲区(即以后仍可读该键值)。无论键盘缓冲区有没有键码都不等待。

3. 读换档键的状态(AH=02H)。该功能将内存中的 0040:0017H 和 0040:0018H 处的内容读至 AX 中。这两个字节存放着换档键的当前状态,表 3-1 列出了其各位为 1 时的含义。

表 3-1 换档键的当前状态

位	单元 0040:0017H	单元 0040:0018H
7	Ins 键已处于插入状态	Ins 键已持续按下
6	Caps Lock 键已上锁	Caps Lock 键已持续按下
5	Num Lock 键已上锁	Num Lock 键已持续按下
4	Scroll Lock 键已上锁	Scroll Lock 键已持续按下
3	Alt 键已按下	Ctrl+Num Lock 键已持续按下
2	Ctrl 键已按下	
1	右 Shift 键已按下	
0	左 Shift 键已按下	

4. 读取/设置键盘速度(AH=03H)。该功能用于读取或设置键盘的响应速度和延迟时间。在调用之前应设置各寄存器的值:

AH=03H

AL=05H(设置键盘响应速度与延迟时间)/06H(读键盘响应速度与延迟时间)

BH=延迟时间(仅当 AL=05H)

BL=键盘响应速度(仅当 AL=05H)

如果是读键盘响应速度与延迟时间,则在调用中断后 BX 中有数据:

BH=延迟时间

BL=键盘响应速度

在 HANENV 系统中,我们设计了一个新的多用途键盘输入函数。该函数的功能有以下几个方面:

1. 接收 BIOS 键盘中断 16H 来的按键信息,并将其转换为 HANENV 系统的统一键盘编码。在 HANENV 系统中,我们将键盘键入的 ASCII 码、扩展 ASCII 码、组合键和功能键的键盘扫描码、鼠标按钮状态以及汉字机内码综合起来,编成一套统一键盘码,用两个字节 (unsigned 类型的变量) 存储。这样可以大大方便应用程序的编程工作。除了西文 ASCII 码和汉字机内码保持原样不变以外,其他编码的码值分布在 256—1024 之间,由头文件 hanenv.h 中的一组宏定义。

2. 接收鼠标按钮信息,并将其转换为 HANENV 系统的统一键盘编码。

3. 接收由鼠标控制的屏幕虚拟小键盘来的的按键信息,并将其转换为 HANENV 系统的统一键盘编码。我们设计了一个使用鼠标控制的弹出式虚拟键盘,使用这个虚拟键盘可以使应用程序的使用人员只使用鼠标就可以完成大部分键盘输入工作(见图 3-1)。

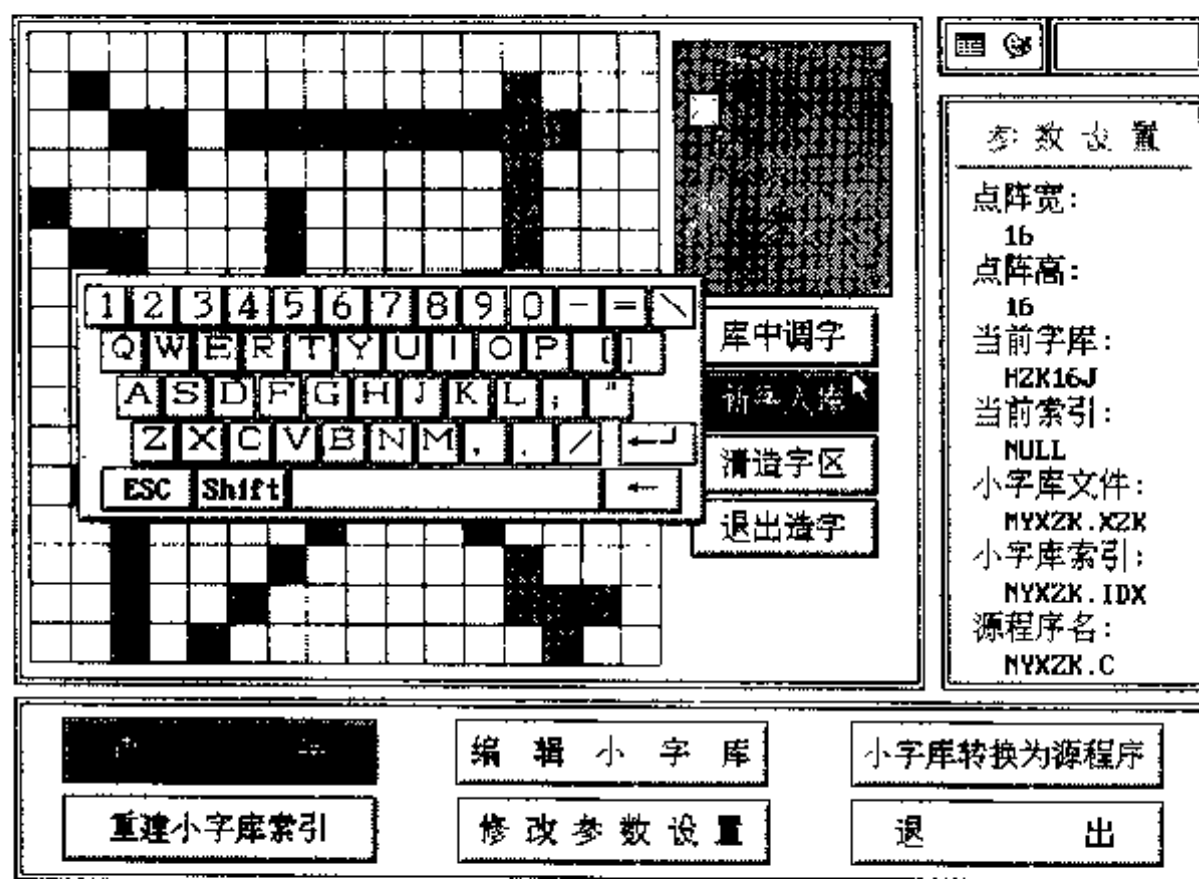


图 3-1 使用鼠标控制的虚拟键盘

4. 根据由以上几种渠道接收的键盘编码驱动事件触发器。在 HANENV 系统中我们提供了事件触发器功能,允许应用程序的设计人员将某些功能模块设计为触发器控制方式,其触发条件可以是热键,也可以是鼠标事件(在屏幕上某区域按下或释放指定的鼠标按钮)。在头文件 hanenv.h 中有触发器变量类型的定义和变量说明:

```
/* -----
   触发器的数据类型和全局变量的定义
   ----- */
typedef struct                                     /* 触发器类型的定义 */
```

```

{
    unsigned press_key;          /* 按键触发事件 */
    unsigned mouse_affair;       /* 鼠标触发事件 */
    int left;                    /* 鼠标左边界 */
    int top;                    /* 鼠标上边界 */
    int right;                   /* 鼠标右边界 */
    int bottom;                  /* 鼠标下边界 */
    unsigned (* trigger)(unsigned h); /* 事件函数 */
} TRIGGER;

extern int _TriggerCount;        /* 触发器个数 */
extern TRIGGER _Triggers[];      /* 触发器数组变量 */

```

在应用程序中定义触发器只需填写相应的触发器变量并将触发器计数变量的值加1。在 HANENV 中最多可以定义 20 个触发器事件,其中第 0 个触发器系为定时器(闹钟)准备,不应随意占用。

### 程序 3-2 多功能键盘信息函数。

该程序被设计为不断地测试键盘和鼠标,直到有键按下或有鼠标按钮按下时再进行处理。

```

/* -----
   函数 geth : 接收键盘信息
   ----- */
#include (hanenv.h)

int _TriggerCount = 0;          /* 触发器个数 */
TRIGGER _Triggers[20];         /* 触发器向量 */
unsigned (* mouseKB)(unsigned h) = NULL; /* 虚拟键盘函数指针 */

unsigned _Cdecl geth(void)
{
    unsigned h = 0;
    int i;

    /* ----- 循环测试键盘和鼠标 ----- */
    do
    {
        /* ----- 如果有键按下 ----- */
        if(kbhit())
        {
            unsigned char al,ah,st;

```

```

/* ----- 从键盘缓冲区取一键码 ----- */
_AH = 0;
geninterrupt(0x16);
al = _AL;
ah = _AH;

/* -- 查询 Alt, Ctrl 和左右 Shift 键的状态 ----- */
st = peekb(0x40, 0x17);

/* -- 如果是左 shift+箭头, 生成相应代码 ----- */
if(ah >= 0x47 && ah <= 0x51 && st & 0x01)
    h = (unsigned)ah | 0x180;

/* -- 如果是右 shift+箭头, 生成相应代码 ----- */
else if(ah >= 0x47 && ah <= 0x51 && st & 0x02)
    h = (unsigned)ah | 0x200;

/* -- 如果是 Ctrl+a-z, 生成相应代码 ----- */
else if(al >= 0x01 && al <= 0x1a && st & 0x04)
    h = (unsigned)al | 0x280;

/* ----- 如果不是组合键或功能键的扫描码, 继续判断 ----- */
else if(al)
{
    /* -- 如果是汉字, 继续取后半部汉字 ----- */
    if(al & 0x80)
    {
        ah = al;
        _AH = 0;
        geninterrupt(0x16);
        al = _AL;
        h = (unsigned)(al * 256 + ah);
    }

    /* ----- 否则就是 ASCII 码 ----- */
    else
        h = (unsigned)al;
}

/* -- 生成组合键或功能键的统一键盘扫描码 ----- */
else
    h = (unsigned)ah | 0x100;
}

```

```

/* -- 无键按下则查看是否有鼠标按钮按下 -- -- -- */
else
{
    h = get_mouse_status(&i,&i);

    /* -- 如果虚拟键盘已安装,由虚拟键盘处理 -- -- -- */
    if(mouseKB && h==LEFT_BUTTON)
        h = (*mouseKB)(h);
}
}while(h==0);

/* -- -- -- 检查并激活事件触发器 -- -- -- */
for(i=0;i<_TriggerCount;i++)
    if(h==_Triggers[i].press_key || h==_Triggers[i].mouse_affair
        && mouse_enter(_Triggers[i].left,_Triggers[i].top,
            _Triggers[i].right,_Triggers[i].bottom))
        return (*(_Triggers[i].trigger))(h);

/* -- -- -- 返回 HANENV 系统的统一键盘码 -- -- -- */
return h;
}

```

下面我们给出一个例子说明 geth 函数的应用和触发器函数的设计方法。

### 程序 3-3 程序设计示例: geth 函数的应用和触发器函数的设计方法。

下面的程序将由键盘输入的字符显示在屏幕下方的提示行上,如果输入字符不是 ASCII 码(例如鼠标按钮),则由扬声器发出一声“嘟”。使用鼠标左键点一下提示行最右端的“键盘”字样,就可以调出虚拟键盘。使用虚拟键盘和鼠标可以模拟由键盘上输入的过程。另外,我们将计算器设计为一个触发器,可以使用热键“F2”或用鼠标左键点击提示行中的“CALC”区激活。程序中的下列函数均为 HANENV 系统的库函数:

```

double    calculor(int x,int y);
void      _Block(int col,int line,int width,int high,int color);
void      outxys(int col,int line,int color,char * string);
void      delight_mouse(void);
void      set_mouse_KB(int x,int y,int color,int bk,int left,int top,int right,
                    int bottom);
void      enableMKB(void);
void      light_mouse(void);
unsigned  geth(void);
void      outxyc(int col,int line,int color,int ch);
void      init_hanenv(void);

```

```
void close_hanenv(void);
```

这些函数的使用说明均可在第15章“HANENV 系统的库函数”中找到。另外,“Beep”是定义于头文件 hanenv.h 中的宏,其值为 07H,即 ASCII 码表中的蜂鸣器符号。

```
/* -----
   测试程序：测试函数 geth 的功能。
   ----- */

#include <stdio.h>
#include <dos.h>
#include <hanenv.h>

unsigned cal(unsigned h)          /* 触发器函数：调用计算器 */
{
    char tmpline[81];
    double result = calcltor(20,260); /* 调用计算器函数 */
    sprintf(tmpline,"Result is %f",result);
    _Block(0,457,30,CHAR_HIGH,LIGHTGRAY);
    outxys(0,457,BLACK,tmpline); /* 显示计算结果 */
    return h;
}

main()
{
    unsigned k;                  /* 用于接收键盘输入的变量 */

    /* -- 初始化 HANENV 系统 ----- */
    init_hanenv();

    /* -- 设置一个触发器 ----- */
    _Triggers[_TriggerCount].press_key = KEY_F2;
    _Triggers[_TriggerCount].mouse_affair = LEFT_BUTTON;
    _Triggers[_TriggerCount].left = 69 * 8;
    _Triggers[_TriggerCount].top = 457;
    _Triggers[_TriggerCount].right = 73 * 8;
    _Triggers[_TriggerCount].bottom = 457 + 18;
    _Triggers[_TriggerCount].trigger = cal;
    _TriggerCount++;

    /* -- 绘制屏幕版面 ----- */
    delight_mouse();
    _Block(0,0,80,480,GREEN);
```

```

    _Block(0,450,80,30,LIGHTGRAY);
    _Block(69,457,4,CHAR_HIGH,WHITE);
    _Block(74,457,4,CHAR_HIGH,WHITE);
    outxys(75,457,BLACK,"KB");
    outxys(69,457,BLACK,"CALC");
    outxys(40,457,BLACK,"Press key is");

    /* -- 设置鼠标虚拟键盘 ----- */
    set_mouse_KB(312,320,BLACK,LIGHTGRAY,0,0,639,449);
    enableMKB();

    /* ---- 循环接收键盘和鼠标输入 ----- */
    light_mouse();
    while((k=geth())!=KEY_ESC)
    {
        if(k<256)                /* ASCII 码显示 */
        {
            _Block(53,457,1,CHAR_HIGH,LIGHTGRAY);
            outxyc(53,457,BLACK,k);
        }
        else                      /* 其他统一键盘码蜂鸣器输出 */
            putch(Beep);
    }

    /* -- 退出 HANENV 系统 ----- */
    close_hanenv();
}

```

注意该程序编译时要联接 HANENV 系统的库文件,请参看 5.2 的有关内容。

#### 程序 3-4 判断是否双击鼠标按钮。

熟悉 WINDOWS 的读者一定知道在 WINDOWS 中鼠标有一种特别的用法,即“双击”,使用鼠标左键“双击”一个图标和“点选”一个图标的效果是不同的。在 HANENV 系统中为了实现这一功能,提供了一个测试函数,用于在使用 geth 函数接收到鼠标按键信息后继续判断是否“双击”。该函数的实现算法比较简单,即在规定的時間间隔(省缺值为 0.4 秒)内检查是否有第二次击键动作。如果有则返回非零值,否则返回零。该函数的源程序为

```

/* -----
   函数 double _press : 测试是否双击鼠标按钮
   ----- */
#include <hanenv.h>

```



```

    _DoublePressTime = 400;
int _Cdecl double _press(unsigned key)
{
    int isdoublepress = NO;
    /* -- 如果是鼠标按键,检查是否双击鼠标键 ----- */
    if(key > 400)
    {

        /* -- 清除鼠标按键计数 ----- */
        _BX    = 0;
        _AX    = 5;
        geninterrupt(0x33);

        /* -- 延迟规定的时间 ----- */
        delay(_DoublePressTime);

        /* -- 测试相应鼠标按键按下次数 ----- */
        _BX    = key - 401;
        _AX    = 5;
        geninterrupt(0x33);
        if(_BX == 1)
            isdoublepress = YES;
    }
    return isdoublepress;
}

```

在 HANENV 系统的库函数 `_GetCode` 中使用了该函数,可参看 12.4。

### 3.4 键盘缓冲区

键盘缓冲区在 BIOS 数据区中,地址为 0040:001EH,共 32 个字节。键盘缓冲区的单位为字(两个字节),因此在键盘缓冲区中最多可以存放 16 个键码。如果键码是普通的 ASCII 码,则高位字节值为 0,低位字节值即为 ASCII 码;如果键码是扩充键盘码,则低位字节值为 0,高位字节值为扫描码。键盘缓冲区被设计成一个环形队列,共使用 4 个 BIOS 变量指向键盘缓冲区,这 4 个变量存放的都是相对于 BIOS 数据区段地址 0040 的偏移量。这四个变量是:

键盘缓冲区首址(0040:0080H):存放键盘缓冲区起始单元的偏移量,一般为 001EH;

键盘缓冲区末址(0040:0082H):存放键盘缓冲区结束单元的偏移量,一般为 003EH;

键盘缓冲区头指针(0040:001AH):存放键盘缓冲区队列的首单元的偏移量,也就是指向下一个从缓冲区读出的字符;

键盘缓冲区尾指针(0040:001CH)存放键盘缓冲区队列的尾单元的偏移量,也就是指

向下一个存入缓冲区的字符；

BIOS 关于键盘缓冲区的工作方式如下：

计算机加电之后键盘缓冲区的头、尾指针均被初始化为指向键盘缓冲区的起始单元 001EH。一旦有键按下，中断 INT09H 即将从 8255 PA 口得到的键盘扫描码进行变换后存入键盘缓冲区尾指针指向的一个字中，然后将该尾指针的值加 2。当该值超过键盘缓冲区的结束单元的偏移量(003EH)时再绕回键盘缓冲区起始单元处(001EH)。如果尾指针已和头指针相等则说明键盘缓冲区已满。键盘缓冲区中的键码是通过调用中断 INT 16H 的 0 号功能读出的。当执行此功能时从头指针指向的地址读一个字到 AX 寄存器中(参看 3.3)，并使头指针的值加 2，当该值超过键盘缓冲区的结束单元的偏移量(003EH)时也再绕回键盘缓冲区起始单元处(001EH)，而如果尾指针已和头指针相等则说明键盘缓冲区已空。

其实，在应用程序中也可以直接对键盘缓冲区进行操作以实现某些特殊功能。此时要注意调整好上述键盘缓冲区的队列指针。

### 程序 3-5 将 HANENV 系统的统一键盘码送入键盘缓冲区。

在 HANENV 中我们定义了一种统一键盘码，其基本思想是应用同一种数据类型存放所有可能可能遇到的键值：ASCII 码、扩展 ASCII 码、扩展键盘码、鼠标按钮状态以及汉字。综合各种情况，我们选用了 C 语言的无符号整型变量存放统一键盘码，即统一键盘码一律占用两个字节表示。这样对于程序的处理来说是相当方便的。关于统一键盘码的详细情况可参看 3.3。

```
/* -----
   函数 ungeth：将 HAN 码送入键盘缓冲区。
   ----- */
#include <hanenv.h>

void ungeth(h)
unsigned h;                /* 欲送入键盘缓冲区的统一键盘码 */
{
    int offset;            /* 键盘缓冲区队列尾指针 */
    int front;             /* 键盘缓冲区头指针 */
    int rear;              /* 键盘缓冲区尾指针 */

    disable();
    front = peekb(0x40,0x80); /* 取键盘缓冲区的头指针 */
    rear = peekb(0x40,0x82); /* 取键盘缓冲区的尾指针 */
    rear -= 2;              /* 修正键盘缓冲区尾指针 */
    offset = peekb(0x40,0x1c); /* 取键盘缓冲区队列尾指针 */
    if(h > 0xa0a0)          /* 是汉字 */
    {
        poke(0x40,offset,h&0xff); /* 作为两个扩展 ASCII 码存入 */
    }
}
```

```

    offset = (offset == rear? offset = front; offset + 2);
    poke(0x40, offset, h >> 8);
}
else if(h > 0xff) /* 是控制码 */
{
    if(h > KEY_Ctr_A) /* 处理 Ctrl+字母 */
    {
        h -= 0x280;
        pokeb(0x40, 0x17, 0x04);
        poke(0x40, offset, h); /* 作为普通键盘码存入 */
    }
    else if(h > KEY_RS_Left) /* 处理右 Shift+扩展键盘码 */
    {
        h -= 0x200;
        pokeb(0x40, 0x17, 0x02);
        poke(0x40, offset, h << 8); /* 作为扩展键盘码存入 */
    }
    else if(h > KEY_LS_Left) /* 处理左 Shift+扩展键盘码 */
    {
        h -= 0x180;
        pokeb(0x40, 0x17, 0x01);
        poke(0x40, offset, h << 8); /* 作为扩展键盘码存入 */
    }
    else
        poke(0x40, offset, h << 8); /* 作为扩展键盘码存入 */
}
else if(h == '\n') /* 是回车键 */
    poke(0x40, offset, 0x0d1c); /* 存入回车换行 */
else /* 是普通 ASCII 码 */
    poke(0x40, offset, h); /* 直接存入键盘缓冲区 */
if(offset == rear) /* 已到键盘缓冲区的结束单元 */
    pokeb(0x40, 0x1c, front); /* 调整键盘缓冲区的尾指针 */
else
    pokeb(0x40, 0x1c, offset + 2); /* 调整键盘缓冲区的尾指针 */
enable();
}

```

### 程序 3-6 将一个字符串送入键盘缓冲区。

本函数可以将一个字符串送入键盘缓冲区。当然，该字符串的长度应小于键盘缓冲区的实际长度，即 16 个字符。送入键盘缓冲区的字符串中既可以包括西文字符，也可以包括汉字。送入键盘缓冲区的字符串可以再由 geth 函数逐个读出。

```

/* -----
   函数 ungetstr : 将字符串送入键盘缓冲区。
   ----- */

#include <dos.h>

void ungetstr(string)
char * string;          /* 欲送入键盘缓冲区的统一键盘码 */
{
    int offset;          /* 键盘缓冲区队列尾指针 */
    int front;           /* 键盘缓冲区头指针 */
    int rear;            /* 键盘缓冲区尾指针 */

    disable();
    front = peekb(0x40,0x80); /* 取键盘缓冲区的尾指针 */
    rear = peekb(0x40,0x82); /* 取键盘缓冲区的尾指针 */
    offset = peekb(0x40,0x1c); /* 取键盘缓冲区队列尾指针 */
    while( * string)
    {
        if( * string == '\n') /* 是回车键 */
            pokeb(0x40,offset,0x0d1c); /* 存入回车换行 */
        else /* 是普通 ASCII 码 */
        {
            pokeb(0x40,offset, * string); /* 直接存入键盘缓冲区 */
            pokeb(0x40,offset+1,0);
        }
        offset += 2;
        string++;
        if(offset == rear) /* 已到键盘缓冲区的结束单元 */
            offset = front; /* 返回键盘缓冲区的起始单元 */
    }
    pokeb(0x40,0x1c,offset); /* 调整键盘缓冲区的尾指针 */
    enable();
}

```

在 HANENV 中设计这两个函数的目的在于使用键盘缓冲区传递数据。使用这两个函数送进键盘缓冲区的数据可以使用中断 INT16H 读出,相当于在键盘上按下了相应的键,在有些情况下使用这两个函数传递数据具有概念清楚、编程简单的特点。这方面应用的例子可以参看 8.3 和 8.4 节。

# 扩充存储器编程

### 4.1 扩充存储器与扩充存储器调用规范 XMS

内存储器是计算机最重要的系统资源之一,计算机拥有的内存储器的大小直接影响着计算机的性能。最初的 PC 机采用了 8086/8088 芯片作为中央处理器,因而确定了 PC 系列微型计算机的主要逻辑框架。8086/8088 是为单任务、单用户系统设计的 16 位处理器。由于它只有 20 位地址总线,因此只能管理  $2^{20}=1\text{M}$  字节的地址空间。对于日益发展的计算机软件来说,1M 字节的内存支持实在是太小了。因此,之后出现的 80286 的地址线数便扩展到了 24 根,它的地址空间变为  $2^{24}=16\text{M}$  字节;再后出现的 80386 和 80486,地址线数更增加为 32 根,所以地址空间可达  $2^{32}=4\text{G}$  字节!

既然 80386 和 80486 的地址空间已达 4G 字节,那么是不是应用程序也就可以使用如此巨大的内存资源了呢? 问题还不是这么简单。

首先,应用程序可用内存资源受到微机内存储器物理配置的限制。由于存储器芯片已经成为微型计算机中最昂贵的部件,所以大多数微型计算机的内存储器只配有数量有限的存储器芯片。例如,尽管 80386/80486 的地址空间可达 4G 字节,但大多数 386/486 微机只装有 4—32M 字节的存储器芯片。

其次,限制应用程序使用内存资源的还有操作系统。要知道,8086 的 CPU 对内存的访问是通过一种被称为实地址模式的内部操作方式(见 4.2)实现的。在实地址模式下,8086/8088 的 20 根地址线均被激活控制,所以内存处理量能达到 1M 字节。而 80286、80386 和 80486 等的 CPU 虽然也能将其所有的地址线激活控制,但必须通过一种被称作保护地址模式的内部操作方式才能实现。目前常用的 DOS 操作系统是在 8086/8088 处理器芯片上发展起来的,它对内存的管理方式沿用了 8086/8088 处理器的实地址模式。在设计 PC 机的内存管理方式时,IBM 的设计人员将实地址模式下 1M 字节内存地址空间中高端的 384K 保留给了视屏缓冲、适配器 ROM 以及 ROM BIOS 等其他硬件设备,仅将 640K 留给了 DOS 及其应用程序,这也就是我们常说的所谓常规内存(如图 4-1 所示)。

由于装有 8086/8088 处理器芯片和 DOS 操作系统的 PC 机巨大的商业成功,使得在保护模式出现之前 DOS 就已经成为微机操作系统的标准。在将 DOS 移至 80286、80386 以及

80486 上时,为了保证 PC 系列微型计算机上已经开发出来的大量软件的兼容性,DOS 的高级版本仍然沿用了这种实地址模式,使得只要是使用 DOS 操作系统的软件,无论是工作于装有 8086/8088 芯片的 PC 机,还是更先进的 386、486 甚至奔腾机,也无论是使用了 DOS 的哪个版本,都是在实地址模式下运行,其应用程序所能直接使用的内存地址空间都无法超过 640K 字节的常规内存。

实践证明 640K 常规内存很快就成为 PC 系列微型计算机发展的严重障碍。当然,最根本的解决方法是重新发展具有大地址空间的计算机硬件和软件,彻底抛弃实地址模式,例如具有 RISC 结构的各种

工作站。但是这种方法必需解决如何移植已经在 DOS 下开发出来的成千上万的应用软件的问题。显然这是一项极为困难的工作。因此,研究在 DOS 环境下突破 640K 常规内存的限制就成为刻不容缓的任务。

1984 年底,80286 刚刚推出时,640K 常规内存容量的缺陷就已经显露。这时 Lotus、Intel 和 Microsoft 公司联合推出了扩页存储器规范 EMS(Expand Memory Specification),定义了采用 8086 系列处理器芯片和运用 Microsoft DOS 操作系统的微型计算机兼容的硬件及软件子系统。EMS 依靠扩页存储器管理器程序 EMM(Expand Memory Manager)来支持,它提供了应用程序和扩页内存板之间的接口。在计算机中安装一块带有 RAM 的内存扩充板,再装入软件驱动程序 EMM,DOS 的应用程序即可通过页面交换方式访问 1M 字节以上的存储器。值得注意的是,即使是使用 8086/8088 芯片的 PC 机也可以通过使用 EMS 来扩充其内存储器。

从 DOS 5.0 和 WINDOWS 3.0 开始,Microsoft 公司又引进了另一种存储器管理规范——XMS(Extended Memory Specification,扩充存储器管理规范)。XMS 使用方便,安全可靠,所以 Microsoft 公司极力推荐使用 XMS。和 EMS 不同的是,XMS 直接使用 80286 以上芯片的保护地址模式功能使用扩充存储器,无需配用专门的扩页内存板。为了使应用 EMS 的应用程序也能在 XMS 下工作,DOS 的高版本还提供了 EMS 的仿真程序。应该注意的是,DOS 5.0 以上版本的 XMS 驱动程序 HIMEM.SYS 同样可以工作于 DOS 的较低版本如 DOS 3.3 之下,使其具有应用扩充存储器的能力。

除了以上两种途径,还可以通过 BIOS 的 INT 15H 中的有关子功能直接调用保护模式下的存储管理功能,从而使用扩充存储器。这种方法使用简便,无需专用的软、硬件支持,但由于应用程序直接对绝对地址操作,因此数据安全性较差。使用这种方法的一个典型范例

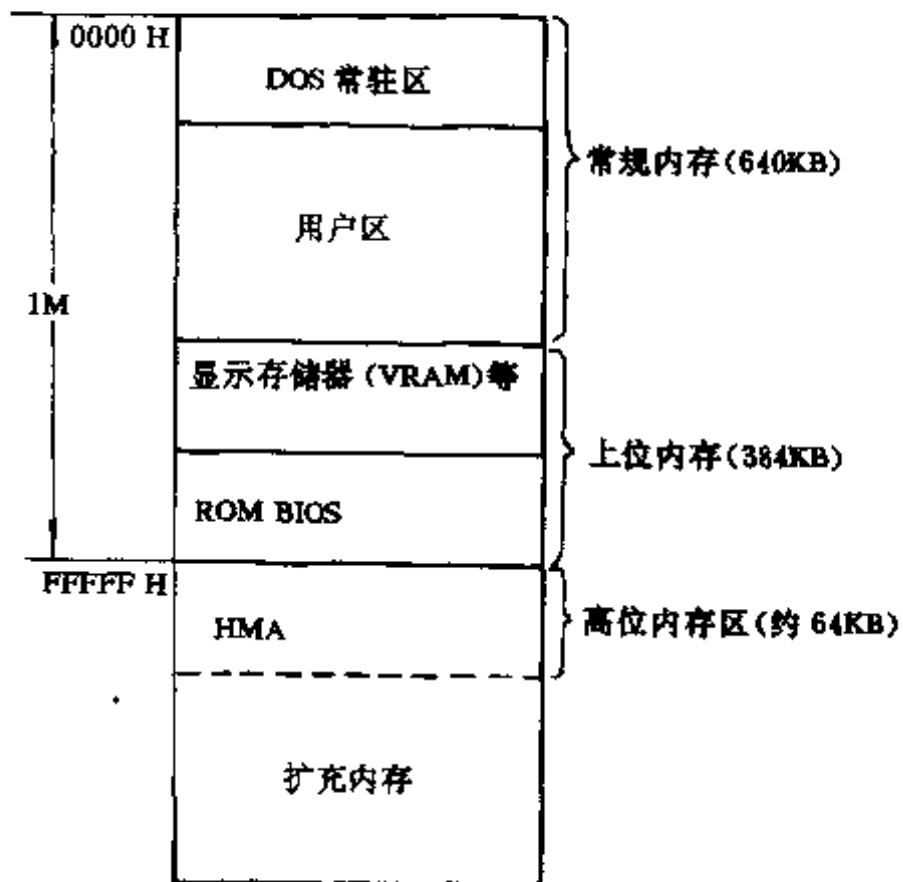


图 4-1 DOS 控制下的内存储器(实地址模式)

是西山汉字系统。

HANENV 系统提供了两组库函数,分别用于通过 BIOS 的 INT 15H 直接访问扩充存储器和通过扩充存储器管理规范 XMS 使用扩充存储器。下面我们分别介绍其设计思想。

## 4.2 直接访问扩充存储器

首先我们简单地介绍一下 80286 以上处理器芯片的实地址模式和保护模式。

在实地址模式下,地址总线为 20 位,CPU 所访问的任何存储器地址均由基地址加偏移量构成。基地址通常为段的起始地址,也称为段地址。段地址和偏移量的长度都是 16 位。由段地址和偏移量构成内存物理地址的规则为在 16 位的段地址后面补上 4 个 0 构成 20 位的段起始地址,然后再加上偏移量(如图 4-2 所示)。

在保护模式下的内存寻址方式与实地址方式下的寻址方式完全不同。保护模式下的内存寻址采用全局描述符表 GDT 索引方法。在全局描述符表中有 6 个地址描述符(Descriptor),分别描述保护模式下 CPU 寻址所需要的地址信息。每个描述符占用 8 个字节,其含义为

第 0~1 字节:读写缓冲区长度  
( $<64K$ )

第 2~3 字节:基地址低位字

第 4 字节:基地址高位字节

第 5 字节:存取权(初值应为 93H)

第 6~7 字节:保留字

地址描述符中的基地址低位字和基地址的高位字节一起正好构成一个 24 位的逻辑地址。

全局描述符表 GDT 中有 6 个地址描述符,其结构为

1. 空地址描述符;
2. 本 GDT 表的地址描述符;
3. 源数据块地址描述符;
4. 目标数据块地址描述符;
5. ROM BIOS 代码地址描述符;
6. ROM BIOS 堆栈地址描述符。

程序 4-1 地址描述符和全局描述符表 GDT 的类型定义。

在 HANENV 系统的头文件 hanenv.h 中,我们将地址描述符和全局描述符表 GDT 定义为结构类型变量:

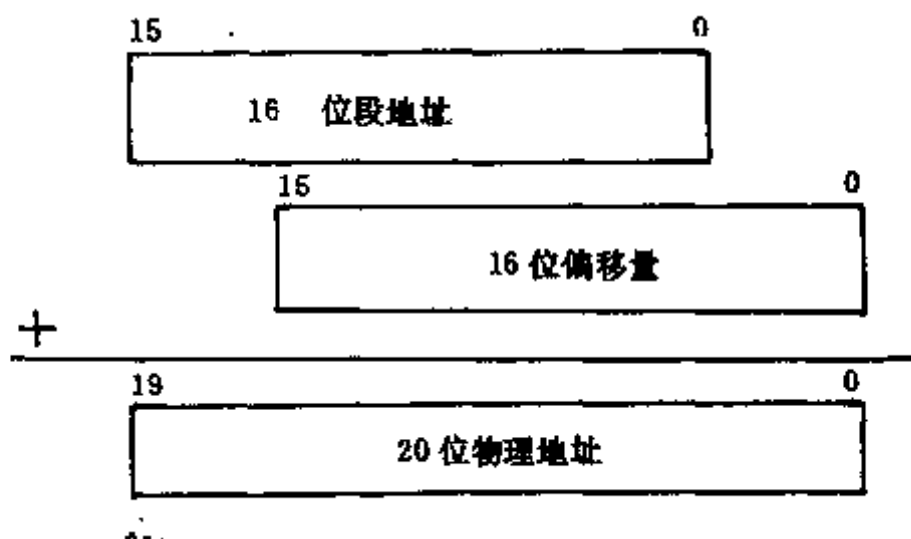


图 4-2 实地址模式的内存物理地址构成

```

/* -----
   保护地址模式下段的描述符类型定义
   ----- */
typedef struct
{
    unsigned size;          /* 读写缓冲区长度          */
    unsigned baselow;       /* 基地址低位字          */
    unsigned charbasehigh;  /* 基地址高位字节        */
    unsigned charattr;      /* 存取权                */
    unsigned blank;         /* 保留字                */
}Descriptor;               /* 描述符类型            */

/* -----
   全局描述符表 GDT 的类型定义
   ----- */
typedef struct
{
    Descriptor BlankDsc;    /* 空地址描述符          */
    Descriptor GDTDsc;      /* 本 GDT 表的地址描述符 */
    Descriptor Source;      /* 源数据块地址描述符    */
    Descriptor Destin;      /* 目标数据块地址描述符  */
    Descriptor BiosCS;      /* ROM BIOS 代码地址描述符 */
    Descriptor BiosSS;      /* ROM BIOS 堆栈地址描述符 */
}GDT;                      /* 全局描述符表 GDT 类型 */

/* -----
   用于实现保护模式下存取扩充内存的全局变量
   ----- */
extern GDT _MemCtrlBlock; /* 全局描述符表 GDT 的说明 */
extern long _CurrentEMM; /* 当前自由扩充存储器地址 */

```

PC/XT 机型上的 BIOS 中断 INT15H 是盒式磁带机中断,由于使用这种老式盒式磁带机作为微型计算机的外存储器的方法已被淘汰,所以 INT15H 中有关磁带机部分的功能几乎已经作废。从 AT 机开始,BIOS 扩充了 INT15H 的服务功能,其第 87H、88H 和 89H 号子功能处理面向保护模式的操作。由于 BIOS 和 DOS 并不直接支持保护模式,因此这里可以理解为 CPU 在 INT15H 中首先被置成保护模式,完成指定任务后再恢复为实地址模式。在 HANENV 系统中我们利用 INT15H 的 87H 和 88H 号子功能设计了读写扩充存储器的有关库函数。

#### 程序 4-2 初始化全局描述符表 GDT。

在首次使用 GDT 传输扩充存储器中的数据时应先初始化全局描述符表 GDT。



```

/* -----
   函数 _Init_GDT : 初始化全局描述符表 GDT
   ----- */
#include (hanenv.h)

GDT _MemCtrlBlock;          /* 全局描述符表 GDT 的定义 */
long _CurrentEMM = 1200000L; /* 当前自由扩充存储器地址 */

void _Cdecl _Init_GDT(void)
{
    /* ----- 初始化空地址描述符 ----- */
    _MemCtrlBlock.BlankDsc.size      = 0;
    _MemCtrlBlock.BlankDsc.baselow   = 0;
    _MemCtrlBlock.BlankDsc.basehigh  = 0;
    _MemCtrlBlock.BlankDsc.attr      = 0;
    _MemCtrlBlock.BlankDsc.blank     = 0;

    /* ----- 初始化本 GDT 表的地址描述符 ----- */
    _MemCtrlBlock.GDTDsc.size        = 0;
    _MemCtrlBlock.GDTDsc.baselow     = 0;
    _MemCtrlBlock.GDTDsc.basehigh    = 0;
    _MemCtrlBlock.GDTDsc.attr        = 0;
    _MemCtrlBlock.GDTDsc.blank       = 0;

    /* ----- 初始化源数据块地址描述符 ----- */
    _MemCtrlBlock.Source.size         = 0;
    _MemCtrlBlock.Source.baselow      = 0;
    _MemCtrlBlock.Source.basehigh     = 0;
    _MemCtrlBlock.Source.attr         = 0x093;
    _MemCtrlBlock.Source.blank        = 0;

    /* ----- 初始化目标数据块地址描述符 ----- */
    _MemCtrlBlock.Destin.size         = 0;
    _MemCtrlBlock.Destin.baselow      = 0;
    _MemCtrlBlock.Destin.basehigh     = 0;
    _MemCtrlBlock.Destin.attr         = 0x093;
    _MemCtrlBlock.Destin.blank        = 0;

    /* ----- 初始化 ROM BIOS 代码地址描述符 ----- */
    _MemCtrlBlock.BiosCS.size         = 0;
    _MemCtrlBlock.BiosCS.baselow      = 0;
    _MemCtrlBlock.BiosCS.basehigh     = 0;
    _MemCtrlBlock.BiosCS.attr         = 0;

```

```

_MemCtrlBlock.BiosCS.blank      = 0;

/* -- 初始化 ROM BIOS 堆栈地址描述符 -- -- -- */
_MemCtrlBlock.BiosSS.size       = 0;
_MemCtrlBlock.BiosSS.baselow    = 0;
_MemCtrlBlock.BiosSS.basehigh   = 0;
_MemCtrlBlock.BiosSS.attr       = 0;
_MemCtrlBlock.BiosSS.blank      = 0;
}

```

#### 程序 4-3 取扩充内存容量。

取扩充内存容量,也就是超过 1M 字节存储空间的地总址量,可以使用 INT15H 的 88H 号子功能完成。其调用方法为

入口寄存器设置:

AH = 88H

返回寄存器:

AX = 连续扩充存储器块数(每块 1K 字节)

```

/* -----
   函数 _GetEMMsize: 取扩充内存容量
   ----- */
#include <dos.h>

unsigned _Cdecl _GetEMMsize(void)
{
    _AH = 0x88;
    geninterrupt(0x15);
    return _AX;
}

```

#### 程序 4-4 保护模式下的数据传输。

INT15H 的 87H 号子功能可以实现保护模式下的数据传输,即在基本存储器(0—640K 字节)和扩充存储器(1M 字节—16M 字节)的范围内的数据传输,CPU 可以访问 16M 字节范围内的任何物理地址。其调用方法为

入口寄存器:

AH = 87H

CX = 以字为单位的传输量(≤8000H)

EX:SI = 指向全局描述符表 GDT

返回寄存器:

AX = 错误码(0=传输成功)

为了实现保护模式下常规内存和扩充存储器之间的数据传输,我们设计了3个库函数:  
\_SetSourAddr 用于设置 GDT 中的源数据地址描述符, \_SetDestAddr 用于设置目标数据地址描述符, \_MoveDataEMM 调用 INT15H 的 87H 号子功能进行数据传输。

```

/* -----
   函数 _SetSourAddr : 设置源数据地址
   ----- */
#include <hanenv.h>

void _Cdecl _SetSourAddr(long addr,unsigned size)
{
    _MemCtrlBlock.Source.baselow = addr & 0x0ffff;
    _MemCtrlBlock.Source.basehigh = addr >> 16;
    _MemCtrlBlock.Source.size = size;
}

/* -----
   函数 _SetDestAddr : 设置目标数据地址
   ----- */
#include <hanenv.h>

void _Cdecl _SetDestAddr(long addr,unsigned size)
{
    _MemCtrlBlock.Destin.baselow = addr & 0x0ffff;
    _MemCtrlBlock.Destin.basehigh = addr >> 16;
    _MemCtrlBlock.Destin.size = size;
}

/* -----
   函数 _MoveDataEMM : 保护模式下的数据传输
   ----- */
#include <hanenv.h>

extern void _SetSourAddr(long addr,unsigned size);
extern void _SetDestAddr(long addr,unsigned size);

void _MoveDataEMM(unsigned size)
{
    struct REGPACK reg;

```

```

    reg.r_es = FP_SEG(&_MemCtrlBlock);
    reg.r_si = FP_OFF(&_MemCtrlBlock);
    reg.r_cx = size >> 1;
    reg.r_ax = 0x8700;
    intr(0x15,&reg);
}

```

下面,我们介绍将数据文件直接装入扩充存储器的程序设计方法。首先要将文件分成小于 64K 的块读入内存,然后再利用 INT15H 的 87H 号子功能将其装入扩充存储器的指定位置。由于 INT15H 对于扩充存储器的地址并不进行统一管理,所以为了避免地址冲突,我们定义了一个全局变量 `_CurrentEMM`,每当将一个文件装入扩充存储器后即修改该变量的值使之指向自由扩充存储器的首地址。

#### 程序 4-5 利用 INT15H 将文件直接装入扩充存储器。

```

/* -----
   函数 _LoadEMM : 利用 INT15H 将文件直接装入扩充存储器
   ----- */
#include (hanenv.h)

long _Cdecl _LoadEMM(file,size)
FILE *file;                /* 欲装入扩充存储器的文件 */
long size;                  /* 文件长度 */
{
    long handle = _CurrentEMM;
    long addr;
    unsigned i,num;

    /* -- 如果剩余扩充存储器尺寸小于文件长度则返回 ----- */
    if(_GetEMMsize()-(int)((_CurrentEMM-1200000L)/1024L)<size/1024+1)
        return 0L;

    /* -- 设置各项参数 ----- */
    _CurrentEMM += size;
    size = size/1024+1;      /* 扩充存储器长度以 K 为单位 */
    _Init_GDT();
    addr = FP_SEG(_HanFont);
    addr = (addr<<4)+FP_OFF(_HanFont);
    _SetSourAddr(addr,1024);
    addr = handle;

    /* -- 将文件读到内存并装入扩充存储器 ----- */

```

```

for(i=0;i<size;i++)
{
    num=fread(_HanFont,sizeof(char),1024,file);
    _SetDestAddr(addr,num);
    _MoveDataEMM(num);
    addr += 1024;
}
/* -- 返回 EMM 自由空间的首地址 ----- */
return handle;
}

```

应该说明的是由于 INT15H 对于扩充存储器的使用并不进行统一的地址分配,完全由程序员决定读写的绝对地址,所以在使用时很容易引起地址冲突。尽管在本节中我们定义了一个全局变量 `_CurrentEMM` 用于标出自由扩充存储器的首地址,但仅作这样的处理在需要对扩充存储器进行频繁读写的场合是不充分的,此时最好改用下节介绍的扩充存储器管理规范 XMS 来访问扩充存储器。

### 4.3 利用扩充存储器管理规范 XMS 访问扩充存储器

扩充存储器管理规范 XMS 由 DOS 提供的 HIMEM.SYS 支持。XMS 不仅支持对于 1M 字节以上的扩充存储器的访问,而且支持对上位存储区 HMA(High Memory Area,在 CPU 处于实地址模式下时,可通过激活 A20 地址线而额外访问的 64K 内存)和上位存储块 UMB(Upper Memory Block,64K—1M 字节之间的内存区域)。因此 XMS 实际上共提供 5 类服务功能:驱动程序信息、HMA 管理、A20 地址线管理、UMB 管理和扩充存储器管理。在 HANENV 中我们只用到了 XMS 的扩充存储器管理功能。与分页式的扩页内存不同,XMS 提供的是一段线性空间,应用程序申请到的是以 K 为单位的一块存储区(而不是若干固定大小的页面),返回给用户程序的是一个句柄。用户可以把申请到的扩充内存块中的数据移动到常规内存中来,也可以把常规内存中的数据移至扩充内存块中去,从而实现对扩充内存的访问。

DOS 将其许多系统功能都定制成中断服务程序,如果要使用这些系统服务功能只要调用特定功能号的系统中断即可。而使用 XMS 的方法则有所不同:首先应调用中断 INT2FH 检查 XMS 驱动程序是否存在,如果存在则可取出 XMS 服务程序的入口地址;以后可以直接调用该入口地址应用 XMS 的各项功能。在 HANENV 系统中提供了利用 XMS 使用扩充存储器的库函数组。要特别注意的是在应用 XMS 之前应该保证在 DOS 根目录下的系统配置文件中有 XMS 的驱动程序,例如:

```
DEVICE=C:\DOS\HIMEM.SYS
```

**程序 4-6** 初始化 HANENV 系统的 XMS 服务程序。

检查 XMS 驱动程序是否已经安装和取 XMS 服务程序的入口地址均使用 BIOS 的

INT2FH 号中断。检查 XMS 驱动程序是否已经安装的调用方法为

入口寄存器:

AX = 4300H

返回寄存器:

AX = XMS 驱动程序安装标志(0=未安装,80H=HIMEM.SYS 安装成功)

在 XMS 驱动程序已经安装成功后,可以使用 INT2FH 的 4301H 号子功能取得 XMS 服务程序的入口地址。其调用方法为

入口寄存器:

AX = 4310H

返回寄存器:

BX = XMS 服务程序入口地址的偏移量

ES = XMS 服务程序入口地址的段地址

在取得 XMS 服务程序入口地址之后,即可通过设置寄存器 AX 的值来调用 XMS 的各种功能。例如,功能 08H 系检查扩充内存是否可用,其调用方法为

入口寄存器:

AX = 08H

返回寄存器:

AX = 以 KB 计的最大自由扩充内存的大小

DX = 以 KB 计的自由扩充内存的总和

BL = 错误代码

其中错误代码的值为:080H=功能未实现,081H=检测到了 VDISK 设备,0A0H=所有扩充内存都已经被分配。

```
/* -----
   函数 init_XMS : 初始化 XMS 服务程序
   ----- */
#include <hanenv.h>

void far (*_FunctionXMS)() = 0L; /* XMS 服务程序入口地址 */
int _LargestXMS           = 0; /* 最大自由扩充内存块尺寸 */
int _AmountOfXMS          = 0; /* 自由扩充内存总量 */

void _Cdecl init_XMS(void)
{
    /* ----- 检查 XMS 驱动程序是否已经安装 ----- */
    _AX = 0x4300;
    geninterrupt(0x2f);
    if(_AL == 0x80)
    {
        /* ----- 取得 XMS 服务程序的入口地址 ----- */
        _AX = 0x4310;
```

```

    geninterrupt(0x2f);
    _FunctionXMS = MK_FP(_ES, _BX);

    /* ----- 最大自由扩充内存的大小 ----- */
    _SizeofXMS();
}

```

#### 程序 4-7 求最大自由扩充内存块的大小和所有自由扩充内存块的总和。

求自由扩充内存的数量可以使用 XMS 的 08H 号功能,见程序 4-6 的说明。

```

/* -----
   程序 _SizeofXMS : 求自由扩充内存的总和
   ----- */
#include <hanenv.h>

void _Cdecl _SizeofXMS(void)
{
    if(_FunctionXMS)
    {
        _AH      = 8;
        (*_FunctionXMS)();      /* 调用 XMS 服务程序 */
        _LargestXMS = _AX;
        _AmountOfXM = _DX;
        _ErrorNo    = _BL;
    }
}

```

#### 程序 4-8 申请一块扩充内存。

申请一块扩充内存可以使用 XMS 的 09H 号功能,其调用方法为入口寄存器:

AH = 09H

DX = 以 KB 计的扩充内存块大小

返回寄存器:

AX = 状态(0=失败,1=成功)

DX = 所分配扩充内存块的句柄

BL = 错误代码

其中错误代码的值为:080H=功能未实现,081H=检测到了 VDISK 设备,0A0H=所有扩充内存都已经被分配,0A1H=所有扩充内存句柄均被占用。

```

/* -----

```

程序 \_GetXMS : 申请一块扩充内存

```
----- */
#include <hanenv.h>

int _Cdecl _GetXMS(int size)
{
    int handle = 0;
    if(_FunctionXMS)
    {
        _DX = size;          /* 扩充内存块大小,单位为 K 字节 */
        _AH = 0x09;
        (*_FunctionXMS)(); /* 调用 XMS 服务程序 */
        if(_AX)
        {
            handle = _DX;
            _SizeofXMS();
        }
        else
            _ErrorNo = _BL;
    }
    return handle;
}
```

#### 程序 4-9 释放扩充内存块。

释放扩充内存块可以使用 XMS 的 0AH 号功能,其调用方法为  
入口寄存器:

    AH     = 0AH  
    DX     = 欲释放的块句柄

返回寄存器:

    AX     = 状态(0=失败,1=成功)  
    BL     = 错误代码

其中错误代码的值为:080H=功能未实现,081H=检测到了 VDISK 设备,0A2H=所提供的句柄无效,0A2H=所提供的句柄加锁。

```
/* -----
    程序 _FreeXMS : 释放扩充内存块
    ----- */
#include <hanenv.h>

void _Cdecl _FreeXMS(int handle)
{
    -----
```



```

if(_FunctionXMS)
{
    /* -- 释放扩充内存块 -- */
    _DX = handle;
    _AH = 0x0a;
    (*_FunctionXMS)();
    if(_AX)
        _SizeofXMS();
    else
        _ErrorNo = _BL;
}
}

```

#### 程序 4-10 扩充内存和常规内存中的数据交换。

扩充内存与常规内存交换数据可以通过 XMS 的 0BH 号功能。其调用方法为入口寄存器：

    AH = 0BH

    DS:SI = 指向扩充内存块移动结构的指针

返回寄存器：

    AX = 状态(0=失败,1=成功)

    BL = 错误代码

其中错误代码的值为：080H=功能未实现,081H=检测到了 VDISK 设备,082H=A20 地址线出现错误,0A3H=无效源句柄,0A4H=无效源偏移量,0A5H=无效目的句柄,0A6H=无效目的偏移量,0A7H=无效长度,0A8H=移动有重叠,0A9H=奇偶校验错。

在移动数据时需要预先准备一个扩充内存移动结构的变量。在 HANENV 系统的头文件 hanenv.h 中有其定义如下：

```

/* -----
   扩充内存移动结构定义
   ----- */
struct EMB
{
    unsigned long len;           /* 需传输的数据字节数(32 位) */
    unsigned sour_han;          /* 源数据块句柄 */
    unsigned long sour_off;      /* 源偏移量(32 位) */
    unsigned dest_han;          /* 目标数据块句柄 */
    unsigned long dest_off;      /* 目标偏移量(32 位) */
};

```

在进行数据传输之前,必需首先填写这个扩充内存移动结构的变量。其中需传输的数据字节数必须为偶数。句柄号为 0 代表常规内存,此时偏移量的高 16 位存放段地址,低 16

位存放段内偏移量。如果传送的源数据块和目标数据块有重叠，只有在源地址小于目标地址时才能保证正确传输。

```

/* -----
   程序 _MoveDataXMS : 扩充内存和常规内存中的数据交换
   ----- */
#include (hanenv.h)

struct EMB _Emb;          /* 扩充内存移动结构变量 */

void _Cdecl _MoveDataXMS(void)
{
    if(_FunctionXMS)
    {
        _DS = FP_SEG(&_Emb);
        _SI = FP_OFF(&_Emb);
        _AH = 0x0b;
        (*_FunctionXMS)();
        if(_AX)
            _ErrorNo = 0;
        else
            _ErrorNo = _BL;
    }
}

```

程序 4-11 利用 XMS 将文件装入扩充存储器。

和使用 INT15H 时相同，也要将文件分成小于 64K 的块读入内存，然后再送入扩充存储器的指定位置。

```

/* -----
   程序 _LoadXMS : 利用 XMS 将文件装入扩充存储器
   ----- */
#include (hanenv.h)

int _Cdecl _LoadXMS(file, size)
FILE *file;          /* 欲装入扩充存储器的文件 */
long size;           /* 文件长度 */
{
    int i, handle;

    /* -- 如果剩余扩充存储器尺寸小于文件长度则返回 -- */
    size = size/1024+1; /* 扩充存储器长度以 K 为单位 */
}

```

```

if(_LargestXMS < size || (handle=_GetXMS(size))==0)
    return 0;

/* ----- 设置各项参数 ----- */
_LargestXMS -= size;
_Emb.sour_han = 0;
_Emb.sour_off = FP_SEG(_HanFont)
_Emb.sour_off <= 16;
_Emb.sour_off += FP_OFF(_HanFont);
_Emb.dest_han = handle;
_Emb.dest_off = 0L;

/* -- 将文件读到内存并装入扩充存储器 ----- */
for(i=0;i<size;i++)

    _Emb.len = fread(_HanFont,sizeof(char),1024,file);
    _MoveDataXMS();
    _Emb.dest_off += 1024
}
return handle
}

```

我们在第1章中已经介绍了将屏幕图象块保存在内存中的方法及相应的程序设计,但是一般来说,保存屏幕图形所需的存储量是很大的,整屏图象( $640 \times 480 \times 16$ )需占内存200K以上。这对于比较大的应用程序来说是个沉重的负担。而如果将屏幕图象存放到硬盘上速度又嫌太慢,影响屏幕显示效果。如果所使用的计算机上配有扩充内存,则可以将屏幕图象保存到扩充内存中去,这样既提高了速度,又不占用常规内存,是个一举两得的好方法。

#### 程序 4-12 将屏幕图象保存到扩充内存中去。

我们采用的方法是以汉字缓冲区\_HanFont为中间媒介,将屏幕图象块逐行由常规内存转储至扩充内存。

```

/* -----
    程序 getblockXMS : 将屏幕图象保存到扩充内存中去
----- */

#include <dos.h>
#include <stdio.h>
#include <hanenv.h>

extern int _ScreenWidth;      /* 逻辑屏幕宽度      */
extern int _ScreenTop;        /* 当前屏幕首行位置 */

```

```

int _Cdecl getblockXMS(left,top,right,bottom)
int left;          /* 屏幕图象块左上角列坐标(以字节为单位) */
int top;           /* 屏幕图象块左上角行坐标(以像素为单位) */
int right;         /* 屏幕图象块右下角列坐标(以字节为单位) */
int bottom;        /* 屏幕图象块右下角行坐标(以像素为单位) */
{
    unsigned handle;
    int width = right-left+1;
    int size = (((long)bottom-top+1)*width*4)/1024+1;
    unsigned from = (top+_ScreenTop)*_ScreenWidth+left;
    unsigned to = (bottom+1+_ScreenTop)*_ScreenWidth;
    unsigned i,j;

    if((handle=_GetXMS(size))!=0)
    {
        /* -- 参数准备 ----- */
        _Emb.len = width;
        _Emb.sour_han = 0;
        _Emb.sour_off = FP_SEG(_HanFont);
        _Emb.sour_off <<= 16
        _Emb.sour_off += FP_OFF(_HanFont)
        _Emb.dest_han = handle;
        _Emb.dest_off = 0;

        /* -- 将屏幕数据逐行经常规内存转储至扩充内存 ---- */
        for(j=from;j<to;j+=_ScreenWidth)
        {
            for(i=0;i<4;i++)
            {
                outportb(0x3ce,4);
                outportb(0x3cf,i);
                movedata(0xa000,j,FP_SEG(_HanFont),FP_OFF(_HanFont),width);
                _MoveDataXMS();
                _Emb.dest_off += width;
            }
        }
    }
    return handle
}

```

程序4-13 由扩充内存将图象数据恢复到屏幕上。

这次我们仍然使用汉字缓冲区 \_HanFont 作为中间媒介,将屏幕图象块逐行由扩充内

存经常规内存拷贝到显示存储器 VRAM 中。

```

/* -----
   程序 putblockXMS : 由扩充内存将图象数据恢复到屏幕上
   ----- */

#include <dos.h>
#include <stdio.h>
#include <hanenv.h>

extern int _ScreenWidth;      /* 逻辑屏幕宽度      */
extern int _ScreenTop;        /* 当前屏幕首行位置 */

void _Cdecl putblockXMS(left,top,right,bottom,handle)
int left;      /* 屏幕图象块左上角列坐标(以字节为单位) */
int top;       /* 屏幕图象块左上角行坐标(以像素为单位) */
int right;     /* 屏幕图象块右下角列坐标(以字节为单位) */
int bottom;    /* 屏幕图象块右下角行坐标(以像素为单位) */
int handle;    /* 屏幕图象数据在扩充内存中的句柄      */
{
    int width = right-left+1;
    unsigned from = (top+_ScreenTop)*_ScreenWidth+left;
    unsigned to = (bottom+1+_ScreenTop)*_ScreenWidth;
    unsigned i,j;

    /* -- 参数准备 ----- */
    _Emb.len = width;
    _Emb.sour_han = handle;
    _Emb.sour_off = 0;
    _Emb.dest_han = 0;
    _Emb.dest_off = FP_SEG(_HanFont);
    _Emb.dest_off <<= 16;
    _Emb.dest_off += FP_OFF(_HanFont);

    /* -- 将扩充内存数据经常规内存送入显示存储器 VRAM -- */
    for(j=from;j<to;j+=_ScreenWidth)
    {
        for(i=1;i<9;i*=2)
        {
            outportb(0x3c4.2);
            outportb(0x3c5.i);
            _MoveDataXMS();
            movedata(FP_SEG(_HanFont),FP_OFF(_HanFont),0xa000,j,width);
            _Emb.sour_off += width;
        }
    }
}

```

```
    }  
}  
  
/* —— 恢复 VGA 寄存器状态并释放扩充内存 —— */  
outportb(0x3c4, 2);  
outportb(0x3c5, 0x0f);  
_FreeXMS(handle);  
}
```

在使用以上两个函数保存和恢复屏幕上矩形区域中的图象时要注意矩形的宽度必需设置为偶数字节, 否则不能正确恢复原来的图象。这是 XMS 驱动程序所要求的。

XMS 的功能很多, 我们在 HANENV 系统中只编写了和其中一部分功能相对应的库函数。读者也可以根据自己的需要, 参考有关资料, 参照上面这些函数的编写方法编写实现其他功能的库函数。

## HANENV 系统的初始化与装配

### 5.1 HANENV 系统的初始化

由前面各章可知,为了实现以直接写屏的方式处理屏幕显示问题,需要将 VGA 的工作模式由字符模式转换为图形模式12H;为了实现屏幕分割和平滑滚动,需要改造原来的鼠标光标驱动程序,并设置若干全局变量以存储屏幕参数。在以后各章中我们还可看到,在处理时钟、定时器以及正文方式时也需要预先做些准备工作,包括设置一批全局变量。我们将这些准备工作集中在一个初始化程序模块中处理,这就是函数模块 `init_hanenv`。

在设计初始化模块时还要考虑一个问题,即应用程序能否在运行期间暂时返回到 DOS 提示符状态下,以便于操作人员进行一些诸如查看目录、拷贝文件等 DOS 工作,然后再回到应用程序中去。要实现这个功能,就还要保存好原来西文 DOS 状态下的屏幕状态,以备调用 DOS 时恢复屏幕之用。当然,应用程序调用 DOS 之前的图形方式下的屏幕内容也要保存下来。

另外,DOS 允许用户使用组合键 `Ctrl+C` 或 `Ctrl+Break` 中断应用程序的执行。但是由于使用 HANENV 系统编写的应用程序中重新设置了鼠标中断 `33H` 和时钟中断 `1CH`,因此如果使用以上组合键中断应用程序的话,则这些中断就得不到恢复,可能引起死机等故障。因此我们在初始化 HANENV 系统时将处理组合键 `Ctrl+C` 的中断 `23H` 改为一个空处理,并相应修改了键盘硬中断 `09H`,这样在使用 HANENV 系统编写的应用程序时就无法再使用组合键 `Ctrl+C` 或者 `Ctrl+Break` 来中断程序的运行了。

在应用程序结束之时,恢复原来的工作状态也是必要的。尤其是鼠标驱动程序,由于其中的鼠标光标驱动部分已由 HANENV 系统接管,因此在退出 HANENV 系统后如果不恢复原来的光标驱动,就有可能造成系统死锁。我们把恢复原来西文字符工作状态所需要的操作集中在退出 HANENV 系统函数 `close_hanenv` 中进行处理。

保存西文字符工作状态要做两项工作:首先要将显示存储器由 `B800`段开始的4 096 个字节(当前屏幕显示内容)保存到内存中去;然后再将当前显示适配器状态(包括 VGA 寄存器、BIOS 数据区和数模转换控制寄存器 DAC)全部保存起来,这可以通过调用 BIOS 的显示器中断 `INT10H` 的 `1CH` 号功能来完成。`INT10H` 的 `1CH` 号功能共有三个子功能,分别为查

询保存显示卡状态所需的存储量、保存和恢复显示适配器状态。其调用方法如下：

### 1. 查询保存显示卡状态所需的存储量

入口寄存器设置：

AH = 1CH

AL = 0

CX = 被保存数据的类型

返回寄存器：

AL = 1CH

BX = 所需的存储量(以64字节组成的块计算)

其中被保存数据的类型中各位的含义为

bit 0 : VGA 卡上的显示寄存器

bit 1 : BIOS 数据区

bit 2 : 数模转换控制寄存器 DAC

### 2. 保存显示适配器状态

入口寄存器设置：

AH = 1CH

AL = 1

CX = 被保存数据的类型(解释同上)

ES:BX = 指向存储缓冲区的指针

返回寄存器：

AL = 1CH

### 3. 保存显示适配器状态

入口寄存器设置：

AH = 1CH

AL = 2

CX = 被保存数据的类型(解释同上)

ES:BX = 指向存储缓冲区的指针

返回寄存器：

AL = 1CH

程序5-1 保存和恢复原西文字符模式的屏幕信息。

```
/* -----
   程序 _SaveEnv 和 _LoadEnv : 保存和恢复西文字符模式屏幕信息
   ----- */
#include <dos.h>
#include <mem.h>
```



```

/* -- 保存原西文字符模式的屏幕信息所需的全局变量 -- -- */
unsigned char _Video[5120];

/* -- 函数 _SaveEnv. -- -- -- -- -- */
void _Cdecl _SaveEnv(void)
{
    /* -- 保存屏幕 -- -- -- -- -- */
    movedata(0xb800,0,FP_SEG(_Video),FP_OFF(_Video)+1024,4096);

    /* -- 保存当前显示适配器状态 -- -- -- -- -- */
    _ES = FP_SEG(_Video);
    _BX = FP_OFF(_Video);
    _CX = 7;
    _AL = 1;
    _AH = 0x1c;
    geninterrupt(0x10);
}

```

恢复原西文字符方式所要做的工作与保存时正好相反：首先恢复原来的显示适配器状态，然后再恢复显示屏幕的内容。

```

/* -- 函数 _LoadEnv -- -- -- -- -- */
void _Cdecl _LoadEnv(void)
{
    /* -- -- -- -- -- 恢复原来的显示适配器状态 -- -- -- -- -- */
    _ES = FP_SEG(_Video);
    _BX = FP_OFF(_Video);
    _CX = 7;
    _AL = 2;
    _AH = 0x1c;
    geninterrupt(0x10);

    /* -- -- -- -- 恢复屏幕显示内容 -- -- -- -- -- */
    movedata(FP_SEG(_Video),FP_OFF(_Video)+1024,0xb800,0,4096);
}

```

有了保存和恢复原西文字符模式的程序模块之后就可以设计我们的初始化和退出函数了。

#### 程序5-2 初始化 HANENV 系统。

该程序模块的结构为首先设置用于定义屏幕结构、汉字处理、正文方式以及时钟和定时

器等的全局变量和函数指针,然后按以下步骤完成 HANENV 系统的初始化工作:

存储原西文屏幕信息;

设置 VGA 的图形模式12H;

如果需要的话,设置逻辑屏幕宽度和屏幕分割;

设置鼠标驱动程序;

设置新的 BIOS 中断1CH,以实现光标、时钟和定时器功能。

```

/* -----
   函数 init_hanenv : 初始化 HANENV 系统
   ----- */

#include (hanenv.h)

/* --- 全局变量定义 ----- */
#ifndef ERR_NO
#define ERR_NO
int _ErrorNo = 0;
#endif

int _ScreenTop      = 0;          /* 当前屏幕首行位置 */
int _ScreenWidth    = 80;        /* 逻辑屏幕宽度(以字节为单位) */
int _ScreenHigh     = 480;       /* 逻辑屏幕高度(以象素为单位) */
int _WindowLeft     = 0;         /* 屏幕显示窗口列坐标 */
int _WindowTop      = 0;         /* 屏幕显示窗口行坐标 */
int _WindowHigh     = 480;       /* 屏幕显示窗口高度 */

int _MouseLeft      = 0;         /* 鼠标活动左边界 */
int _MouseTop       = 0;         /* 鼠标活动上边界 */
int _MouseRight     = 639;       /* 鼠标活动右边界 */
int _MouseBottom    = 479;       /* 鼠标活动下边界 */
int _MouseSpeed     = 200;       /* 鼠标响应速度 */

int _TextCol        = 0;         /* 光标当前列 */
int _TextLine       = 0;         /* 光标当前行 */
int _TextWinLeft    = 0;         /* 正文窗口左边界 */
int _TextWinTop     = 0;         /* 正文窗口上边界 */
int _TextWinRight   = 79;        /* 正文窗口右边界 */
int _TextWinBottom  = 480;       /* 正文窗口下边界 */
int _TextColor      = LIGHTGRAY; /* 正文字符颜色 */
int _Background     = BLACK;     /* 正文区背景色 */
int _EditColor      = BLACK;     /* 编辑字符颜色 */
int _EditBk        = LIGHTGRAY; /* 编辑区背景色 */
int _TitleColor     = WHITE;     /* 标题字符颜色

```

```

int _TitleBk      = BLUE;      /* 标题区背景色          */
int _TabColor     = RED;       /* 表格颜色              */
int _TabBk        = WHITE;     /* 表格背景色            */
int _Xtimes       = 1;        /* 横向放大倍数          */
int _Ytimes       = 1;        /* 纵向放大倍数          */
int _VideoBusy    = NO;       /* 显示卡寄存器组工作标志 */
HZK *_CurrentHZK  = NULL;     /* 当前汉字库            */
int _BoxLineColor = BLACK;     /* 框线颜色              */
unsigned _BarColor = GRAY_BAR; /* 按键颜色              */
unsigned char _HanFont[1152]; /* 汉字点阵缓冲区        */
void interrupt far (*_Oldint1CH)(); /* 原 BIOS 中断 INT1CH 地址 */
void interrupt far (*_Oldint09H)(); /* 原 BIOS 中断 INT09H 地址 */
void interrupt far (*_Oldint23H)(); /* 原中断 INT23H( ^ C )   */
void far (*_Int1CHfun[10])(); /* 时钟处理程序向量      */

```

```
extern void _Cdecl _SmoothScroll(int x,int y);
```

```
extern void _Cdecl set_mode_ctrl(void);
```

```

/* -----
   函数 emptyfunction : 空函数,用于定时器中
   ----- */

```

```

unsigned _Cdecl emptyfunction(unsigned h)
{
    return h;
}

```

```

/* -----
   函数 _Newint1CH ; 新1CH 中断处理
   ----- */

```

```

void interrupt _Newint1CH(void)
{
    static int i;
    for(i = 0; i < 10; i++)
        if(_Int1CHfun[i])
            (*_Int1CHfun[i])();
    (*_Oldint1CH)();
}

```

```

/* -----
   函数 _Newint09H ; 新09H 中断处理
   ----- */

```

```

void interrupt _Newint09H(void)
{

```

```

unsigned char ch      = inportb(0x60);
unsigned char excode = peekb(0x40,0x17);

if((ch==224 || ch==46) && excode&4)
    pokeb(0x40,0x17,excode&0xfb);
(*_Oldint09H)();
}
/* -----
   函数 _Newint23H : 新23H 中断处理(^ C)
   ----- */
void interrupt _Newint23H(void
{
}
/* -----
   函数 init_hanenv : 初始化 HANENV 系统的汉字处理环境
   ----- */
void _Cdecl init_hanenv(void)
{
    int i;

    /* ----- 存储原西文屏幕信息 ----- */
    _SaveEnv();

    /* ----- 设置 VGA 的图形模式12H(640×480×16) ----- */
    _AH    = 0x00;
    _AL    = 0x12;
    geninterrupt(0x10);

    /* ----- 设置逻辑屏幕宽度 ----- */
    if(_ScreenWidth > 80)
    {
        _VideoBusy = YES;
        outportb(0x3d4,0x13);
        outportb(0x3d5,_ScreenWidth/2);
        poke(0,0x44a,_ScreenWidth);
        set_mode_ctrl();
        _SmoothScroll(_WindowLeft,_ScreenTop);
        _VideoBusy = NO;
    }

    /* ----- 设置屏幕分割 ----- */
    if(_WindowHigh<480)
    {

```

```

    _SplitScreen(_WindowHigh);
    _ScreenTop = 480 - _WindowHigh;
    smooth_scroll(_WindowLeft, _ScreenTop);
}

/* -- 设置鼠标 ----- */
reset_mouse();
set_mouse_range(0, 0, _ScreenWidth * 8 - 1, _ScreenHigh - 1);
light_mouse();

/* -- 设置新的 BIOS 中断1CH ----- */
_Triggers[0].press_key = 1; /* 触发器0专为定时器而保留 */
_Triggers[0].trigger = emptyfunction;
for(i=0; i<10; i++)
    _Int1CHfun[i] = NULL;
_Oldint1CH = getvect(0x1c); /* 保存原中断1CH的指针 */
setvect(0x1c, _Newint1CH);
_Oldint09H = getvect(0x09); /* 保存原中断09H的指针 */
setvect(0x09, _Newint09H);
_Oldint23H = getvect(0x23); /* 保存原中断23H的指针 */
setvect(0x23, _Newint23H);
}

```

我们要说明的是在应用程序中如果要设置逻辑屏幕尺寸和屏幕分割的话，必须在使用 `init_hanenv` 函数之前调用下列函数：

```

void set_screen_width(int width); /* 设置逻辑屏幕宽 */
void set_screen_high(int high); /* 设置逻辑屏幕高度 */
void split_screen(int high); /* 设置屏幕分割, 参数为主屏幕高 */

```

实际上，以上所列为定义于头文件 `hanenv.h` 中的带参宏，其定义可参看第14章：“HANENV 系统的头文件”，使用说明可以参看第15章：“HANENV 系统的库函数”。

### 程序5-3 退出 HANENV 系统。

退出 HANENV 系统的主要步骤为  
 恢复原来的 BIOS 中断1CH；  
 重置鼠标驱动程序；  
 恢复原来的西文屏幕显示内容。

```

/* -----
    函数 close_hanenv : 退出 HANENV 系统

```

```

----- */
#include <dos.h>

extern void _Cdecl _LoadEnv      (void);

void _Cdecl close_hanenv(void)
{
    /* --- 恢复原来的 BIOS 中断1CH 和中断23H ----- */
    setvect(0x1c, _Oldint1CH);
    setvect(0x09, _Oldint09H);
    setvect(0x23, _Oldint23H);

    /* --- 重置鼠标驱动程序 ----- */
    _AX      = 0;
    geninterrupt(0x33);

    /* --- 设置西文字符显示模式 ----- */
    _AH      = 0x00;
    _AL      = 0x03;
    geninterrupt(0x10);

    /* --- 恢复原来的西文屏幕显示内容 ----- */
    _LoadEnv();
}

```

有时需要在应用程序的执行过程中调用 DOS 的命令处理程序,即暂时返回到 DOS 提示符状态下,以便于操作人员进行一些诸如查看目录、拷贝文件等 DOS 工作,然后再回到应用程序中去。因此我们设计了一个函数 call \_DOS 用来实现此功能。

#### 程序5-4 临时返回 DOS 提示符状态。

由于我们在 init \_hanenv 函数中改变了鼠标驱动程序和 BIOS 中断 INT1CH 的中断向量,所以在调用 DOS 的命令处理程序的前后要注意这种情况。

```

/* -----
    程序 Call _DOS : 临时返回 DOS 提示符状态
----- */
#include <hanenv.h>
#include <mem.h>
#include <fcntl.h>
#include <stdlib.h>

int _Cdecl call _DOS(envname)

```

```

char * envname;      /* 应用程序文件名 */
{
    int handle;        /* 当前屏幕图象在扩充内存中的句柄 */
    char * * scr;      /* 当前屏幕图象在常规内存中的缓冲区指针 */
    int ismouselight = ismouselight(); /* 当前鼠标光标状态 */

    /* --- 保存当前屏幕图象 --- */
    if(ismouselight)
        delight_mouse();
    if((handle=getblockXMS(0,0,79,479))==0
        && (scr=getblock(0,0,79,479))==NULL)
        return 0;

    /* --- 保存显示适配器状态 --- */
    _ES = FP_SEG(_HanFont);
    _BX = FP_OFF(_HanFont);
    _AH = 0x1c;
    _CX = 7;
    _AL = 1;
    geninterrupt(0x10);

    /* --- 恢复原来的 BIOS 中断1CH --- */
    setvect(0x1c, _Oldint1CH); /* 恢复原来的 BIOS 中断1CH */

    /* --- 重置鼠标驱动程序 --- */
    _AX = 0;
    geninterrupt(0x33);

    /* --- 设置西文字符显示模式 --- */
    _AH = 0x00;
    _AL = 0x03;
    geninterrupt(0x10);

    /* --- 恢复原来的西文屏幕显示内容 --- */
    _LoadEnv();

    /* --- 调用 DOS 命令处理程序 --- */
    printf("\nPress EXIT to return %s... \n", envname);
    system("");

    /* --- 存储西文屏幕信息 --- */
    _SaveEnv();

```

```

/* ----- 设置 VGA 的图形模式12H(640×480×16)----- */
_AH = 0x00;
_AL = 0x12;
geninterrupt(0x10);

/* ----- 恢复显示适配器状态 ----- */
_ES = FP_SEG(_HanFont);
_BX = FP_OFF(_HanFont);
_AH = 0x1c;
_CX = 7;
_AL = 2;
geninterrupt(0x10);

/* ----- 恢复原屏幕图象 ----- */
if(handle)
    putblockXMS(0,0,79,479,handle);
else
    putblock(0,0,79,479,scr);

/* ----- 设置鼠标 ----- */
reset_mouse();
set_mouse_range(0,0,_ScreenWidth*8,_ScreenHigh);
if(ismouselight)
    light_mouse();

/* ----- 设置新的 BIOS 中断1CH ----- */
_Oldint1CH = getvect(0x1c);    /* 保存原中断1CH 的指针 */
setvect(0x1c,_Newint1CH);
}

```

## 5.2 HANENV 系统的装配

随本书一起发行有软盘《汉字编程环境 HANENV》一套,内容包括 HANENV 系统头文件、已经编译成功的库文件、工具软件,以及所有库函数的源程序和 HANENV 系统专用的汉字库等。其详细目录如下:

1# 盘:

子目录 <include>	: hanenv.h —— HANENV 系统的头文件
子目录 <lib>	: hanenvl.lib —— HANENV 系统的大模式函数库
子目录 <source>	: source.exe —— HANENV 系统的源程序自解压缩文件
子目录 <tools>	: hantool.exe —— 字库管理工具
	ucxzk.exe —— UC DOS 下的小字库制作工具
	pes.exe —— 程序文本编辑器



- 子目录 <source> —— hantool 和 ucxzk 的源程序
- 子目录 <system> : himem.sys —— 扩充内存管理规范 XMS 的驱动程序  
mouse.com —— 鼠标驱动程序
- 子目录 <hzk> : hzk16.exe —— 16×16点阵简繁体字库自解压缩文件  
wbx.cod —— 五笔字型输入法码表

2# 盘:

- hzk24.exe : 24×24点阵宋仿黑楷四体字库自解压缩文件

以上两张软盘随本书一起发行。除此而外,本书还配有一套扩展功能盘,内容包括32×32点阵、36×40点阵的宋、仿宋、黑、楷各体汉字库,以及已经编译成功的 HANENV 系统的小、中、紧缩和巨型模式函数库,有兴趣的读者可按本书封底介绍的方法向出版社邮购。

HANENV 系统的安装很简单,首先在硬盘中安装 Turbo C 2.0 或 Borland C++,然后将《汉字编程环境 HANENV》的1# 盘插入 A 驱动器,拷贝头文件 hanenv.h 和大模式函数库文件:

```
C>COPY A:\INCLUDE\HANENV.H C:\TC\INCLUDE
```

```
C>COPY A:\LIB\HANENV.LIB C:\TC\LIB
```

然后使用 Turbo C 的使用程序 tlib 将大模式函数库文件加入 Turbo C 的相应模式的运行库中:

```
tlib cl+hanenv.lib
```

此后在使用 Turbo C 编写调试程序时就可以象使用 Turbo C 的其他库函数一样使用 HANENV 系统中的库函数了。

当然,也可以不将 HANENV 系统的库函数直接加入 Turbo C 的运行库,而代之以使用工程文件的方法。具体方法可以参看 Turbo C 的有关内容。

使用 HANENV 系统编写应用程序还要用到汉字库。HANENV 系统配有多种汉字库,具体使用哪种字库应该根据实际情况选用。一般来说,说明、菜单、提示等均可使用16×16点阵的简体字库,而封面、标题等可以使用24×24以上的高点阵字库。不过高点阵字库占用的存储很多,而通常标题、封面所用汉字无多,因此可以使用 HANENV 系统所配的工具软件 hantool 等制作成高点阵的小字库使用。

安装16×16点阵字库最简单的方法是将其直接拷贝到应用程序的编程子目录中去。但这样一来,如果同时开发若干个应用程序,硬盘上势必就有好几个同样的汉字库文件,造成空间浪费。因此我们建议专门为 HANENV 系统建立一个子目录,将所有的字库文件都拷贝到该目录中,在应用程序中调用 load\_HZK 函数时字库名之前加上相应的路径即可。如果需要开发编写更加通用的应用程序(如商品软件),事先无法得知用户机器硬盘配置,则可以在应用程序中使用自动搜索路径的方法:

```
#include <dir.h>
```

.....

```
char * hzkname=searchpath("hzk16j");
```

```
....
```

```
load _HZK(2,16,hzkname,XMS);
```

以上程序段中的 searchpath 系 Turbo C 的库函数(其说明在头文件 dir.h 中),功能为在当前路径设置中去查找给定的文件,如果查找成功则返回带路径的文件名,否则返回空指针。这样,只要将字库文件所在的路径(即 HANENV 系统子目录)加入批处理文件 autoexec.bat 中,则无论应用程序安装在哪个子目录下均可使用汉字库文件了。

如果编写需要汉字输入功能的应用程序,可能要用到自定义汉字输入法的码表文件。HANENV 系统的#1软盘的 HZK 子目录中有一个名为 wbx.cod 的文件,就是五笔字型输入法的码表文件。该文件的使用方法和汉字库文件类似,如果不在应用程序的当前目录下,就应该在文件名前加上路径,或者使用 Turbo C 的 searchpath 函数。

使用 HANENV 系统最好配有鼠标。如果所用计算机尚未安装鼠标,那么首先应该选购一个鼠标。鼠标的种类很多,大体上可以分为机械式和光电式两大类。机械式鼠标价格便宜,使用方便,但可靠性稍差。光电式鼠标价格稍贵,使用时必需和专用的底板配合,但可靠性较高。读者可以根据实际情况进行选择。

鼠标的安装可以分为两个步骤:首先将鼠标插头插入计算机背面多功能卡(IDE 卡)上的串行接口插座,然后用螺丝紧固(螺丝系鼠标插头上所配)。注意串行接口的插座有9芯和25芯两种,在选配鼠标时应注意挑选。如果插头、插座规格不配套,也可以使用9芯转25芯转接线转接。有些计算机带的鼠标使用专用插座,如果需要更换而又一时找不到专用鼠标时,也可以仿照上法将鼠标接在串行接口上。安装好鼠标以后,还需要安装鼠标驱动程序方可使用鼠标。安装鼠标驱动程序有两种方法:如果鼠标驱动程序的格式为设备驱动程序(文件名后缀为.SYS),则应将其加入设备配置文件 config.sys 中;

```
DEVICE=C:\DOS\MOUSE.SYS
```

如果鼠标驱动程序的格式为命令文件(文件名后缀为.COM),则可直接运行此鼠标驱动程序。但为了避免每次使用鼠标之前都要运行鼠标驱动程序的麻烦,可以将鼠标驱动程序加入批处理文件 AUTOEXEC.BAT 中。这样,每次开机时就可以自动执行鼠标驱动程序的安装了。一般鼠标驱动程序随鼠标一起发售。如果一时找不到,可以从《汉字编程环境 HANENV》的1#盘的 SYSTEM 子目录中拷贝。我们选用的鼠标驱动程序是第二种,可以将其拷贝至 C 盘根目录下或 DOS 子目录中,然后在批处理文件中加入一行:

```
C:\DOS\MOUSE
```

或

```
C:\MOUSE
```

即可。

如果读者的计算机中的 DOS 版本在5.0以上,则可以直接使用 DOS 子目录中的扩充内存调用规范(XMS)的驱动程序 HIMEM.SYS 以支持 HANENV 系统对扩充内存的调用,其方法为在设备配置文件 config.sys 中加入一行:

```
DEVICE=C:\DOS\HIMEM.SYS
```

其实,即使是使用低版本 DOS 的计算机也可以使用扩充内存。在《汉字编程环境 HANENV》的1#盘的 SYSTEM 子目录中也有文件 HIMEM.SYS,读者可以直接拷贝至 DOS 子目录中,然后再在设备配置文件中加入上述命令行。

注意在使用以上各个命令行时应该根据机器的实际配置情况进行修改。

最后,《汉字编程环境 HANENV》1#软盘中的 SOURCE 子目录中还有一个名为 SOURCE.EXE 的自展压缩文件,其中包扩了 HANENV 的所有库函数的源程序,供读者参考使用。

综上所述,我们建议在硬盘中建立如下子目录树安装 HANENV 系统:

```
TC----- INCLUDE    ; HANENV.H
              LIB      ; HANENV*.LIB
(以上目录由 Turbo C 的安装程序自动建立)
HANENV-- SOURCE    ; *.C, *.ASM
              TOOLS   ; *.EXE
              HZK *
              WBX.COD
```

### 5.3 HANENV 系统的工具软件

在 HANENV 系统的#2软盘中还有三个工具软件:pes、hantool 和 ucxzk。其中 pes 是一个使用 HANENV 系统开发的文本编辑程序,窗口式图形界面,全面支持鼠标操作,自带汉字处理功能,可以直接在西文 DOS 环境下编辑含有汉字的文本或程序文件。后两个软件都是用于制作供 HANENV 系统中汉字显示函数调用的小字库或特殊图标。特别是 hantool,本身也是应用 HANENV 系统设计编程,功能齐全,界面比较新颖清晰。其源程序也在 HANENV 系统的#1软盘中,供读者在使用 HANENV 系统编写应用程序时参考。下面我们分别介绍这三个应用软件的功能设置和使用方法:

#### 5.3.1 程序文本编辑器 pes

pes 编辑器使用 WINDOWS 风格的窗口式图形界面,全面支持鼠标操作。编辑器的各项功能设置灵活,诸如编辑窗口中的汉字的字体和大小、以及各种编辑项目如文字、背景、编辑块、回车换行符、图标等的颜色以及编辑窗口的位置和大小均可重新定义,以适应不同用户的要求。pes 编辑器的编辑窗口如图5-1所示。

pes 编辑器的使用方法为

```
PES [〈路径\文件名〉]
```

其中文件名中可以加通配符“\*”和“?”。如果不使用〈路径\文件名〉部分,则可直接进入主菜单。主菜单的第一项为“编辑”,可以进入编辑界面编辑文件,其他几项则用于设置编辑器的工作参数(见第五部分:参数设置)。在编辑状态下使用命令^KQ 可以退至主菜单,而选用命令^KD 或^KX 则可直接退出 PES。

pes 全面支持鼠标操作,例如在编辑状态下的鼠标操作有:

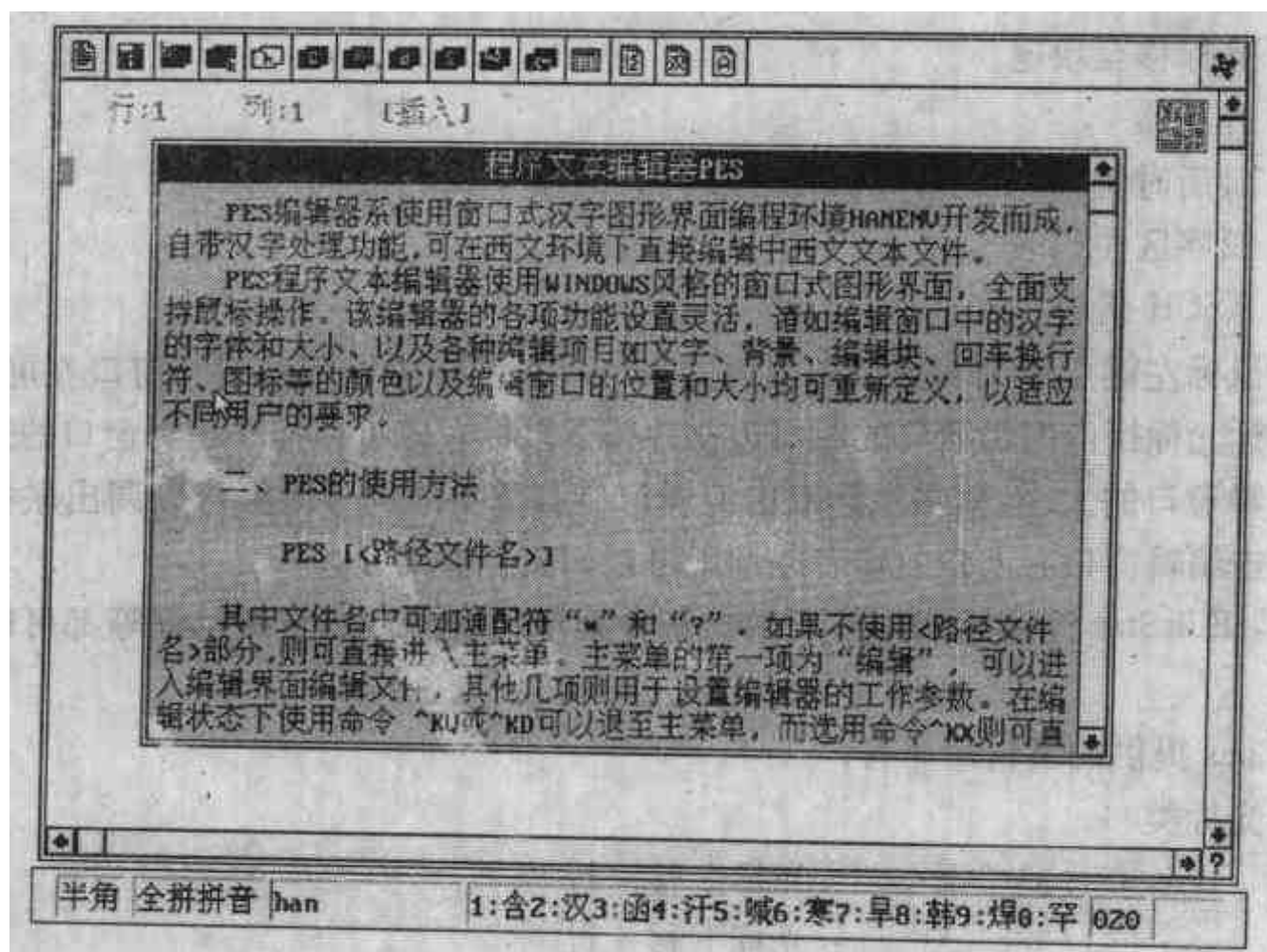


图5-1 pes 编辑器的编辑窗口


使用鼠标移动光标至新的编辑位置:

- \* 鼠标点击某字符或汉字可将光标移向该字
- \* 鼠标点击编辑窗口右边的滚动条两端的按键可使光标上移或下移一行
- \* 鼠标点击编辑窗口下方的滚动条两端的按键可使光标左移或右移一字
- \* 鼠标点击编辑窗口右边的滚动条条身可以上翻或下翻一页
- \* 用鼠标拖动编辑窗口右边滚动条中的游标至某位置,可按比例调整当前行的位置
- \* 鼠标点击编辑窗口下方的滚动条条身可使光标向左或向右移动一个制表符宽度(省缺值为8)
- \* 用鼠标拖动编辑窗口下方滚动条中的游标至某位置可按比例调整光标在行中位置

使用鼠标左键点击编辑窗口顶部的图标可以直接调用以下命令:

- \* 文件操作命令菜单
- \* 和文件存盘
- \* 定义块首
- \* 定义块尾
- \* 取消块定义
- \* 块拷贝
- \* 块移动
- \* 块删除

- \* 块内容写入文件
- \* 光标移至块首
- \* 光标移至块尾
- \* 计算器
- \* 日历时钟
- \* 汉字区位码表
- \* ASCII 码表

使用鼠标左键点选编辑窗口右上角的“ ”形图标可以拖动编辑窗口在屏幕上移动,如果将鼠标光标指向编辑窗口的四面边框并按下鼠标左键可以拖动编辑窗口的边框移动从而改变编辑窗口的大小,使用鼠标点击编辑窗口右下角的问号图标可以调出联机帮助而使用鼠标点击编辑窗口右上角的作者印章则可以调出作者简介。

另外, EditStar 系列编辑器中所带的各种菜单、文件目录表、提示框等都可以使用鼠标操纵。

目前 pes 提供的编辑命令有:

#### 移动光标类

↑	或 ^ E	: 光标上移一行
↓	或 ^ X	: 光标下移一行
→	或 ^ S	: 光标右移一个字符或汉字
←	或 ^ D	: 光标左移一个字符或汉字
PgUp	或 ^ R	: 向前翻一页
PgDn	或 ^ C	: 向后翻一页
Home	或 ^ QS	: 光标移至行首
End	或 ^ QD	: 光标移至行尾
^ Home	或 ^ R	: 光标移至当前页第一行
^ End	或 ^ C	: 光标移至当前页最后一行
^ PgUp		: 光标移至文件第一行
^ PgDn		: 光标移至文件最后一行

#### 编辑类

Ins		: 切换插入/改写状态
Del	或 ^ G	: 删除当前光标下的字符或汉字
Backspace	或 ^ H	: 删除当前光标前的字符或汉字
^ Y		: 删除一行
^ N		: 在当前光标处插入一行
^ U		: 在当前光标处恢复被删除的内容(汉字或字符、行、块)。 本命令还用于将计算器和日历命令的结果插入当前光标处。

#### 块操作类

^ KB	或 F5	: 定义块首
^ KK	或 F6	: 定义块尾

^ KH		: 关闭/打开块显示
^ KC	或 F7	: 块拷贝
^ KY	或 F9	: 块删除
^ KV	或 F8	: 块移动
^ KW		: 将块中内容写入文件
^ QB		: 光标移至块首
^ QK		: 光标移至块尾
查找替换类		
^ QF		: 查找
^ QA		: 替换
^ L		: 继续查找或替换
^ QL		: 光标移至指定行
文件操作类		
^ KS	或 F2	: 文件存盘(不返回)
^ KR	或 F4	: 读文件至当前光标处(文件名中可加通配符“*”和“?”)
^ KD	或 ^ KX	: 文件存盘退出
^ KQ		: 不存盘退出
F10		: 临时返回 DOS 命令行
F3		: 清编辑区
^ QN		: 转换编辑窗口
ESC		: 文件操作菜单
F1	或 ^ KJ	: 联机帮助
工具类		
^ TA		: ASCII 码表
^ TC		: 计算器
^ TH		: 汉字区位码表
^ TT		: 日历时钟
^ F3		: 调色板

pes 编辑器的下列参数允许重新设置:

编辑器的色彩搭配,包括编辑区背景、字符、块等14种颜色;

编辑器所用字符或汉字的大小和字体(简体或繁体);

制表符的宽度;

编辑窗口的位置和大小。

以上各项调整内容和使用调色板的调整结果均可存入参数文件 pes. pal。

### 5.3.2 字库管理工具 hantool

字库管理工具 hantool 的界面如图5-2所示。

其功能设置如下

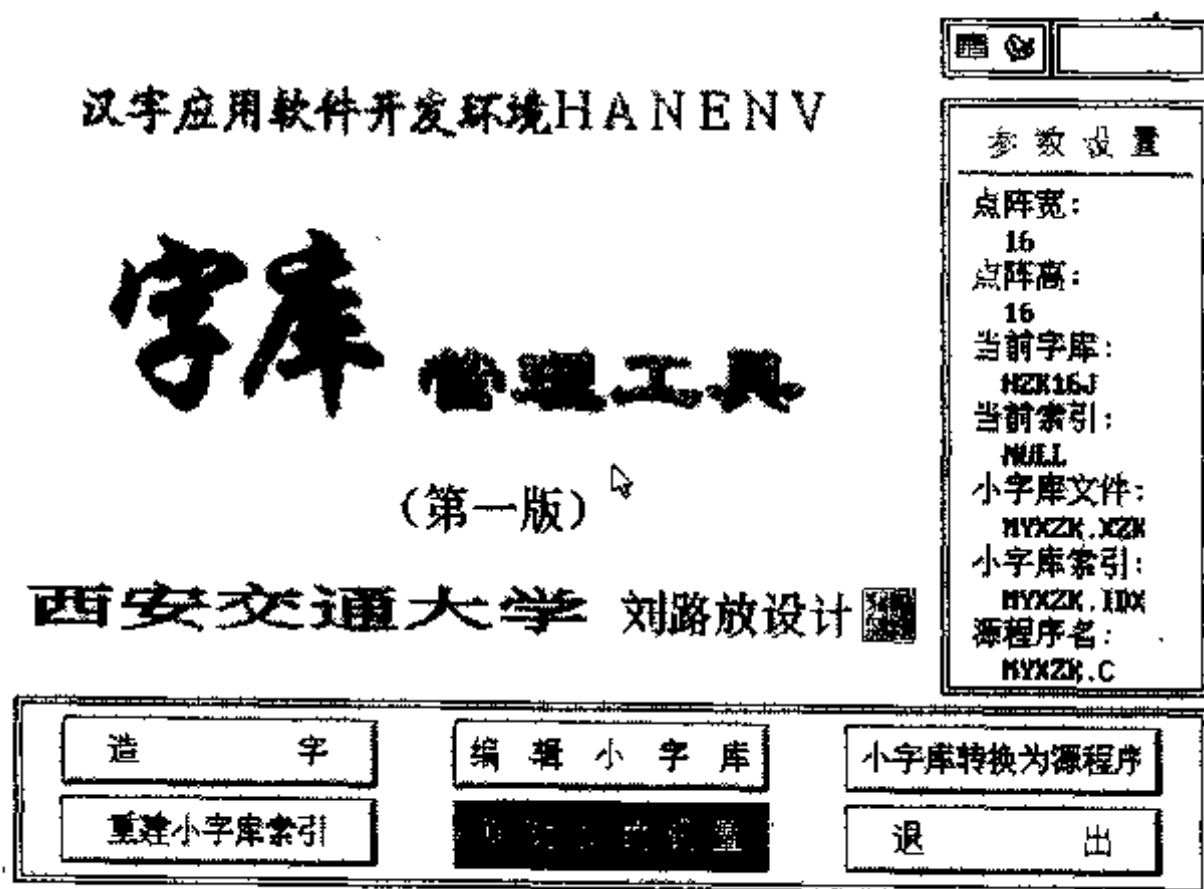


图5-2 字库管理工具 hantool 的主菜单界面

hantool 的主菜单共有六项：

1. 造字：运用鼠标绘制各种尺寸的字模点阵；
2. 编辑小字库：从标准汉字库中提取若干字模点阵组成小字库；
3. 小字库转换为源程序：将小字库转换为 C 语言的全局数组变量；
4. 重新索引小字库：检验小字库的内容并为小字库加上索引；
5. 参数设置：修改当前汉字库的名称、点阵尺寸等参数；
6. 退出：退出 hantool。

除了以上主菜单中所列功能以外，hantool 还设有如下热键和图标命令：

7. 时钟：在屏幕右上角有一时钟，每逢正点、半点报时；
8. 联机帮助：F1键启动，内容为 hantool 的简要使用说明；
9. 计算器：F2键或用鼠标点选屏幕右上角的图标启动；
10. 调色板：F3键或用鼠标点选屏幕右上角的图标启动；
11. ASCII 码表：F4键启动，标准和扩充 ASCII 码的码值—图象对照表；
12. 区位码表：F5键启动，1—87区所有国标区位码的汉字或图象；
13. 调用 DOS 命令：F10键启动，临时返回 DOS 提示行。

下面我们重点介绍几个主要模块的使用方法：

1. 主菜单的使用方法：hantool 的主菜单既可以通过使用鼠标点选相应按键来选择功能模块，也可以使用键盘控制菜单。在使用键盘控制菜单时，向左方向键和向上方向键将前移当前按键（键帽为红色），向右方向键和向下方向键将后移当前按键，按下回车键即可选择执行当前按键代表的功能模块。如果按下 ESC 键可以直接退出 hantool。



2. 参数设置模块: 参数设置模块用于修改 hantool 的下列工作参数:

- 点阵宽——用于指出当前汉字库中字模点阵和造字点阵的宽度, 以像素为单位, 必须使用8的倍数, 最小值为8, 最大值为96, 省缺值为16。

- 点阵高——用于指出当前汉字库中字模点阵和造字点阵的高度, 以像素为单位, 最小值为1, 最大值为96, 省缺值为16。

- 当前字库——供造字模块和编辑小字库模块使用的汉字库的文件名, 其点阵的宽度和高度必须与以上两项相符。当前字库可以使用 HANENV 系统提供的标准汉字库, 也可以使用由造字模块、编辑小字库模块和使用 ucxzk 程序生成的小字库。如果使用小字库则应有索引。注意, hantool 不要求当前字库文件一定在当前目录中, 但如果当前字库文件不在当前目录中, 则其路径 y 应加入批处理文件 autoexec. bat 当中。当前字库的省缺值为 HZK16J。

- 当前索引——如果使用 HANENV 系统提供的标准汉字库, 则无需索引文件, 此时当前索引项填写 NULL(空); 如果使用小字库, 则应有索引文件本项即填写索引文件的文件名。由编辑小字库模块生成的小字库本身带有索引; 而如果使用由造字模块和 ucxzk 程序生成的小字库不带索引, 必需先使用重新索引小字库模块为其建立索引文件之后方可作为当前字库。当前索引项的省缺值为 NULL。

- 小字库文件——该参数用于指明编辑小字库模块所生成的小字库、重新索引小字库模块和小字库转换为源程序模块所使用的小字库的文件名。小字库文件的省缺值为 MYXZK.XZK。

- 小字库索引——该参数用于指明编辑小字库模块和重新索引小字库模块所生成的小字库索引的文件名。小字库索引的省缺值为 MYXZK.IDX。

- 源程序名——小字库转换为源程序模块所生成的 C 语言源程序文件名, 省缺值为 MYXZK.C。

退出参数设置模块可以使用 ESC 键, 也可以使用鼠标右键。

3. 造字模块: 造字模块的界面如图5-3所示。

进入造字模块以后, 首先在屏幕左上方开辟一块造字编辑区域, 按屏幕左侧参数表中的点阵宽度和高度打上方格。在造字框的右上方是一同步显示窗口, 将以点阵的实际尺寸显示造字编辑区的内容。造字的方法很简单: 只要用鼠标左键点选造字编辑区中的某个方格, 就相当于在字模点阵的相应位置上写点(白格变蓝格)或取消点(蓝格变白格)。在造字过程中随时可以通过造字编辑区右上角的窗口观察所造点阵的实际效果。

另外, 在造字编辑区的右下方是一按键式菜单, 共有4项:

- 库中调字——从当前汉字库中取一汉字或图形字符的字模点阵送入造字编辑区。按下此键后, 屏幕最下方会出现汉字输入提示行, 汉字输入法为“双拼拼音”。如果想使用其他汉字输入法, 可以使用 Alt+F1~F10键切换。hantool 共装有区位、全拼、双拼、五笔、西文和图形符号等6种输入法, 其中五笔字形输入法需要使用外挂码表文件。五笔字型的码表文件名为 wbx.cod, 可以放在当前目录中, 也可以和字库文件放在一起。使用 Ctrl+F9键可以切换全角和半角西文符号。另外, 使用热键 F9 或用鼠标点选提示行最右端的“键盘”提示符可以调出和取消鼠标虚拟小键盘。

- 新字入库——该模块将写好的字模点阵存入汉字库。按下此键后, 屏幕上会弹出一个



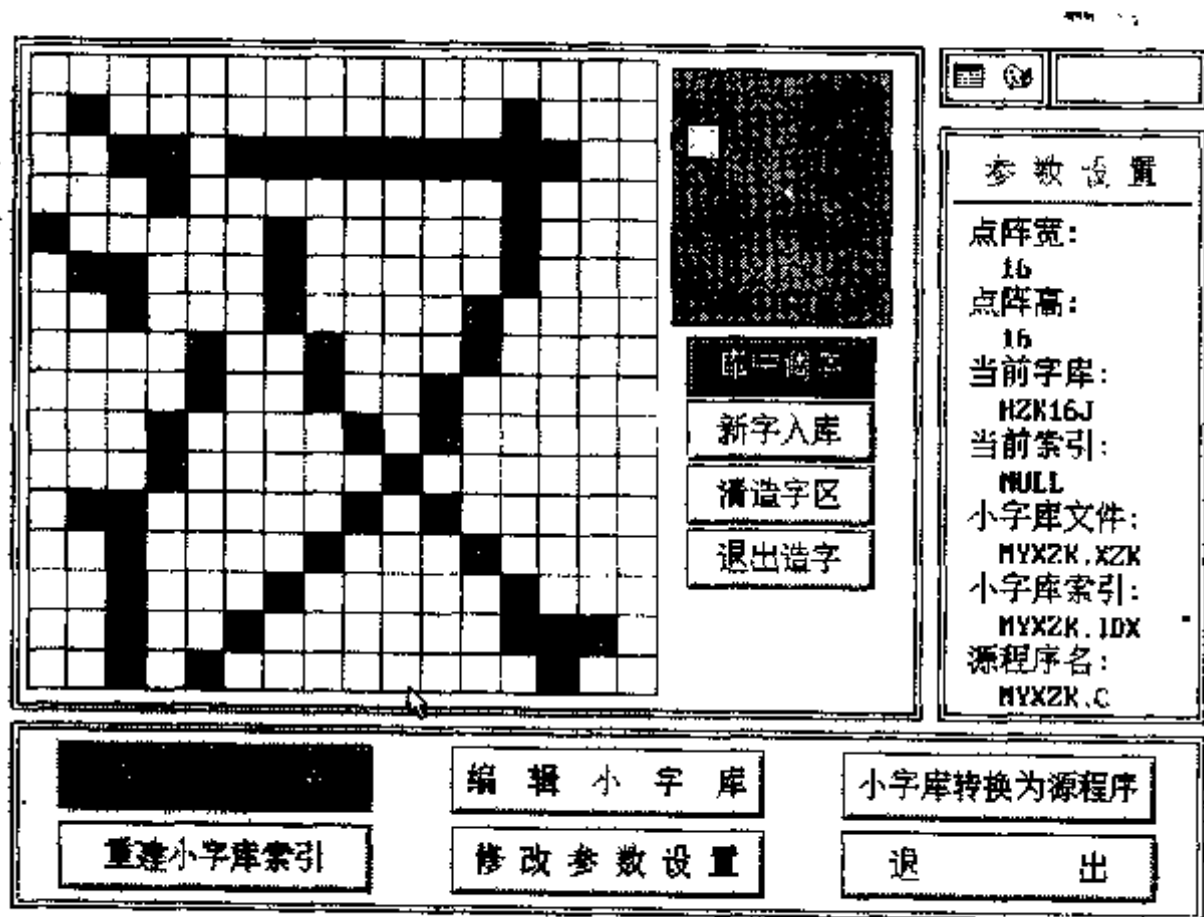


图5-3 hantool 的造字功能

提问框：“是否将新字加入当前字库？”如果回答“是”，则弹出提示行，需要输入该新字在字库中的位置。这种情况多用于修改标准字库中的某字的点阵，或者是造一标准字库中没有的非常用字，此时可以用其代替字库中原来某字（最好同音）的点阵，此后可以使用拼音输入法直接输入该字。如果是画出一个符号或图标，则可以将其放入10—15区的空白处，以后可以使用区位码调用该符号。必须说明的是新字入库以后不能立即在 hantool 中调用，因为此时只是更新了硬盘上的字库文件的内容，并未修改扩充内存中的运行字库。必需退出 hantool 后再次进入方可调用。如果要将新字另外存入小字库，则应回答“否”，此时屏幕上会再弹出一个提问框，要求输入小字库名称。该小字库可以和屏幕右侧的参数表中的小字库名相同，也可以另外起名。加入小字库的新字放在文件尾部，其在小字库中的序号会在屏幕上显示出来。不过，为了方便使用，最好再使用主菜单中的“重新索引小字库”项为小字库建立一个索引文件。

- 清造字区——按下此键后将清除造字编辑区的内容。
- 退出造字——按下此键后退出造字模块。

4. 编辑小字库模块：该模块用于从当前字库中提取某些字模组成一个小字库。进入该模块后会在屏幕左上方弹出一个小小字库编辑区，同时在屏幕下端弹出汉字输入提示行。关于提示行的说明可以参看造字模块的有关内容。注意只能输入汉字和全角图形符号，半角西文 ASCII 码不能进入小字库。一般来说输入小字库的汉字不应重复，以节约存储。在输入所有汉字之后使用回车键结束编辑。生成的小字库及其索引可以直接使用 load\_HZK 装入应用程序，也可以使用小字库转换为源程序模块，将小字库转换为全局数组的源程序后再供 DrawF、\_OouF 等函数调用。

5. 重新索引小字库模块：该模块用于为没有索引的小字库重建索引。进入该模块后会在屏幕左上方显示一个索引编辑区，该索引区的左方一框用于显示小字库中的点阵，右方一框用于显示输入的索引字。关于提示行的说明可以参看造字模块的有关内容。重建索引的方法是当索引编辑区左方的方框中出现字库中的第一个字模点阵时输入相应的汉字，此汉字即会在索引编辑区右方的方框中显示。此时左方的方框中又会出现字库中下一个字模点阵，依此类推，直到为小字库中的所有字模点阵都输入索引字。

6. 小字库转换为源程序模块：该模块用于将小字库中的字模点阵转换为一个全局数组。该数组可以直接加入应用程序的源程序中，或是单独编译成一个目标文件和应用程序的其他部分连接起来。这样做的特点是可以省去独立的小字库及其索引文件，使应用程序显得比较简洁。特别是在制作比较复杂的界面程序时，可能用到多个不同字体和大小的小字库，更应采用这种方法。该模块的使用方法最简单，只要按下主菜单中的相应按键，即可将当前小字库转换为源程序，这两个文件皆由屏幕右侧的参数表中的对应项目确定。

7. 联机帮助命令：按下热键 F1 即可调出帮助菜单。帮助菜单中有 hantool 的简要使用说明。帮助菜单的翻页可以使用鼠标操作菜单右侧的滚动条，也可以使用键盘上的 Home、End、PgUp、PgDn 和上下方向键。

8. 计算器命令：按下热键 F2 或用鼠标点选屏幕右上方的计算器图标即可调出一个按键式仿真计算器。该计算器不仅可以作四则运算，更可以进行诸如 sin、cos 等函数运算。在操作上该计算器亦与常用计算器有所不同，计算器顶部开有两个窗口，左侧窗口用于显示计算表达式，右侧窗口用于显示计算结果。计算表达式的写法与通常的横式一样，便于书写复杂的计算式。该计算器可以使用鼠标，也可以使用键盘操作。

9. 调色板命令：按下热键 F3 或用鼠标点选屏幕右上方的调色板图标即可调出一个滑标式调色板。该调色板可以通过改变 VGA 的 16 种显示色以及屏幕边缘显示色的红、绿、蓝三原色的比例而改变 hantool 的屏幕显示效果（见图 11-6）。修改过的调色板数据存储在当前目录下的 hantool.plt 文件中，以后每次使用 hantool 时就会按新的调色板数据调整显示色彩。但该调色板更重要的用途是为应用程序取得调色信息。由图 11-3 中可以看出，在调色板下方有一排各种颜色的小方块，共有 17 个，前 16 个代表当前屏幕上可以使用的 16 种色彩，最右边的一个中空黑框代表屏幕边缘颜色。在最左端的黑色小方块外面还套着一个矩形框，使用鼠标操纵调色板最下方的滚动条就可以通过移动矩形框来选择要修改的屏幕色彩。在调色板上方还有 3 个横向滚动条，分别用于调整被选中的颜色的红、绿、蓝三原色分量的强度。调整的效果可以通过被选中的颜色块的变化反映出来（但屏幕边缘色变化的效果只能通过直接观察屏幕边缘得出）。如果是为应用程序的界面调配颜色，当调整完成后，还应该记下这 3 个滚动条左面括号中的数字，以便于填写应用程序中的颜色设置函数 set\_color 的参数。

10. 查询 ASCII 码：按下热键 F4 即可调出一个 ASCII 码表。该码表包括所有 256 个标准 ASCII 码和扩充 ASCII 码。码表的第一列为十进制码，第二列为十六进制码，最后一列为 ASCII 码的显示符号（见图 11-4）。可以通过使用鼠标操作码表右边的滚动条或者使用键盘上的 PgUp、PgDn 等按键控制翻页。

11. 查询区位码：按下热键 F5 即可调出一个区位码表。该表包括国标码 1—87 区中所有区位码的符号（见图 5-4）。可以通过使用鼠标操作码表右边的滚动条或者使用键盘上的

PgUp, PgDn 等按键控制翻页。

12. 调用 DOS 命令：按下热键 F10 即可调出 DOS 环境。此时屏幕显示被恢复为进入 hantool 之前的西文版面，并显示提示：

C>Press EXIT to return HANTOOL...

Microsoft(R) MS-DOS(R) Version 6.20

(C) Copyright Microsoft Corp  
1981-1993.

C>\_

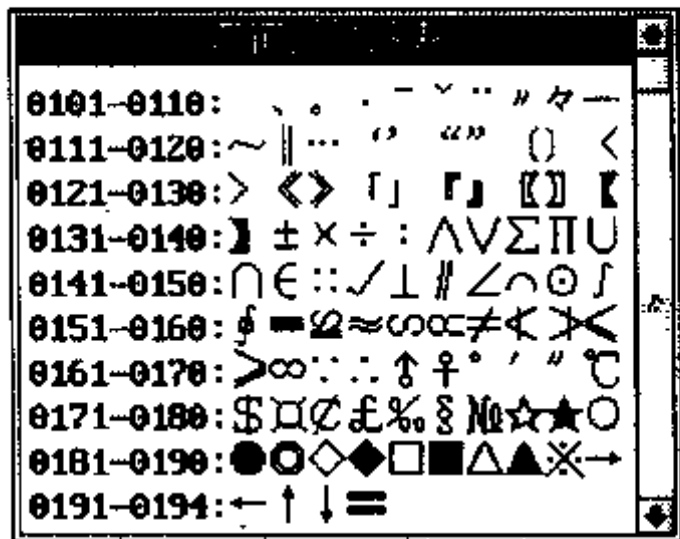


图5-4 区位码表

这样就可以使用任意 DOS 的命令进行诸如列目录、显示文件内容、拷贝文件等工作了。应当注意的是这时 hantool 程序并未退出内存，还要占用大约150K 的内存，所以不能运行特别大的程序。在做完应该在 DOS 状态下完成的工作以后，敲入命令：

C>EXIT

即可返回 hantool 中原来的工作点。

### 5.3.3 高点阵小字库制作工具 UCXZK

HANENV 系统配备的最大规模的标准字库是36×40点阵的字库，如果需要更高点阵的汉字必须另想办法。使用 hantool 的造字模块可以制作尺寸较大的字模，但是相当费时。如果读者拥有汉字操作系统 UCDOS 3.0或3.1，就可以使用 HANENV 系统的高点阵小字库制作工具 UCXZK 制作各种规格的小字库了。UCXZK 的工作原理比较简单，即利用 UCDOS 的特显功能显示所需要的字模，然后直接从屏幕上读取字模的点阵信息并将其存入小字库中。UCXZK 的使用方法如下：

1. 进入 UCDOS 并启动特显功能：

C>UP

C>TX

2. 按下列格式使用 UCXZK：

UCXZK (字模宽) (字模高) (字体号) (小字库索引)

其中字模宽和字模高均以象素为单位，最大可取96；字模宽应取8的倍数。字体号的选取可以参考 UCDOS 说明书中有关特显的部分，如果没有选配扩充字库，则

0 = 宋体

1 = 仿宋体

2 = 黑体

3 = 楷体

小字库索引是一个包括所有要加入小字库中的汉字的字符串。应该说明的是，由于 UCXZK 从屏幕上直接读取字模，所以索引字符串的长度不宜太长，以显示的字模能放在一

行上为宜。如果需要制作较大的小字库,可以分别制作几个较短小字库,然后再通过 DOS 的拷贝命令连接起来。UCDOS 生成的小字库一律取 myxzk, xzk 之名。另外,UCXZK 生成的小字库不带索引,如果需要的话可以使用 hantool 中的“重新索引小字库”功能为其生成一个索引文件。

# 第 2 篇

## 汉字处理

## 汉字显示

一般说来,要在计算机的显示屏幕上显示一个汉字可以按下述步骤进行:

首先要找到该汉字的字模,即该汉字的形状信息。记录一个汉字的形状有两种基本方法:矢量法和点阵法。矢量法的原理是使用直线段(矢量)或曲线段(如 3 次样条插值曲线)来模拟汉字的笔画或轮廓。一般说来,使用 3 次样条插值曲线的汉字轮廓字模(即所谓 PostScript 字模)的输出效果最好,在实现汉字的放大、变形等功能时可以精确地保持汉字轮廓的光滑和流畅。但矢量法也存在输出速度慢、信息存储量大和处理小号字的效果欠佳等缺点,目前主要用于汉字的精密打印输出。由于本书的基本主题是如何设计应用程序的中文用户界面,主要用到汉字的屏幕显示,所以我们没有采用矢量字模。

另一种基本方法就是点阵法。点阵法的基本原理是将汉字区域用纵横交错的等距直线划成一个个小方格,然后根据每个方格是否落在汉字的笔画上决定该方格的数据值是 1 还是 0,然后将整个区域的数据值按一定规则作为汉字的字模点阵存储起来。在显示汉字时的顺序正好相反:根据存储的字模点阵决定屏幕上相应位置像素的颜色。构造汉字点阵的方法可见图 6-1。

由图 6-1 可以看出,方格的数量越多,则汉字的形状越清楚,但所需的信息存储量也就越大。作为显示汉字的方法来说,汉字点阵的大小还要受到显示屏幕分辨率的约束。由于显示器的大小基本是固定的,除了特殊情况,通常台式机均使用 31cm(14 英寸)的彩色或黑白监视器,笔记本机多使用 23cm(9 英寸)的液晶显示器。因此我们总是希望在能清楚地表示汉字结构的条件下做到同时在显示屏幕上显示尽可能多的汉字。

设计显示用汉字点阵字模的另一个重要参考依据是微型计算机上原来的西文字符显示方式。EGA/VGA 上最常用的西文显示方式是每屏显示 25 行,每行显示 80 个字符。由于西文字符的形状是矩形的,上下长而左右窄;而汉字的基本形状为方形,同时汉字的结构比西文字符复杂,所以在设计汉字字模时一般取汉字的宽度为两个西文字符的宽度,高度和西文字符相等。这样,在显示汉字时每屏仍可显示 25 行,但每行只能显示 40 个汉字。由于汉字和西文字符的比例十分简明,所以在汉字和西文字符同屏显示时也不会引起混乱。

VGA 的图形显示模式 12H 的分辨率为  $640 \times 480$ ,即每屏可以显示 480 线,每条线由 640 个像素构成。因此,西文字符点阵的宽度为  $640 \div 80 = 8$  个像素,汉字点阵的宽度为 640

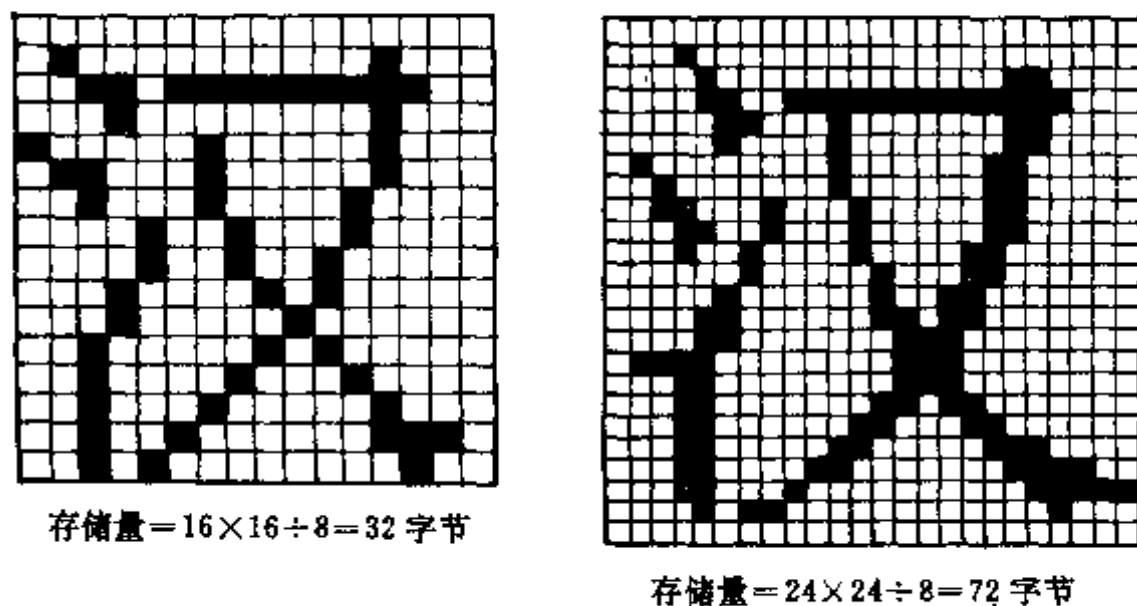


图 6-1 汉字点阵的构造

$\div 40 = 16$  个像素。在设计汉字点阵的高度时还应该考虑两个因素：一个是为了阅读方便起见，在显示汉字时应该在两行汉字之间留有一定的间距；另一个是在输入汉字时还需要一个提示行显示有关信息。因此，如果需要能够同屏显示 25 行正文，那么同屏显示的最大汉字行数实际上应为 26 行。基于以上考虑，我们取汉字点阵的高度也是 16 个像素，加上行间距两个像素点，则显示 25 行正文和一个提示行共需  $(16 + 2) \times 26 = 468$  线，这样每屏还多余  $480 - 468 = 12$  线。实际上，我们可以将这 12 条线用于装饰提示行，使之具有美观醒目的效果。

16×16 汉字点阵是基本的显示用汉字字模。有时我们的应用程序还需要在屏幕上输出一些比较大的汉字，用于应用程序的封面制作和标题显示，因此在 HANENV 系统中我们还提供了尺寸为 24×24 点阵和 48×48 点阵的汉字库以及制作更大点阵字库的工具软件，同时 HANENV 系统中的各种汉字显示函数也都具有显示和放大各种大小的汉字点阵字模的能力。

汉字的字模数据量是相当大的。以 16×16 点阵的字模为例，存储每个汉字的字模点阵就需要  $16 \times 16 = 256$  个位(bit)，也就是 32 个字节。国标 GB2312-80《信息交换用汉字编码字符集》中规定了 94 区×94 位=8 836 个编码位置，共包括了 6 763 个汉字和 682 个非汉字图形字符。在设计汉字库时，我们去掉了国标 GB2312-80 中 88 区以后的空白区，因此存放所有这些汉字和图形符号就需要  $94 \times 87 \times 32 = 261\,696$  个字节，约等于 256K。而当汉字字模的点阵尺寸更大时汉字库所需要的存储空间就更大了。

在拿到汉字字模点阵以后，就可以在屏幕上的指定位置“写”汉字了。其实，显示汉字是通过对汉字点阵进行逐点处理而完成的，因此与其说汉字是“写”到屏幕上的还不如说是“画”到屏幕上的。在显示汉字的过程中将一个点显示在屏幕上的指定位置是最重要的环节，其算法的好坏直接影响到汉字显示的速度。向屏幕写一个像素点的方法很多，例如使用 Turbo C 的图形库中的 putpixel 函数或者直接调用 BIOS 的中断 INT10H 的 0CH 号子功能都可以实现向屏幕写像素的功能。但是这两种方法的共同缺点是速度太慢，使用它们来写汉字的效果很差。因此在 HANENV 系统中我们直接对显示存储器 VRAM 和 EGA/VGA



显示卡的各寄存器进行操作,这是向屏幕上写像素点最快的方法。

其实,在第1章中我们已经介绍了 EGA/VGA 卡的显示存储器(VRAM)的组织方式的特点:在模式 12H 下对屏幕操作的基本单位并不是像素,而是由 8 个毗邻像素构成的字节。因此,每个  $16 \times 16$  点阵的汉字字模的一行(16 个像素)可以分两次写到屏幕上,整个汉字只需 32 次屏幕写操作即可完成。而利用画像素点的方法写汉字就需要  $16 \times 16 = 256$  次屏幕写操作!但是这种方法对汉字的显示位置有一定的要求:显示汉字的横坐标必需取 8 的整倍数。幸而对于大多数汉字应用程序来说这不能算是一个太大的限制。为了适应不同应用场合的要求,在 HANENV 中我们提供了两类汉字显示函数,分别使用了逐点显示和按字节显示两种直接对 EGA/VGA 显示卡进行操作的方法。

## 6.1 显示一个像素点

在屏幕指定位置写一个像素是所有屏幕图形操作的基础,其速度对整个系统的效率影响很大。在 HANENV 系统中我们采用了直接对显示存储器 VRAM 操作的方式来实现在屏幕指定位置写像素。一般说来,在 EGA/VGA 显示卡上可以使用几种不同的寄存器操作实现这一功能,但其中最快、最简便的方法是利用图形控制器的置位/复位寄存器(索引号 00H)。在这种方式下首先要将所写像素的颜色值放入置位/复位寄存器,然后将允许置位/复位寄存器(索引号 01H)设置为所有颜色平面均接收来自置位/复位寄存器的填充数据;因为是只写一个像素,因此还要将位屏蔽寄存器(索引号 08H)设置为仅像素对应位可写,然后对显示存储器 VRAM 的相应地址进行先读后写的操作即可实现写像素。这种方法的优点是一次即可同时处理所有的颜色平面。

程序 6-1 在屏幕上指定坐标处显示指定颜色的像素。

下面的程序用于在屏幕上指定坐标处显示一个指定颜色的像素点。首先我们要将其坐标换算为显示存储器 VRAM 中的地址,然后设置好置位/复位寄存器、允许置位/复位寄存器和位屏蔽寄存器的值,即可对显示存储器 VRAM 的相应地址进行读写。最后,还应该将被我们修改过的各个寄存器的值恢复为其省缺值。

```
/* -----
   函数 _PutPixel : 在屏幕上指定坐标处显示指定颜色的像素
   ----- */
#include <hanenv.h>

void _Cdecl _PutPixel(int x,int y,int color)
{
    /* --- 将像素坐标换算为显示存储器 VRAM 中的地址 --- */
    char far *addr = MK_FP(0xa000,(y+_ScreenTop)*_ScreenWidth+(x>>3));

    /* --- 计算像素在字节内的位置 ----- */
    unsigned char mask = 0x80 >> (x&7);
```



```

/* -- 设置 EGA/VGA 图形控制器的有关寄存器 ----- */
outportb(0x3ce,0);      /* 象素颜色放入置位/复位寄存器 */
outportb(0x3cf,color);
outportb(0x3ce,1);      /* 设置允许置位/复位寄存器      */
outportb(0x3cf,0x0f);
outportb(0x3ce,8);      /* 设置位屏蔽寄存器          */
outportb(0x3cf,mask);

/* -- 读写显示存储器的相应地址 ----- */
*addr &= 0xff;

/* -- 恢复被修改过的各有关寄存器 ----- */
outportb(0x3ce,8);      /* 恢复位屏蔽寄存器          */
outportb(0x3cf,0xff);
outportb(0x3ce,1);      /* 恢复允许置位/复位寄存器      */
outportb(0x3cf,0);
outportb(0x3ce,0);      /* 恢复置位/复位寄存器          */
outportb(0x3cf,0x0f);
}

```

由于写象素是其他许多显示器屏幕操作的基础,所以我们希望其速度越快越好。而提高执行速度的方法之一是在程序设计上下功夫,尽量利用寄存器和减少不必要的操作。一般认为要做到这一点最好使用汇编语言编写程序。我们知道,在 Turbo C 中可以使用行间汇编来直接调用汇编指令,这样比完全使用汇编语言编写程序要简单。

#### 程序 6-2 使用行间汇编在屏幕上指定坐标处显示指定颜色的象素。

我们使用和程序 6-1 完全相同的算法实现在屏幕上指定坐标处显示指定颜色的象素,只是这次采用了行间汇编来编写程序。

```

/* -----
   程序 _PutPixel : 在屏幕上指定坐标处显示指定颜色的象素
   ----- */
#pragma inline          /* 程序中包括了汇编程序代码      */
#include <hanenv.h>

void _Cdecl _PutPixel(int x,int y,int color)
{
    /* -- 将象素坐标换算为显示存储器 VRAM 中的地址 ----- */
    _VideoBusy = YES;
    asm mov ax,y          /* 计算偏移量并将其送入 BX 中      */
    asm add ax,_ScreenTop
}

```

```

asm mul _ScreenWidth
asm mov dx,x
asm sar dx,1
asm sar dx,1
asm sar dx,1
asm add ax,dx
asm mov bx,ax
asm mov ax,0a000h      /* 将 VRAM 的段地址送入 ES 中 */
asm mov es,ax

/* —— 设置 EGA/VGA 图形控制器的有关寄存器 —— */
asm mov dx,3ceh
asm xor al,al
asm mov ah,color
asm out dx,ax          /* 象素颜色放入置位/复位寄存器 */
asm mov ax,0f01h
asm out dx,ax          /* 设置允许置位/复位寄存器 */
asm mov cl,x           /* 计算象素在字节内的位置 */
asm and cl,7
asm mov ax,80h
asm sar ax,cl
asm mov ah,al
asm mov al,8
asm out dx,ax          /* 设置位屏蔽寄存器 */

/* —— 读写显示存储器的相应地址 —— */
asm mov al,es:[bx]
asm mov es:[bx],al

/* —— 恢复被修改过的各有关寄存器 —— */
asm mov ax,0ff08h
asm out dx,ax          /* 恢复位屏蔽寄存器 */
asm mov ax,01h
asm out dx,ax          /* 恢复允许置位/复位寄存器 */
asm mov ax,0f00h
asm out dx,ax          /* 恢复置位/复位寄存器 */
_VideoBusy = NO;
}

```

本函数的编译要稍稍复杂些。首先要将汇编语言编译器 masm.exe 或 tasm.exe 装入 tc 子目录下,然后再使用 Turbo C 的命令行编译器编译:

```
tcc -c -ml -G putpixel
```

如果编译无误的话,即可生成大模式目标文件 putpixel.obj。倘若需要生成其他模式的目标文件,可以修改上面的编译命令中的参数。然后可以使用 Turbo C 的库函数管理器将已经编译好的目标文件加入 HANENV 系统的函数库中(详见 5.2)。

那么这两个程序的速度究竟相差多少呢?我们编写了一个用于速度测试的小程序分别对它们进行了测试。

### 程序 6-3 测试某函数的运行时间。

我们编写的测试程序首先记下程序开始时的系统时间,然后调用函数 PutPixel 在屏幕上显示  $640 \times 480$  个蓝色像素点(清屏)。然后再记下结束时的系统时间并和程序开始时的系统时间进行比较。实际上,只要修改该程序中的调用函数部分即可测试其他函数的运行时间。

```
/* -----
   程序 TEST.C : 测试函数的执行时间
   ----- */
#include <dos.h>

#define BLUE      1

int _ScreenWidth  = 80;    /* 逻辑屏幕宽度(以字节为单位)    */
int _ScreenTop    = 0;    /* 当前屏幕首行位置          */

extern void _Cdecl _PutPixel(int x,int y,int color);

main()
{
    struct time t1,t2;
    int i,j;
    int clock;

    /* ----- 设置 EGA/VGA 的图形显示模式 12H ----- */
    _AH = 0;
    _AL = 0x12;
    geninterrupt(0x10);

    /* ----- 取开始运行的系统时间 ----- */
    gettimeofday(&t1);

    /* ----- 在显示屏幕上写 640×480 个像素点 ----- */
    for(i=0;i<480;i++)
        for(j=0;j<640;j++)
```

```

        _PutPixel(j,i,BLUE);

/* --- 取清屏后的系统时间 --- */
gettime(&t2);

/* --- 计算两个时间之差并显示 --- */
clock = (t2.ti_hund+t2.ti_sec*100+t2.ti_min*6000)
        -(t1.ti_hund+t1.ti_sec*100+t1.ti_min*6000);
printf("It cost %d, %d seconds.",clock/100,clock%100);
getch();

/* --- 恢复 EGA/VGA 的字符显示模式 03H --- */
_AH = 0;
_AL = 0x03;
geninterrupt(0x10);
}

```

在编译本程序时应使用 Turbo C 的命令行编译器进行编译:

```
tcc -ml -G test putpixel.obj
```

在 386/40 兼容机上分别测试上述两个写像素点的函数,其结果为完全使用 C 语言编写的 \_PutPixel 函数写满一屏像素点需时约 6.9 秒,而使用行间汇编编写的 \_PutPixel 函数只需要 5.8 秒,可见在算法相同的情况下,使用行间汇编编写的函数要比完全使用 C 语言编写的函数快约 14%。

## 6.2 显示一个字模点阵

有了写像素函数 \_PutPixel 就可以编写在显示屏幕上显示汉字的程序了。为了使我们编写的函数不仅可以显示不同大小(如  $16 \times 16$  或  $24 \times 24$  点阵)的汉字,而且可以显示西文字符和我们使用 HANENV 系统的造字工具生成的特殊尺寸的汉字、符号和图形,我们将所写字模点阵的尺寸、颜色和存放地址等都作为函数的参数。

**程序 6-4** 在显示屏幕上指定坐标处写一个点阵字模。

为了在显示屏幕上显示一个字模点阵,首先必须提供显示坐标和颜色、字模点阵的大小和存放位置。值得注意的是为了节约存储,在字模点阵中都是使用一个字节存放连续的 8 个像素,因此在利用 \_PutPixel 函数显示字模点阵时还应该将一个个像素从字节中分离出来。

```

/* -----
   函数 _DrawF : 在显示屏幕上指定坐标处写一个字模点阵
   ----- */

```

```

#include <dos.h>

extern void _Cdecl_PutPixel(int x,int y,int color);

void _Cdecl_DrawF(x,y,width,high,color,font)
int x,y;                /* 字模点阵的显示坐标 */
int width;              /* 字模点阵的宽度(以字节为单位) */
int high;               /* 字模点阵的高度(以像素为单位) */
int color;              /* 字模点阵的颜色 */
unsigned char *font;    /* 字模点阵的存放地址 */
{
    register unsigned char i,j,k; /* 循环控制变量 */

    for(i=0;i<high;i++) /* 显示字模点阵的一行 */
    {
        for(j=0;j<width;j++) /* 显示字模点阵的一个字节 */
            for(k=0;k<8;k++) /* 显示字模点阵的一个像素 */
                if((0x80>>k) & font[i*width+j])
                    _PutPixel(x+j*8+k,y+i,color); /* 写像素 */
    }
}

```

利用和程序 6-3 中类似的测试程序可以发现使用 DrawF 函数写满一屏汉字(25 行×40 列,共 1000 个汉字)所花费的时间比写满一屏像素点的时间要少得多,例如写满一屏“啊”字只需 2.36 秒左右。这当然是因为在写汉字时我们只写了有笔画处的像素的缘故。当然,这样作有一个缺点,就是如果汉字点阵下面的背景图案没有被清除,就可能影响到所显示的汉字的视觉效果。此时可以根据情形分别选用 HANENV 中的 Block 函数或 Bar 函数进行清屏,这两个函数的速度都要比直接用 PutPixel 函数清屏快得多。

那么,函数 DrawF 所显示的点阵字模是从哪里来的呢?我们设计 HANENV 系统时,考虑了以下几种情形:

一是使用数组直接提供字模点阵。这种方法适用于显示特殊图形标记的点阵、少量标题或封面用大点阵汉字或在整个程序仅使用少量汉字,因而可以将全部所需汉字直接装入内存的情况。特殊图形标记、大点阵汉字和小字库都可以使用 HANENV 系统提供的工具软件 hantool 制作并转换为 C 源程序(全局数组变量)。例如,我们可以利用 hantool 制作一个包含汉字“中国”的小字库,然后再将其转换为一全局数组变量(方法详见 5.3);

```

/* -----
   汉字点阵字模数组
   ----- */
unsigned char _Fonts[]=
{

```

```

/* —— “中”字的字模点阵 —— */
0x01,0,0x01,0,0x01,0x04,0x7F,0xFFFE,
0x41,0x04,0x41,0x04,0x41,0x04,0x41,0x04,
0x7F,0xFFFC,0x41,0x04,0x01,0,0x01,0,
0x01,0,0x01,0,0x01,0,0x01,0,

/* —— “国”字的字模点阵 —— */
0,0x04,0x7F,0xFFFE,0x40,0x04,0x40,0x24,
0x5F,0xFFF4,0x41,0x04,0x41,0x04,0x4F,0xFFE4,
0x41,0x04,0x41,0x44,0x41,0x24,0x5F,0xFFF4,
0x40,0x04,0x40,0x04,0x7F,0xFFFC,0x40,0x04
};

```

这样如果我们要在屏幕上坐标(100,100)处显示红色的“中国”两字就可以使用如下函数调用:

```

/* ——
使用 DrawF 在屏幕上显示汉字字模点阵
—— */
_DrawF(100,100,2,16,RED,_Fonts); /* 显示汉字“中” */
_DrawF(116,100,2,16,RED,_Fonts+32);/* 显示汉字“国” */

```

第二种情况是显示西文字符。因为 HANENV 系统工作在图形显示模式下,所以西文字符也应按处理汉字的方式才能显示。由于西文字符数量较少(包括扩展 ASCII 码在内不过 256 个),所以在 HANENV 系统中我们将其字模点阵已直接作为全局数组变量装入内存。考虑到不同应用场合的需要,我们设计了两套西文字符的字模点阵:一套是标准字模,包括所有 256 个 ASCII 码。考虑到应用扩展 ASCII 码中的制表符号时的纵向连续性,我们将西文字符的字模点阵的大小定为  $8 \times 18$ (即显示时没有行间距)。当然,为了统一起见,应该取所有的西文字符的字模点阵的大小均为相同尺寸,只不过其他字符的最下方两线都是空白。

这套标准西文字模存放在全局数组变量 `_ChrFont` 中,其中具体字符的地址可以使用下式计算:

字符地址 = `_ChrFont + (字符的 ASCII 码) * CHAR_HIGH`

其中符号常数 `CHAR_HIGH` 为定义于 `hanenv.h` 中的宏,其值为 18(参看第 14 章“HANENV 系统的头文件”)。

例如我们要在屏幕上坐标(100,100)处连续显示 10 个红色字母“A”,就可以使用如下函数调用:

```

/* ——
使用 DrawF 在屏幕上显示 ASCII 码字字模点阵
—— */

```

```

int i;
for(i=0;i<10;i++)
    _DrawF(100+i*8,100,1,CHAR_HIGH,RED,_ChrFont+'A'*CHAR_HIGH);

```

第二套西文字符的字模点阵的尺寸为  $6 \times 8$ , 适用于需要填写较小号字符的场合。这套字模只包括 ASCII 字符集中的可打印字符, 即其值应在 20H—7EH 之间。另外, 这套西文字符的字模中还包括了 5 个自定义符号, 即分别指向上、下、左和右的 4 个箭头和 1 个矩形棒。这些自定义符号可以使用 hanenv.h 中定义的符号常数调用:

```

/* -----
   扩充微型西文 ASCII 字符
   ----- */
#define UP_ARROW      127      /* 向上箭头      */
#define DOWN_ARROW    128      /* 向下箭头      */
#define RIGHT_ARROW    129     /* 向右箭头      */
#define LEFT_ARROW     130     /* 向左箭头      */
#define SMALL_BAR      131     /* 矩形块        */

```

这套小尺寸西文字符的点阵字模存放在全局数组变量 TinyFonts 中, 可参看随书 1 号软盘中 source 目录中的源程序 tinyfont.c。

#### 程序 6-5 向屏幕输出小西文字符串的函数。

在编程实践中, 我们发现多数屏幕输出都是以字符串为单位的。因此设计一个字符串输出函数可以大大提高编程效率。在 HANENV 系统中我们设计了一个用于输出由第二套西文字符的字模组成的西文字符串的函数 drawtiny:

```

/* -----
   函数 drawxytiny : 向屏幕输出小西文字符串
   ----- */
#include <dos.h>

extern unsigned char _TinyFonts[];

void _Cdecl drawxytiny(x,y,color,s)
int x;                /* 字符串显示列坐标(以像素为单位) */
int y;                /* 字符串显示行坐标(以像素为单位) */
int color;            /* 字符串显示颜色 */
char *s;              /* 字符串指针 */
{
    while(*s)
    {

```

```

    _DrawF(x,y,1,8,color,_TinyFonts+(( * s) - ' ') * 8);
    x += 6;
    s ++;
}
}

```

其实,在大多数情况下我们不能预先确定应用程序所使用的汉字集合,从而将其编辑为小字库。此时要求由标准汉字库提供所要显示汉字的字模点阵。但是标准汉字库所需存储量太大,不宜直接装入基本内存。因此我们采用如下变通方法解决这个问题:

我们在 HANENV 系统的初始化函数 `init_hanenv` 中定义了一个用于存放汉字字模点阵的全局数组变量 `_HanFont`,其大小为 1,152 字节,最大可以存放一个  $96 \times 96$  的字模点阵。同时设计了一个函数 `takefont` 用于将指定汉字从汉字库中拷贝到 `_HanFont` 中,然后即可利用 `_DrawF` 等函数将 `_HanFont` 中的点阵写到屏幕上。无论汉字库是在硬盘上还是已经装载到扩展内存中,`takefont` 函数均可从中读取汉字的字模点阵。该函数的使用方法为

```
int takefont(unsigned h);
```

参数 `h` 为汉字的机内码(两个字节的最高位均置 1 的国标码)。如果读汉字字模点阵成功,该函数返回非零值,否则返回零。该函数的设计和编程将在第 7 章中介绍。

#### 程序 6-6 显示汉字字符串。

类似函数 `drawtiny`,我们设计一个能显示字符串的函数。字符串中可以包括汉字和西文字符(ASCII 码符号和扩展 ASCII 码符号)。该函数的算法可以表示为

```

while(字符串中还有符号)
{
    取一符号送入变量 h 中;
    如果 h 中的符号是汉字,则
    {
        取汉字字模点阵;
        显示汉字;
    }
    否则如果 h 中的符号为 ASCII 码,则
        显示西文字符;
        根据字模宽度修改显示坐标;
}

```

其中从字符串中分离一个符号由函数 `takehan` 完成。该函数的调用方法为

```
unsigned char * takehan(unsigned char * string,unsigned * h);
```

该函数由给定的字符串首部读取一个符号(汉字或单字节字符)存入变量 `h` 中,然后返回指向该字符串剩余部分的指针。其源程序为



```

/* -----
   函数 takehan : 从字符串中分离出一个汉字或字符
   ----- */
#include <stdio.h>

unsigned char *_Cdecl takehan(string,h)
unsigned char * string;          /* 字符串指针          */
unsigned * h;                    /* 从字符串中分离出的符号 */
{
    if((*h = *string++) == 0)    /* 字符串中已无符号      */
        return NULL;
    else if(*h > 0xa0 && *string > 0xa0) /* 字符串的当前符号为汉字 */
        *h |= (*string++) << 8; /* h 中为汉字机内码      */
    return string;              /* 返回剩余字符串        */
}

```

显示汉字字符串的函数 drawxys 可以设计为

```

/* -----
   函数 drawxys : 显示汉字字符串
   ----- */
#include <hanenv.h>

void _Cdecl drawxys(x,y,color,s)
int x,y;                        /* 字符串的坐标          */
int color;                      /* 字符串的颜色          */
char * s;                      /* 字符串指针            */
{
    int width = _CurrentHZK->fontwidth; /* 汉字字模宽          */
    int high = _CurrentHZK->fonthigh; /* 汉字字模高          */
    unsigned h;
    while((s=takehan(s,&h)) != NULL) /* 分解字符串          */
    {
        if(IsSP(h) && takefont(h)) /* h 为汉字机内码且读字模成功 */
        {
            _DrawF(x,y,width,high,color,_HanFont);
            x += width * 8;
        }
        else if(h < 256) /* h 为西文字符或扩展 ASCII 码 */
        {
            _DrawF(x,y,1,CHAR_HIGH,color,_ChrFont+h*CHAR_HIGH);
            x += 8;
        }
    }
}

```

```

    }
}

```

程序中的 `_CurrentHZK` 是一个结构类型的全局变量,存放当前正在使用的汉字库的有关信息,其结构将在第7章中介绍;而 `isSP` 是用于判断符号是否汉字的函数,其使用方法为

```
int isSP(unsigned h);
```

该函数(带参宏)的定义可在 HANENV 系统的头文件 `hanenv.h` 中找到(见第14章:“HANENV 系统的头文件”)。

### 6.3 以更快的速度显示汉字

使用 `draw...` 族函数显示汉字和字符已能满足大多数应用程序对速度的要求,但是还有某些需要大量、连续地显示汉字或字符的应用场合如编辑软件、字处理软件或在线帮助等,对汉字显示的速度要求很高。因此,我们根据 EGA/VGA 图形显示模式 12H 的特点:即对屏幕操作的基本单位并不是像素,而是由 8 个毗邻像素构成的字节,另外设计了一套快速显示汉字或字符的函数。

使用这种方法对汉字的显示位置有一定的要求:显示汉字的横坐标必需取 8 的整倍数。但是我们发现在同屏大量显示汉字时往往采用类似字符显示方式来组织版面的结构,即在全屏幕或某矩形屏幕区域中汉字或字符汉字按行、按列整齐排放,这时横坐标限制对版面的设计影响甚小。

#### 程序 6-7 快速显示汉字点阵。

我们采用和 `_PutPixel` 相同的原理来实现快速显示汉字点阵,只是这次我们一次就同时向屏幕上写 8 个像素。后面我们可以看到,由于选择了高效率的算法,即使不用汇编语言编写程序也可以达到很高的显示速度。

```

/* -----
   函数 _OutF : 快速显示汉字点阵
   ----- */
#include <hanenv.h>

void _Cdecl _OutF(col,line,fontwidth,fonthigh,color,font)
int col;           /* 汉字显示的列坐标(以字节为单位) */
int line;          /* 汉字显示的列坐标(以像素为单位) */
int fontwidth;     /* 汉字字模点阵宽度(以字节为单位) */
int fonthigh;      /* 汉字字模点阵高度(以像素为单位) */
int color;         /* 汉字显示的颜色 */
unsigned char *font;
{
    /* --- 将像素坐标换算为显示存储器 VRAM 中的地址 --- */

```

```

char far * addr;
int i,j;                /* 循环控制变量 */
_VideoBusy = YES;
addr = MK_FP(0xa000,(line+_ScreenTop)*_ScreenWidth+col);

/* -- 设置 EGA/VGA 的寄存器 ----- */
outportb(0x3ce,0);      /* 象素颜色放入置位/复位寄存器 */
outportb(0x3cf,color);
outportb(0x3ce,1);      /* 设置允许置位/复位寄存器 */
outportb(0x3cf,0x0f);

/* ----- 循环:将字模点阵中的字节写到屏幕 ----- */
for(i=0;i<fonthigh;i++)
{
    for(j=0;j<fontwidth;j++)
    {
        outportb(0x3ce,0x08);    /* 设置位屏蔽寄存器 */
        outportb(0x3cf,*font++);
        *(addr+j) &= 0xff;        /* 读写 VRAM 的相应地址 */
    }
    addr += _ScreenWidth;        /* 修改显示地址 */
}

/* ----- 恢复被修改的寄存器 ----- */
outportb(0x3ce,8);          /* 恢复位屏蔽寄存器 */
outportb(0x3cf,0xff);       /* 恢复位屏蔽寄存器 */
outportb(0x3ce,1);          /* 恢复允许置位/复位寄存器 */
outportb(0x3cf,0);          /* 恢复允许置位/复位寄存器 */
outportb(0x3ce,0);          /* 恢复置位/复位寄存器 */
outportb(0x3cf,0x0f);       /* 恢复置位/复位寄存器 */
_VideoBusy = NO;
}

```

那么,使用\_OutF 比使用\_DrawF 究竟能快多少呢? 我们使用一个类似于测试写象素时用的测试程序,测出用\_OutF 函数写满一屏汉字(25 行×40 汉字)所花费的时间只有 0.22—0.27 秒左右,几乎要比使用\_DrawF 快 10 倍。

## 6.4 汉字点阵的放大

有了汉字或字符的字模点阵以后,还可以在显示的同时对其进行放大。HANENV 系统提供了两个放大显示字模点阵的函数\_DrawFont 和\_OutFont,分别使用逐点显示法和字节显示法。为减少函数的参数和简化函数的调用过程,我们在函数 init\_hanenv 中定义了两个

全局变量 `_Xtimes` 和 `_Ytimes`, 用于存放字模点阵的横向放大倍数和纵向放大倍数, 其初值均为 1。

#### 程序 6-8 使用逐点显示法放大显示汉字或字符的字模点阵。

使用逐点显示法放大显示汉字或字符的字模点阵比较简单, 只是将原来的一个点放大为一个小矩形, 这可以使用函数 `_Bar` 完成。另外, 一般汉字系统中的 24 点阵字库中的字模是专为打印设计的, 其像素排列方式为竖排 (HANENV 系统所配的所有高点阵字库, 包括 24 点阵字库在内均为横排)。为了直接使用这种竖排的汉字库, 我们设计使用一个全局变量 `_FontDirection` 控制显示字模点阵时的方向。

在头文件 `hanenv.h` 中定义了两个符号常数 (宏) 用于说明显示字模时的方向:

`HORIZONTAL` : 横向显示字模点阵

`VERTICAL` : 纵向显示字模点阵

在菜单应用中需要用虚线体显示被屏蔽的按钮的键名, 为了实现该功能, 我们设计使用一个全局变量 `_FontBkStyle` 控制显示字模点阵时的背景风格。

在头文件 `hanenv.h` 中定义了两个符号常数 (宏) 用于说明显示字模点阵时的背景风格:

`SOLIDLINE` : 用实线显示字模点阵

`DOTTEDLINE` : 用虚线显示字模点阵

```
/* -----
   函数 _DrawFont : 用逐点显示法放大显示字模点阵
   ----- */
#pragma inline
#include <hanenv.h>

extern int _ScreenWidth;          /* 逻辑屏幕宽度(以字节为单位) */
extern int _ScreenTop;           /* 当前屏幕首行位置 */
extern int _Xtimes;              /* 横向放大倍数 */
extern int _Ytimes;              /* 纵向放大倍数 */

int _FontBkStyle = SOLIDLINE;
int _FontDirection = HORIZONTAL;

extern void _Cdecl _Bar(int x,int y,int w,int h,int color,unsigned long pattern);

void _Cdecl _DrawFont(x,y,width,high,color,font)
int x,y;                          /* 字模点阵的显示坐标 */
int width;                        /* 字模点阵的宽度(以字节为单位) */
int high;                         /* 字模点阵的高度(以像素为单位) */
int color;                        /* 字模点阵的颜色 */
unsigned char *font;              /* 字模点阵的存放地址 */
{
```

```

int i,j,k;                                /* 循环控制变量 */

if(_FontBkStyle)                          /* 如果需要用虚线显示字模 */
{
    unsigned char *ptr = font;

    _CL = 0xaa;                            /* 0xaa : 10101010B, 虚线模板 */
    for(i=0;i<high;i++)
    {
        for(j=0;j<width;j++)
            *(ptr++) &= _CL;              /* 字模点阵用虚线模板处理 */
        asm rol cl,1;                     /* 换行则模板循环移位 */
    }
}
for(i=0;i<high;i++)                      /* 对点阵高度循环 */
    for(j=0;j<width;j++)                 /* 对点阵宽度(字节数)循环 */
        for(k=0;k<8;k++)                 /* 对点阵宽度(字节内)循环 */
            if((0x80>>k) & font[i*width+j])
            {
                if(_FontDirection)
                    _Bar(x+i*_Ytimes,y+(j*8+k)*_Xtimes,_Xtimes,_Ytimes,color,
                        0xffff0000);
                else
                    _Bar(x+(j*8+k)*_Xtimes,y+i*_Ytimes,_Xtimes,_Ytimes,color,
                        0xffff0000);
            }
}

```

由于程序复杂了,其运行时间也会相应增加。利用 DrawFont 写满一屏汉字“啊”(取横向放大倍数和纵向放大倍数均为1)需时 5.10 秒左右,相当于使用 DrawF 函数(2.36 秒)时的两倍左右。

同样,我们也可以利用和编写 OutF 相同的原理编写一个放大显示字模点阵的函数 OutFont。由于同时处理 8 个像素,所以显示速度肯定快于 DrawFont 函数。

#### 程序 6-9 使用字节显示法放大显示汉字或字符的字模点阵。

使用用字节显示法放大显示字模点阵要稍微复杂些。在从字模点阵中取出一个字节后,要根据横向放大倍数将其放大为若干字节,然后才可以写入显示存储器 VRAM 中。为了编程简单起见,我们限制横向放大倍数不得超过 4,因为实际上放大倍数太大时所显示的汉字轮廓会出现较严重的“锯齿”现象,影响屏幕效果。

```
/* -----
```

函数 \_OutFont : 用字节显示法放大显示字模点阵

```

----- */
#include <hanenv.h>

void _Cdecl _OutFont(col,line,fontwidth,fontheigh,color,font)
int col;                /* 汉字显示的列坐标(以字节为单位) */
int line;               /* 汉字显示的列坐标(以像素为单位) */
int fontwidth;          /* 汉字字模点阵宽度(以字节为单位) */
int fontheigh;          /* 汉字字模点阵高度(以像素为单位) */
int color;              /* 汉字显示的颜色 */
unsigned char *font;
{
    /* ----- 将像素坐标换算为显示存储器 VRAM 中的地址 ----- */
    char far *addr;
    int i,j,x,y;         /* 循环控制变量 */
    unsigned char mark;   /* 标记:字节中对应位是否应写像素 */
    unsigned long buff;   /* 4 字节变量:放大后的字节 */
    unsigned long mode;   /* 4 字节变量:和 mark 配合生成 buff */

    if(_Xtimes>4)         /* 横向放大倍数不应超过 4 */
        _Xtimes = 4;

    _VideoBusy = YES;
    addr = MK_FP(0xa000,(line+_ScreenTop)*_ScreenWidth+col);

    /* ----- 设置 EGA/VGA 的寄存器 ----- */
    outportb(0x3ce,0);    /* 像素颜色放入置位/复位寄存器 */
    outportb(0x3cf,color);
    outportb(0x3ce,1);    /* 设置允许置位/复位寄存器 */
    outportb(0x3cf,0x0f);

    /* ----- 循环:将字模中的字节放大后写屏幕 ----- */
    for(i=0;i<fontheigh;i++) /* 对点阵高度循环 */
    {
        for(y=0;y<_Ytimes;y++) /* 对纵向放大倍数循环 */
        {
            for(j=0;j<fontwidth;j++) /* 对点阵宽(字节)循环 */
            {
                /* ----- 将点阵中的字节放大后送入 buff ----- */
                buff = 0L;
                if(*font)
                {
                    mark = 0x01;

```

```

mode = 15 >> (4 - _Xtimes);
for(x=0;x<8;x++) /* 对每个像素循环 */
{
    if( mark & (*font))
        buff |= mode;
    mark <<= 1;
    mode <<= _Xtimes;
}
}

/* ----- 将放大后的字节写入 VRAM ----- */
for(x=0;x<_Xtimes;x++) /* 对横向放大倍数循环 */
{
    outportb(0x3ce,0x08); /* 设置位屏蔽寄存器 */
    outportb(0x3cf, *((unsigned char *)(&buff)+_Xtimes-x-1));
    *(addr+j*_Xtimes+x) &= 0xff; /* 读写 VRAM 的相应地址 */
}
font++;
}
addr += _ScreenWidth; /* 修改显示地址 */
font -= fontwidth;
}
font += fontwidth;
}

/* -- 恢复被修改的寄存器 ----- */
outportb(0x3ce,8); /* 恢复位屏蔽寄存器 */
outportb(0x3cf,0xff); /* 恢复位屏蔽寄存器 */
outportb(0x3ce,1); /* 恢复允许置位/复位寄存器 */
outportb(0x3cf,0); /* 恢复允许置位/复位寄存器 */
outportb(0x3ce,0); /* 恢复置位/复位寄存器 */
outportb(0x3cf,0x0f); /* 恢复置位/复位寄存器 */
_VideoBusy = NO;
}

```

经过实测,用 OutFont 写一屏放大倍数为 1 的汉字需时 1.21 秒,大大快于使用 DrawFont 函数。

在 HANENV 中,要显示较大的汉字有两种办法:一是使用有放大功能的显示函数,一是使用大的字模点阵。在存储空间允许的情况下,我们推荐使用大字模点阵。这样不仅显示的汉字质量好(笔画轮廓不会出现“锯齿”状台阶),而且由前面各程序的测试结果可以看出,速度也快得多。只有在存储空间特别紧张或者要显示特大汉字(超过 96×96 点阵),以及不能使用小字库时才有必要选用带放大功能的显示函数。

## 6.5 HANENV 系统中的汉字(字符)显示函数族

在应用程序的编程时通常总是选定一种尺寸的汉字字模作为基础字模,用来显示菜单、提示信息或说明等中间的汉字信息。在这种情况下,可以简化汉字显示函数的设计。为了便于使用,在 HANENV 系统中从 DrawF、DrawFont、OutF 和 OutFont 派生出了一个汉字或字符显示函数族,用于输出当前汉字库中的汉字和标准西文 ASCII 字符以及由它们组成的字符串。下面我们简单介绍一下这组函数的功能和调用格式,而更详细的使用说明可以参看第 15 章:“HANENV 系统的库函数”。

1. 在屏幕指定坐标处显示一个西文 ASCII 码字符:

```
void drawxyc (int x,int y,int color,int ch);
void outxyc (int col,int line,int color,int ch);
void putxyc (int col,int line,int color,int bk,int ch);
void drawxychar (int x,int y,int color,int ch);
void outxychar (int col,int line,int color,int ch);
void putxychar (int col,int line,int color,int bk,int ch);
void drawxyt (int x,int y,int color,int ch);
```

2. 在屏幕指定坐标处显示一个当前汉字库中的汉字:

```
void drawxyh (int x,int y,int color,unsigned han);
void outxyh (int col,int line,int color,unsigned han);
void putxyh (int col,int line,int color,int bk,unsigned han);
void drawxyhan (int x,int y,int color,unsigned han);
void outxyhan (int col,int line,int color,unsigned han);
void putxyhan (int col,int line,int color,int bk,unsigned han);
```

3. 在屏幕指定坐标处显示一个字符串:

```
void drawxys (int x,int y,int color,char *s);
void outxys (int col,int line,int color,char *string);
void putxys (int col,int line,int color,int bk,char *string);
void drawxystr (int x,int y,int color,char *string);
void outxystr (int col,int line,int color,char *string);
void putxystr (int col,int line,int color,int bk,char *string);
void drawxytiny (int x,int y,int color,char *string);
```

以上三组函数的具体功能和其函数命名有关:draw…表示该函数使用逐点法输出点阵;…c、…h 和…s 表示显示时不放大;…char、…han 和…str 表示函数具有放大能力,其放大倍数由全局变量 Xtimes 和 Ytimes 指定;out…表示显示前不清屏,而 put…表示显示前先以参数 bk 指定的颜色清显示区的背景。

4. 正文方式的显示函数。这组函数的共同特点是由全局变量指定显示的位置和颜色。在第 9 章中我们还要详细介绍这种工作方式。

```
void outc (int ch);
```



---

```
void outchar      (int ch);  
void outh        (unsigned han);  
void outhan      (unsigned han);  
void outhl       (unsigned han,int whichhalf);  
void outhanl     (unsigned han,int whichhalf);  
void outs        (char * string);  
void outstr      (char * string);  
void putnstr     (char * string,int width);
```

以上这些函数有些是宏,其定义可以在第14章“HANENV系统的头文件”中找到。其他函数的源程序中有一部分已在本书的有关章节中做了介绍。

## 汉字库的组织

由第 6 章的介绍可以看出,为了丰富 HANENV 系统的屏幕显示能力,我们设计了 4 个基本的字模显示函数 DrawF、\_DrawFont、\_OutF 以及 \_OutFont,它们均可输出不同大小的字模点阵。

相对于西文字符的显示而言,显示汉字有一个明显的问题:汉字的结构复杂、数量众多,因而汉字库的规模一般都相当大。以常用的 16×16 点阵的国标汉字库为例,每个汉字的字模点阵需要占用 32 字节,而整个字库即需占存储 256K 之多;而 24×24 点阵的单个汉字即需占用 72 字节,整个汉字库的大小达 580K 以上,因此像处理西文字符的字模点阵那样将整个汉字库直接装入常规内存是不现实的。所以我们将汉字的操作分为两步实现,即首先将汉字的字模点阵由汉字库中装入显示缓冲区 \_HanFont,然后再进行显示。这样,汉字库就可以放在硬盘上,或者装入扩充存储器中了。

### 7.1 HANENV 的汉字库结构

为了灵活地进行汉字显示工作,我们定义了一个汉字库结构类型 HZK,用来表示有关某个汉字库的具体属性。

```

/* -----
1. 有关汉字库的数据类型定义、全局变量和函数说明
----- */
typedef struct                                /* 汉字库类型 */
{
    int fontwidth;                          /* 点阵宽(以字节为单位) */
    int fonthigh;                           /* 点阵高(以像素为单位) */
    unsigned char * codelist;               /* 指向小字库索引表的指针 */
    int wherefont;                          /* 字库装载位置 */
    int fontcount;                          /* 字库中的字模数量 */
    FILE * fontfile;                       /* 字库文件名 */
}

```

```

unsigned long EMM_addr;          /* 字库在扩充内存中的地址      */
int fonthandle;                  /* 字库在 XMS 扩充内存中的句柄 */
unsigned char *fontbuff;         /* 字库区在常规内存中的指针    */
}HZK;

```

说明:

1. codelist 是一字符类型的指针变量,如果被装载的字库是国标字库,则该指针应被赋 0 值,如果被装载的字库是经过编辑的小字库,则该指针指向小字库的索引表。小字库的索引表是一个字符串,由小字库中的汉字的机内码组成,各汉字按其在小字库中出现的顺序排列。在小字库中不应出现重复的汉字。

2. 字库装载位置可以使用宏定义:

```

#define DSK      0          /* 汉字库在硬盘中          */
#define EMM      1          /* 汉字库在扩充内存中      */
#define XMS      2          /* 汉字库在 XMS 扩充内存中 */
#define MEM      3          /* 小字库在内存中          */

```

3. 如果使用我们提供的国标字库,则字模数量可按式计算:

字库中的字模数量 = 94(位) × 87(区) = 8 178

如果使用小字库,则该项就是小字库中的字模数量。该项目由函数 load\_HZK 自动填写。

4. EMM\_addr(字库在扩充内存中的地址)、fonthandle(字库在 XMS 扩充内存中的句柄)和 fontbuff(字库区在常规内存中的指针)三项由汉字库装载函数 load\_HZK 根据情况自动填写。

我们还定义了一个 HZK 类型的全局变量 \_CurrentHZK,用来存放当前汉字库的有关参数:

```

extern HZK *_CurrentHZK;      /* 当前汉字库结构变量的指针 */

```

为了在同一个应用程序中装载若干个不同的汉字库(点阵大小或字体不同),可以分别设置几个 HZK 类型的指针型变量。但只有全局指针变量 \_CurrentHZK 指向的汉字库可以直接读取。因此在需要显示其他字库中的点阵时,可以通过交换字库指针变量的值来解决。

下面我们首先介绍汉字库的装载函数 load\_HZK 和卸载函数 free\_HZK 的设计与实现。

#### 程序 7-1 装载汉字库。

在设计汉字库装载函数 load\_HZK 时最重要的问题是将汉字库放在什么地方。如前所述,国标汉字库体积太大,不宜放在常规内存中;虽然将其放在硬盘上处理起来最简单,但

速度也最慢。最好的方法是将汉字库装入扩充内存,因其容量大,速度快,通常也较少为应用程序的其他部分所占用。因此我们在设计该函数时提供了一个说明字库装载位置的参数 wherefont,可以取值 XMS、EMM、DSK 和 MEM,分别表示汉字库装载于扩充内存中(分别使用 XMS 规范或 BIOS 的中断 INT15H 进行存取)、硬盘上或常规内存中。只有经过编辑的小字库(长度小于 64K)才能直接装载于常规内存中。

```

/* -----
   函数 load_HZK : 装载汉字库
   ----- */

#include <hanenv.h>
#include <fontl.h>
#include <alloc.h>

HZK * _Cdecl load_HZK(width,high,hzkname,codelist,wherefont)
int          width;          /* 点阵宽(以字节为单位) */
int          high;           /* 点阵高(以像素为单位) */
char         * hzkname;       /* 字库文件名 */
unsigned char * codelist;     /* 小字库索引表指针 */
int          wherefont;      /* 字库装载位置 */
{
    HZK * hzk;                /* 指向汉字库结构变量的指针 */
    long hzksize;              /* 汉字库文件长度 */

    /* ----- 为汉字库结构信息申请存储 ----- */
    if((hzk=(HZK *)malloc(sizeof(HZK)))==NULL)
        return NULL;

    /* ----- 打开汉字库文件 ----- */
    if((hzk->fontfile=fopen(hzkname,"r+b"))==NULL)
    {
        free(hzk);
        return NULL;
    }

    /* ----- 计算汉字库长度 ----- */
    fseek(hzk->fontfile,0L,SEEK_END);
    hzksize = ftell(hzk->fontfile);
    fseek(hzk->fontfile,0L,SEEK_SET);

    /* ----- 填写汉字库参数 ----- */
    hzk->fontwidth = width;
    hzk->fonthigh  = high;

```

```

hzk->fontcount = hzksize/(width * high);
hzk->codelist   = codelist;
hzk->wherefont = wherefont;

/* ----- 装载汉字库 ----- */
switch(wherefont)
{
case XMS: /* 将汉字库装载到扩充内存中(用 XMS 规范) */
    if((hzk->fonthandle = _LoadXMS(hzk->fontfile, hzksize)) == 0)
    {
        free(hzk);
        return NULL;
    }
    break;
case EMM: /* 将汉字库装载到扩充内存中(用 INT15H) */
    if((hzk->EMM_addr = _LoadEMM(hzk->fontfile, hzksize)) == 0L)
    {
        free(hzk);
        return NULL;
    }
    break;
case MEM: /* 将汉字库装载到常规内存中(限小字库) */
    if((hzk->fontbuff = malloc(hzksize)) == NULL)
    {
        free(hzk);
        return NULL;
    }
    fread(hzk->fontbuff, hzksize, 1, hzk->fontfile);
    break;
default: /* 汉字库留在硬盘上 */
    break;
}

/* -- 返回汉字库结构变量的地址 ----- */
return hzk;
}

```

在装载小字库时会遇到一个问题：如何处理小字库的索引。在 5.3 节中我们介绍过，HANENV 的字库管理工具 hantool 生成的小字库均带有一个索引文件，而 load\_HZK 函数要求一个字符串数组作为小字库的索引。这样做的原因是小字库的规模通常不大（在几十至二三百字之间），索引的尺寸很小（每个汉字两个字节），将其直接放在内存中可以省去索引文件，使得应用程序的结构比较简单。因此在编写用到小字库的应用程序时可以通过编辑软件的读文件功能将小字库的索引直接加入源程序中，改编成全局数组变量的初值，例如某

应用程序(交通事故管理)用于菜单显示的小字库的索引为

```
char * xzkidx =
```

“查询道路交通事故登记表一二数据录入月附报生成死亡情况分析季年输出  
维护系统初始化整理索引各库修改代码提示字典文件菜单增加”;

而 HANENV 系统的计算器函数 calculor 之中用的小字库索引为

```
char * xzkidx = "789+←C456-(π123×)H0.,/=数据溢出表达式为空缺项少右括  
号引运算符连续类型错负平方根求对函名误过多值”;
```

### 程序 7-2 卸载汉字库。

如果在应用程序中某个汉字库已经不在使用,应该及时将其卸载,以释放被其占用的资源。

应该说明的 BIOS 的中断 INT15H 没有对扩充内存提供什么管理功能,各应用程序根据自己的需要随意使用扩充内存,因此在卸载汉字库时也不必考虑归还自己占用的那部分扩充内存。但是如果计算机上已经装有 XMS 或 EMS 的驱动程序,就不宜再使用 INT15H 随意地使用扩充内存了,因为这样做有可能破坏 XMS 的工作状态,引起其他程序工作失常。

```
/* -----  
函数 free_HZK : 释放汉字库  
----- */  
#include <hanenv.h>  
#include <alloc.h>  
  
HZK * _Cdecl free_HZK(HZK * hzk)  
{  
    if(hzk)  
    {  
        switch(hzk->wherefont)  
        {  
            case XMS:  
                _FreeXMS(hzk->fonthandle);  
                break;  
            case MEM:  
                free(hzk->fontbuff);  
        }  
        if(hzk->fontfile)  
            fclose(hzk->fontfile);  
        free(hzk);  
    }  
}
```

```

return NULL;
}

```

## 7.2 从汉字库中取字模点阵

要从汉字库中取字模点阵,首先必须求出汉字字模在字库中的位置。如果使用小字库,那么汉字在小字库中的位置要靠索引得出,其计算公式为

汉字点阵的字节数 = 点阵宽(以字节为单位)×点阵高(以象素为单位)

汉字在字库中的位置 = 汉字在索引中的位置×汉字点阵的字节数

如果使用 HANENV 系统提供的国标字库,则汉字在字库中的位置可以直接按下式由汉字的机内码求出:

区码 = 汉字机内码低字节 - 160

位码 = 汉字机内码高字节 - 160

汉字在字库中的位置 = ((区码 - 1) × 94 + 位码 - 1) × 汉字点阵的字节数

注意,如果使用其他汉字系统的现成汉字库,则上述公式可能还要进行修改。某些汉字系统的汉字库去掉了国标 GB2312-80《信息交换用汉字编码字符集》中 10-15 区的空白区,因此在利用此类汉字库时虽然计算 1-9 区的图形字符的地址时仍可使用上述公式,但在计算 16 区以后的汉字时则要在区码中再减去 6。区分这种字库可以看其文件长度:以 16×16 点阵的汉字库为例,这种字库的长度约为 243K,而我们提供的国标字库的长度约为 256K。另外, HANENV 系统的高点阵汉字库(24×24 点阵以上)中的字模结构与一般汉字系统中用于打印输出的字模的结构不同,不能直接使用后者代替前者(参看 7.3 和 6.4)。

### 程序 7-3 从汉字库中取字模点阵。

如果被装载的汉字库是经过编辑的小字库,那么首先应查找索引以确定汉字字模在字库中的位置。我们使用顺序查找法进行查找,在小字库中的汉字数量不多时,其查找效率相当高。这种方法的一个主要优点是不要求对索引进行排序。

和装载汉字库类似,读字模点阵时也要按汉字库的装载位置分别进行处理。

```

/* -----
   函数 takefont : 从汉字库中取字模点阵
   ----- */
#include <mem.h>
#include <hanenv.h>

int _Cdecl takefont(h)
unsigned h;          /* 汉字机内码 */
{
    int size = _CurrentHZK->fonthigh * _CurrentHZK->fontwidth;
    long pos, addr;

```

```

/* - 如果是小字库,则首先在索引中查找汉字在字库中的位置 - */
if(_CurrentHZK->odelist)
{
    for(pos=0;pos<_CurrentHZK->fontcount;pos++)
        if(h==*((unsigned *)_CurrentHZK->odelist+pos*2))
        {
            pos*=size;
            goto COPY;
        }
    return NO;
}
else
    pos=(long)(((h&0xff)-161)*94+(h>>8)-161)*size;
COPY:
switch(_CurrentHZK->wherefont)
{
    case DSK: /* 直接使用硬盘上的汉字库文件 */
        fseek(_CurrentHZK->fontfile,pos,SEEK_SET);
        fread(_HanFont,sizeof(char),size,_CurrentHZK->fontfile);
        break;
    case XMS: /* 汉字库已装载到扩充内存中(用 XMS 规范) */
        _Emb.len = size;
        _Emb.sour_han = _CurrentHZK->fonthandle;
        _Emb.sour_off = pos;
        _Emb.dest_han = 0;
        _Emb.dest_off = FP_SEG(_HanFont);
        _Emb.dest_off<<= 16;
        _Emb.dest_off += FP_OFF(_HanFont);
        _MoveDataXMS();
        break;
    case EMM: /* 汉字库已装载到扩充内存中(用 INT15H) */
        addr = FP_SEG(_HanFont);
        addr = (addr<<4)+FP_OFF(_HanFont);
        _SetDestAddr(addr,size);
        addr = _CurrentHZK->EMM_addr+pos;
        _SetSourAddr(addr,size);
        _MoveDataEMM(size);
        break;
    case MEM: /* 汉字库已装载到常规内存中(仅限小字库) */
        memcpy(_HanFont,_CurrentHZK->fontbuff+pos,size);
        break;
}

```



```

return YES;
}

```

### 7.3 HANENV 系统的字库及其生成

与本书一起发行的 HANENV 系统的软盘中已经包括了下列国标汉字库(其中 32 点阵和 40 点阵的各体字库在扩充盘中,未与本书配套出售,如果读者需要这些字库可以直接向出版社邮购):

HZK16J : 16 点阵简体字库	HZK32F : 32 点阵仿宋体字库
HZK16F : 16 点阵繁体字库	HZK32H : 32 点阵黑体字库
HZK24S : 24 点阵宋体字库	HZK32K : 32 点阵楷体字库
HZK24F : 24 点阵仿宋体字库	HZK40S : 40 点阵宋体字库
HZK24H : 24 点阵黑体字库	HZK40F : 40 点阵仿宋体字库
HZK24K : 24 点阵楷体字库	HZK40H : 40 点阵黑体字库
HZK32S : 32 点阵宋体字库	HZK40K : 40 点阵楷体字库

以上字库均包括国标 GB2312-80《信息交换用汉字编码字符集(基本集)》1-87 区的所有字符和汉字(10-15 区的空白也在内,供存放自造的扩充汉字或符号用)。这些字库中字模点阵均采用按行排列,如图 7-1 所示。

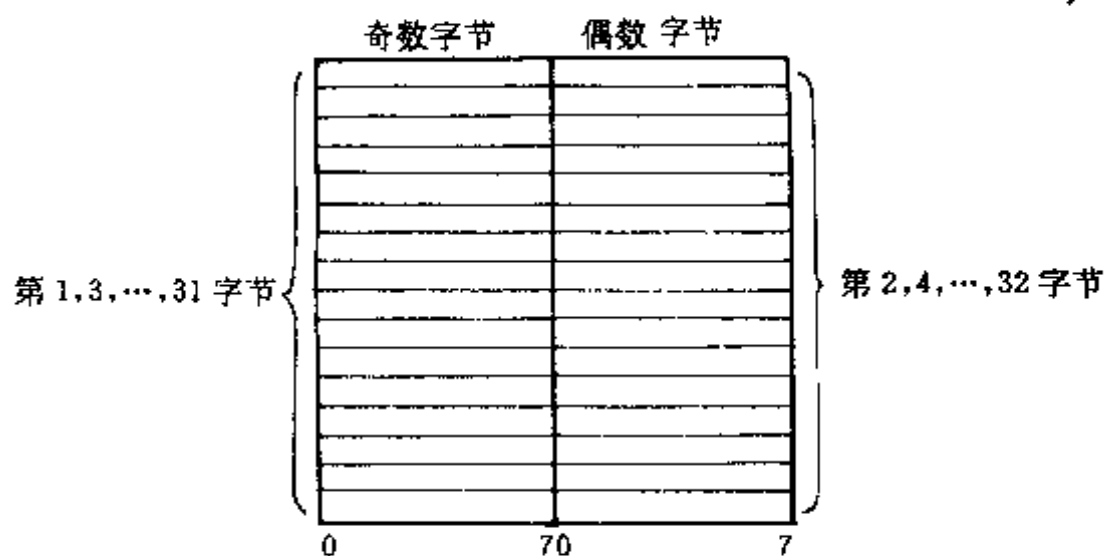


图 7-1 HANENV 中字模点阵的排列

应该说明的是一般汉字系统的高点阵字库都是为打印机设计的,所以其排列方法与图 7-1 不同。如果读者没有买到 HANENV 系统的软盘而又想按照本书介绍的方法构造自己的汉字处理模块,也可以对手头现有的汉字库进行改造。改造的一般步骤为。

1. 分析字库的结构。有的字库包括 1-87 区的所有字符和汉字,如 HANENV 系统的所有字库;也有的字库去掉了 10-15 区的空白部分;还有的汉字系统将 1-9 区的图形符号和 16-87 区的汉字分别存放在不同的字库中,特别是高点阵字库更是如此。在分析时应仔细阅读原汉字系统的说明书或 readme 文件,确定字库是否点阵字库(矢量字库较难转换),图形符号是否分开存放等。然后再查看字库文件的长度,与由表 7-1 中的数据进行比较。比较的结果可能有些误差,这是因为有些汉字系统自行扩展了 10 区的图形符号、去掉了 87 区尾

部的空白或增加了一些扩充余地等原因所致。一般来说,如果某字库的长度和表列数据之差可以被相应的字模大小除尽,则可基本判定其点阵规格正确,可以在以后进一步分析误差的原因。如果其差不能被相应的字模大小除尽,则说明预先假设的点阵规格有误。在判定如 $36\times 40$ 点阵(简称40点阵)这类非正方形字模点阵时最易出现这种情况。

表 7-1 汉字库的长度数据(表中数据均以字节为单位)

点阵	字模大小	1—9 区(字符)长度	10—15 区(空白)长度	16—87 区(汉字)长度
$16\times 16$	32	27,072	18,048	213,568
$24\times 24$	72	60,912	40,608	480,528
$32\times 32$	128	108,288	72,192	854,272
$36\times 40$	180	152,280	101,520	1,201,320
$40\times 40$	200	169,200	112,800	1,334,800
$48\times 48$	288	243,648	162,432	1,922,112

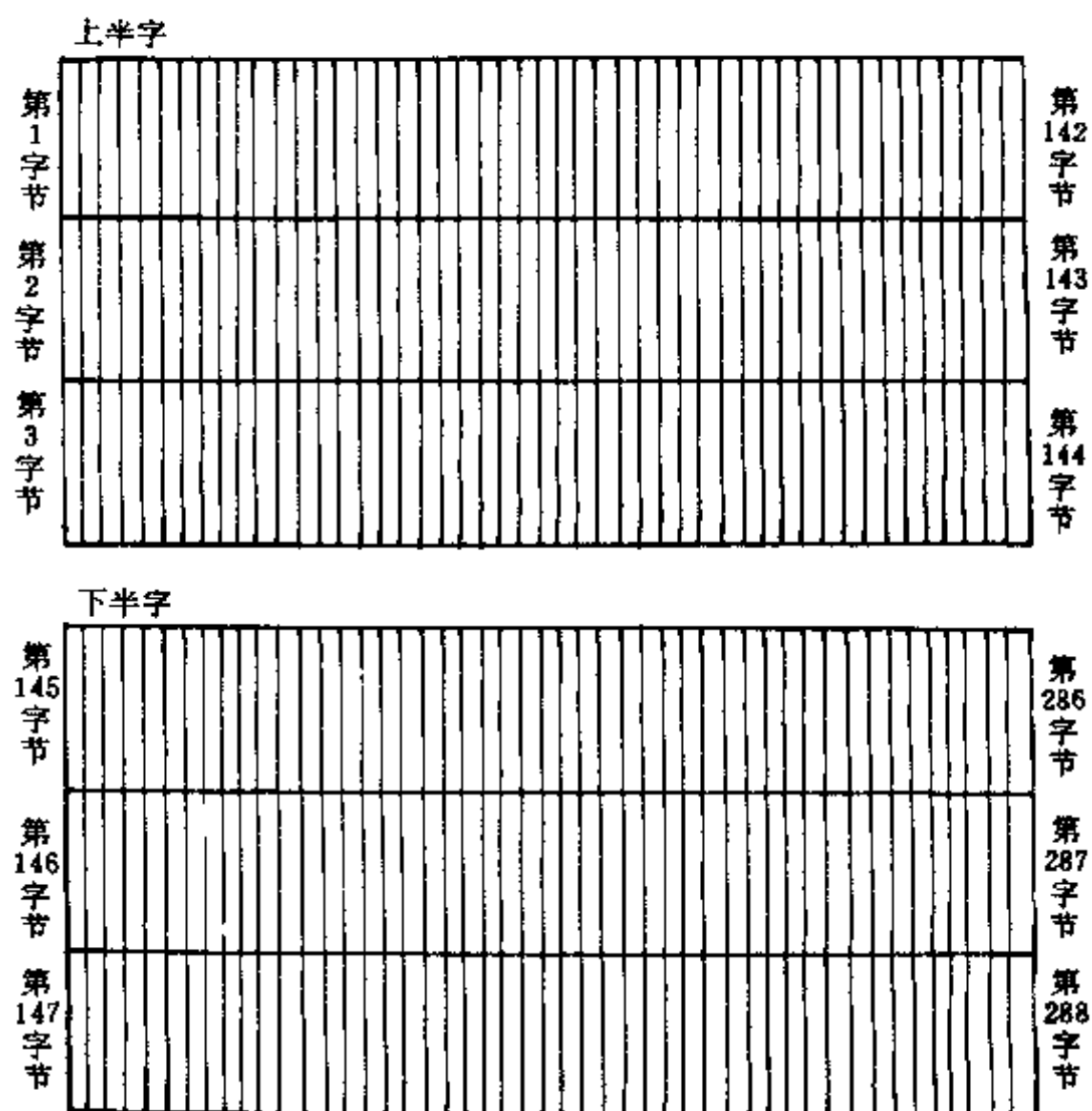


图 7-2 某 $48\times 48$ 点阵的字模排列方法

2. 确定字模中象素的排列方式。一般来说,各汉字系统的 $16\times 16$ 点阵作为显示字库,基本上都采用了横排的方式,可以直接使用 DrawF 等函数验证;而 $24\times 24$ 点阵以上的字模多是为打印而设计,其象素竖排最为常见。对于 $24\times 24$ 点阵,如果使用 DrawF 等函数显示则会发现汉字“倒”下来了。此时可以改用函数 DrawFont 显示,并将全局变量 FontDirection 设置为纵向显示(VERTICAL)进行验证。而对于更高点阵的字模,可能还有更加复杂的排列方法。例如,有某 $48\times 48$ 点阵的字模排列方法如图 7-2 所示。此时可以自己编一小段

程序对其字模中的字节重新排序后再使用函数 DrawFont 显示观察。

3. 在弄清了字模中像素的排列和字库的结构以后,即可编制一个转换程序将其转换为 HANENV 系统格式的汉字库了。由于各汉字系统中的字库情况千差万别,难以编写通用的转换程序,所以下面我们仅举一例进行说明。

#### 程序 7-4 转换 24 点阵打印字库为 HANENV 系统的高点阵显示字库。

设被转换的字库 hzk24.old 为 24×24 点阵,像素竖排,缺 10—15 区的空白。HANENV 系统要求字模中的像素横排,有 10—15 区的空白。

```

/* -----
   程序 tran24zk : 转换 24 点阵的打印字库为 HANENV 的显示字库
   ----- */
#include <stdio.h>
#include <mem.h>
#include <conio.h>

#define FONT_SIZE 72          /* 24 点阵占用字节数为 72 */

void tran24(unsigned char *font);

main()
{
    FILE *source = fopen("hzk24.old","rb");
    FILE *destin = fopen("hzk24.new","wb");
    unsigned char font[FONT_SIZE];
    int fontcount = 0;          /* 正在转换的字模数 */

    /* ----- 转换 1—9 区的图形符号 ----- */
    for(;fontcount<9*94;fontcount++)
    {
        fread (font,1,FONT_SIZE,source);
        tran24(font);
        fwrite(font,1,FONT_SIZE,destin);
    }

    /* ----- 生成 10—15 区的空白区 ----- */
    memset(font,0,FONT_SIZE);
    for(;fontcount<16*94;fontcount++)
        fwrite(font,1,FONT_SIZE,destin);

    /* ----- 转换 16—87 区的汉字 ----- */

```

```

for(;fontcount<86*94;fontcount++)
{
    fread(font,1,FONT_SIZE,source);
    tran24(font);
    fwrite(font,1,FONT_SIZE,destin);
}
fclose(source);
fclose(destin);
}

/* -----
   函数 tran24 : 转换 24×24 点阵的竖排字模为横排字模
   ----- */

#define FONT_WIDTH    3
#define FONT_HIGH     24

void tran24(unsigned char *font)
{
    unsigned char tmp[FONT_HIGH][24];
    int i,j,k;

    /* -- 将点阵数据展开为 24×24 的数组 ----- */
    for(i=0;i<FONT_HIGH;i++)
        for(j=0;j<FONT_WIDTH;j++)
            for(k=0;k<8;k++) /* 显示字模点阵的一个像素 */
            {
                tmp[i][j*8+k] = 0;
                if((0x80>>k) & font[i*FONT_WIDTH+j])
                    tmp[i][j*8+k] = 1;
            }

    /* -- 将数组转向后再装入字模缓冲区 ----- */
    memset(font,0,FONT_SIZE);
    for(i=0;i<FONT_HIGH;i++)
    {
        for(j=0;j<FONT_WIDTH;j++)
            for(k=0;k<8;k++)
            {
                if(tmp[j*8+k][i])
                {
                    font[i*FONT_WIDTH+j] |= 0x80>>k;
                    printf("\333\333");
                }
            }
    }
}

```

```
        else
            printf(" ");
    }
    printf("\n");
}
```

上面 tran24 函数中的 printf 函数用于在屏幕上显示转换后的汉字点阵。由于显示的速度相当慢,所以转换一个字库可能要相当长的时间。所以也可以在调试时使用上述程序,调试完毕后再将函数 tran24 中所有的 printf 语句都删去,这样可以大大加快程序的运行速度。

## 汉字输入模块的设计

由于计算机的键盘是由使用拼音文字的西方人发明的, 用其输入西方文字中的字母和符号既方便, 速度也很快。而中文所用的字符——汉字的结构和西文的字符完全不同, 所以直接使用西文键盘输入汉字是有困难的。以往开发的专用中文输入设备价格昂贵又不实用, 所以目前绝大多数中文操作系统都采用利用标准键盘输入汉字编码的方法解决汉字的输入问题。

所谓编码方法就是使用一定长度的西文字符序列来表示某个汉字。具体的表示方案很多, 据统计已达上千种之多, 但常用的不过十余种而已。一般说来, 好的汉字输入编码方案要满足两个条件: 一是要易记易学, 一是要能快速输入。但是要同时满足以上两个条件谈何容易。全拼拼音码最为易学, 尤其是青年人, 几乎都在学校学过汉语拼音。但由于汉语不算四声, 仅有近 400 个音节, 而常用汉字就有六七千个, 所以同音字很多, 造成拼音码的重码率很高, 影响了输入速度。国标区位码无重码, 码长固定, 但由于编码和字音、字形之间都没有直接的联系, 几乎无人能做到将全部汉字的区位码都记住。因此才出现了像五笔字型等既考虑了输入速度, 编码规则又相对简单的输入方法。

由于目前尚缺乏一种专业、业余通用, 学习简单而输入速度又高的权威汉字输入方法, 所以大多数汉字系统都配有若干套不同的输入方法供用户选择。HANENV 系统的汉字输入由一个基本模块和几个输入法模块组成, 并为应用程序的开发人员留有扩充新输入方法的接口。

### 8.1 汉字输入函数 gethan

为了满足不同用户的需要, 我们提供了若干种汉字输入法可供选择。在 HANENV 系统中我们定义了一个汉字输入法模块结构类型, 用以表示各汉字输入法的参数:

```
/* -----  
    汉字输入法模块的类型定义  
----- */
```

```

typedef struct
{
    char    modename[9];          /* 汉字输入法名称          */
    int     maxcodes;             /* 码表长度                */
    char    * codeset;            /* 输入法所用按键集合      */
    int     wildchar;             /* 通用替代键              */
    int     haveload;             /* 该汉字输入法是否已经装载 */
    unsigned (* get_han)();       /* 汉字输入法模块指针      */
    int     wherecodes;           /* 码表位置                */
    FILE    * codefile;          /* 码表文件                */
    long    filefront;           /* 码表文件头指针          */
    long    filelen;             /* 码表文件长度            */
    int     handle;              /* 码表在 XMS 扩充内存的句柄 */
    unsigned long EMM_addr;       /* 码表在扩充内存中的地址  */
}MODE_TYPE;

```

和大多数汉字系统的习惯相同,我们采用复合功能键 Alt+Fn 切换各种汉字输入法。因此,我们定义了一个 MODE\_TYPE 类型的全局数组变量,共有 10 个元素,分别用来存放定义于 Alt\_F1 到 Alt\_F10 上的各个汉字输入法:

```
extern MODE_TYPE _Mode[];
```

提示行是汉字输入模块的重要组成部分。在设计汉字输入模块时我们提供了两种风格的提示行,一种是仿 WINDOWS 界面风格的,提示行上的各种信息均在提示窗口中显示;另一种是传统中文操作系统风格的提示行,形式较简单。仿 WINDOWS 界面风格的汉字输入提示行可以参看图 5-1,其中编辑器窗口的下方有一个拼音输入法的提示行。

为了简化应用程序的编程,我们设计了一个和 geth 用法完全一样的新键盘输入函数 gethan。新的输入函数不仅能够对键盘和鼠标信号作出相应的反应,而且增加了汉字输入的有关功能。因此,在需要输入汉字的应用程序中可以使用函数 gethan 代替输入函数 geth。

#### 程序 8-1 汉字输入函数。

为了便于理解,下面我们给出 gethan 函数的简化工作过程:

```

使用 geth 函数取一个键盘码;
如果该键盘码是输入法切换功能键 Alt_F1 至 Alt_F10,则
    处理提示行;
如果当前某汉字输入法已装入,则
    将该键盘码作为汉字输入码的首码交该汉字输入法模块处理;
否则
    查看是否应该将该键盘码转换为全角字符。

```

```

/* -----
   函数 gethan : 基本汉字输入模块
----- */

#include <hanenv.h>
#include <ctype.h>

/* ----- 西文字符输入方式 ----- */
#define BANJIAO      0      /* 输入半角字符 */
#define QUANJIAO     1      /* 输入全角字符 */

/* ----- 全角单、双引号输入方式 ----- */
#define LEFT_QUOTATION /* 输入全角左引号 */
#define RIGHT_QUOTATION /* 输入全角右引号 */

/* ----- 全局变量定义 ----- */
int _HanMode   = ALT_F10;      /* 当前汉字输入方法 */
int _AscMode   = BANJIAO;     /* 西文字符输入方式 */
int _DispPrompt = NO;         /* 提示行是否已经弹出 */
int _PromptStyle = WIN_STYLE; /* 提示行风格 */
int _PmtColor   = BLACK;      /* 提示行字符颜色 */
int _PmtBk      = LIGHTGRAY;  /* 提示行背景颜色 */
int _PromptLine = 450;        /* 提示行位置 */
int _SectionCode;             /* 区码 */
int _PositionCode;            /* 位码 */
int _Quotation   = LEFT_QUOTATION; /* 左右全角单引号标志 */
int _DBQuotation = LEFT_QUOTATION; /* 左右全角双引号标志 */
MODE_TYPE _Mode[10]           /* 汉字输入法参数 */
{
    {"国标区位", 0, NULL, 0, YES, _GetSP, 0},
    {"", 0, NULL, 0, NO, NULL, 0},
    {"", 0, NULL, 0, NO, NULL, 0},
    {"", 0, NULL, 0, NO, NULL, 0},
    {"", 0, NULL, 0, NO, NULL, 0},
    {"英文数字", 0, NULL, 0, YES, _GetAscii, 0},
    {"", 0, NULL, 0, NO, NULL, 0},
    {"", 0, NULL, 0, NO, NULL, 0},
    {"图形符号", 0, NULL, 0, YES, _GetFigure, 0},
    {"", 0, NULL, 0, NO, NULL, 0}
};

extern unsigned (* mouseKB)(); /* 虚拟键盘的激活条件 */

```



```

extern void _Cdecl _ClearPrompt(int clear_mode);

unsigned _Cdecl gethan(void)
{
    unsigned h = geth();          /* 从键盘读入一个键码 */

    /* ----- 如果是输入方式切换键,转换输入方式 ----- */
    if(h >= KEY_Alt_F1 && h <= KEY_Alt_F10)
    {
        /* -- 如果该输入方式已被装入,显示或清理提示行 ----- */
        if(_Mode[h - KEY_Alt_F1].haveload)
        {
            delight_mouse();

            /* ----- 如果提示行尚未弹出,首先弹出提示行 ----- */
            if(_DispPrompt == NO)
            {
                _DispPrompt = YES;

                /* ----- 保存提示行位置上原来的图象 ----- */
                _MoveImage(0, _PromptLine, 80, 30, 0, 780 - _ScreenTop);

                /* ----- 显示提示行 ----- */
                _Block(0, _PromptLine, 80, 30, _PmtBk);
                draw_rectangle(0, _PromptLine, 640, 30, _PmtColor, 0xff);

                /* ----- 如果是 WINDOWS 风格的提示行,显示提示框 ----- */
                if(_PromptStyle == WIN_STYLE)
                {
                    _Hole(14, _PromptLine + 4, 35, 22, DARKGRAY, _PmtBk + 8, 1);
                    _Hole(54, _PromptLine + 4, 68, 22, DARKGRAY, _PmtBk + 8, 1);
                    _Hole(126, _PromptLine + 4, 100, 22, DARKGRAY, _PmtBk + 8, 1);
                    _Hole(230, _PromptLine + 4, 324, 22, DARKGRAY, _PmtBk + 8, 1);
                    _Hole(558, _PromptLine + 4, 28, 22, DARKGRAY, _PmtBk + 8, 1);
                    _Hole(590, _PromptLine + 4, 36, 22, DARKGRAY, _PmtBk + 8, 1);
                }

                /* ----- 如果是 DOS 风格的提示行,显示提示分隔符 ----- */
                else
                {
                    outxyc(15, _PromptLine + 7, _PmtColor, ':');
                    outxyc(69, _PromptLine + 7, _PmtColor, '[');
                    outxyc(73, _PromptLine + 7, _PmtColor, ']');
                }
            }
        }
    }
}

```

```

}

/* ----- 显示西文字符输入提示 ----- */
outxys(2, _PromptLine+7, _PmtColor, _AscMode?"全角":"半角");

/* ----- 如果鼠标虚拟键盘已装入,显示其提示符 ----- */
if(mouseKB)
    outxys(74, _PromptLine+7, _PmtColor, "键盘");
}

/* ----- 清提示行,显示输入法名称 ----- */
_HanMode    = h-KEY_Alt_F1;
_PositionCode = 0;
_ClearPrompt(0);
outxys(7, _PromptLine+7, _PmtColor, _Mode[_HanMode].modename);
light _mouse();
}

/* ----- 提示行已弹出,但所要之输入法未安装,故取消提示行 ----- */
else if(_DispPrompt)
{
    _DispPrompt = NO;
    _HanMode    = ALT_F10;

    /* ----- 恢复提示行下面原来的图象 ----- */
    delight _mouse();
    _MoveImage(0, 780-_ScreenTop, 80, 30, 0, _PromptLine);
    light _mouse();
}
}

/* ----- 如果所需输入模式已安装,进入当前输入模式处理函数 ----- */
else if(_Mode[_HanMode].get_han)
    h = (* (_Mode[_HanMode].get_han))(h);

/* ----- 如果输入为西文字符,并且西文模式为全角方式 ----- */
if(_AscMode==QUANJIAO && isprint(h))
{
    /* ----- 如果在汉字输入模式下,转换汉字的编辑符号 ----- */
    if(_HanMode!=ALT_F10)
    {
        switch(h)
        {
            case '.': /* 句号“。” */
                h = 0xa3a1;

```

```

        break;
    case '/':          /* 顿号“、”          */
        h = 0xa2a1;
        break;
    case '[':          /* 左书名号“《”          */
        h = 0xb6a1;
        break;
    case ']':          /* 右书名号“》”          */
        h = 0xb7a1;
        break;
    case '\\':         /* 全角双引号          */
        h = _DBQuotation? 0xb1a1:0xb0a1;
        _DBQuotation = ! _DBQuotation;
        break;
    case '\':         /* 全角单引号          */
        h = _Quotation? 0xafa1:0xaea1;
        _Quotation = ! _Quotation;
        break;
    }
}

/* ----- 如果为半角西文字符,转换为全角字符 ----- */
if(h==' ')
    h = 0xalal;
else if(isprint(h))
    h = ((161+h-'!')*256+163);
}

/* ----- 返回得到的统一键盘编码 ----- */
return h;
}

```

在函数 gethan 中用到了一个用于清提示行信息的内部过程,见程序 8-2。

#### 程序 8-2 清提示行。

gethan 函数用于清理提示行上的各种显示信息。

```

/* -----
    内部函数 _ClearPrompt : 清提示行
    ----- */
#include (hanenv.h)

```

```

void _Cdecl _ClearPrompt(clear_mode)
int clear_mode;          /* 清理模式 */
{

    /* ---- 如果清理模式为 0, 首先清理输入法名称区 ---- */
    if(clear_mode(1)
        _Block(7, _PromptLine+7, 8, CHAR_HIGH, _PmtBk);

    /* ---- 如果清理模式为 0 或 1, 还要清理输入码区 ---- */
    if(clear_mode<2)
        _Block(16, _PromptLine+7, 12, CHAR_HIGH, _PmtBk);

    /* ---- 无论清理模式如何, 一律清理重复码区和重复码提示区 ---- */
    _Block(29, _PromptLine+7, 40, CHAR_HIGH, _PmtBk);
    _Block(70, _PromptLine+7, 3, CHAR_HIGH, _PmtBk);
}

```

该函数也用于各种汉字输入法模块的程序设计。如果读者要设计新的汉字输入法模块, 也可以直接调用该函数清提示行信息。

## 8.2 输入法模块及其装入

在函数 gethan 中, 一旦当前已经处于某汉字处理方式下, 则应进入具体汉字输入法模块进行处理。在 HANENV 系统中我们提供了三类输入法模块: 第一类是基本输入法模块, 包括西文字符输入法、区位码输入法和图形字符输入法, 其特点是采用计算法由输入码求汉字机内码; 第二类是特制输入法, 目前有拼音输入法, 提供全拼拼音输入法和双拼拼音输入法, 其特点是根据输入法编码规则的特点组织码表信息, 查表速度快, 码表体积小; 以及双音输入法, 其特点是不仅能输入单字, 而且能输入词组, 大大提高了输入效率; 第三类是通用输入法, 其特点是不考虑特定输入法编码规则的细节, 只要用户提供按一定规则组织的代码对照表文件, 不用编程即可实现一种新的汉字输入法。

由于 HANENV 系统的汉字输入法采用了相对独立的编程方法, 读者也可以自行开发第二类汉字输入法。

### 程序 8-3 西文字符输入法模块。

实际上, 该模块相当于一个空操作模块, 作用是占用一个输入法切换位置。

```

/* -----
    函数 _GetAscii : 西文字符输入
    ----- */
#include (hanenv.h)

```

```

unsigned _Cdecl _GetAscii(h)
unsigned n;          /* 进入该输入法时的编码串首码 */
{
    return h;
}

```

极少有人愿意使用区位码输入汉字,但由于区位码和汉字机内码是一一对应的,所以作为万般无奈时的最后手段(如使用拼音码而又不会读某字),各汉字操作系统均配有区位码输入方法。区位码的编码规则相当简单:使用4位数字串表示某汉字或图形符号的区位码,区码在前,位码在后。例如,汉字“啊”的区位码是“1601”。

#### 程序 8-4 区位码输入法模块。

该输入法模块比较简单,其简要处理过程如下:

```

    如果首码是数字,则
    {
        继续接受编码的剩余部分处理;
        将输入码串组成区位码;
    }
    返回区位码。

```

这里要注意的是如果首码(由参数h带入)不是数字,则直接返回该码。相当于在区位码输入法中按下字母键和标点符号键时仍按ASCII码处理。在其他输入法模块中也是如此,首先要判定首码是否为该输入法的合法键码集合,如果不是,即直接返回。读者在开发自己的输入法模块时也要注意这个问题。

```

/* -----
   函数 _GetSP : 区位码输入法
   ----- */

#include <hanenv.h>

extern void _Cdecl _ClearPrompt(int clear_mode);

unsigned _Cdecl _GetSP(h)
unsigned h;          /* 进入该输入法时的编码串首码 */
{
    unsigned i=0, code=0;

    /* ----- 如果 h 为数字,按区位码首码处理 ----- */
    if(h>='0' && h<='9')

```

```

{

/* ----- 清提示行 ----- */
_ClearPrompt(1);

/* ----- 对区位码的码长(4)循环 ----- */
while(i<4)
{

/* ----- 显示首码,处理首码 ----- */
outxyc(16+i,_PromptLine+7,_PmtColor,h);
code=code*10+h-'0';
i++;

/* ----- 接收其余各码,如果不是数字键 ----- */
while(i<4 && ((h=geth())<'0' || h>'9'))
{

/* ----- 如果输入了退格键,作废当前码 ----- */
if(i>0 && (h==Backspace || h==KEY_Left))
{
_Block(16+(--i),_PromptLine+7,1,CHAR_HIGH,_PmtBk);
code/=10;
}

/* ----- 如果要切换输入法 ----- */
else if(h<=KEY_Alt_F1 && h>=KEY_Alt_F10)
{
ungeth(h);
return 0;
}

/* ----- 如果要作废当前的输入码 ----- */
else if(h==KEY_ENTER)
{
_ClearPrompt(1);
return 0;
}

/* ----- 其他各键均不合法 ----- */
else
putch(Beep);
}
}

```

```

/* —— 生成汉字的机内码,并将对应的汉字在提示行显示 —— */
h = (code/100+160)|(code%100+160)<<8;
outxyh(29,_PromptLine+7,_PmtColor,h);
}

/* ————— 返回汉字的机内码 ————— */
return h;
}

```

■ 国标 GB2312-80《信息交换用汉字编码字符集》中第 1 区到第 9 区是图形符号区,由于一般的汉字输入法均不对这些符号编码,所以除了部分全角西文 ASCII 码可以在切换到全角模式后输入,其他符号就只能使用区位码输入法输入了。但是其中有些符号如制表符号等的使用频率相当高,采用区位码输入太不方便。因此我们设计了一个专用的图形符号输入法模块用以快速输入这些图形符号。

在设计一种输入法时首先要进行的是功能设计。我们设计的图形符号输入法的基本功能为:

1. 输入区号(1—9)即可调出该区的所有图形符号,首先显示该区的第 1 页(前 10 个图形符号);
2. 用“+”键向后翻页,“-”键向前翻页;
3. 用数字键“0”-“9”选择所要的图形符号;
4. 其他键直接作为西文 ASCII 码输出,但不改变当前输入法。

也就是说,如果我们要画一条粗横线,只需在选择图形符号输入法后键入区号“9”,然后连续按下“5”键选定制表符号“—”即可。

设计好功能以后,就可以编程实现了。HANENV 系统的图形符号输入法模块的程序设计见程序 8-5。

#### 程序 8-5 图形符号输入法模块。

图形符号输入法模块的基本处理流程为

如果已在图形符号输入法中则

{

    如果接收到数字键则

        计算图形符号的机内码;

    如果接收到“-”键则

        提示行重码区向前翻页;

    如果接收到“+”键则

        提示行重码区向后翻页;

}

否则(即第一次进入图形符号输入法)

输入区号;  
返回图形符号的机内码。

```

/* -----
   函数 _GetFigure : 输入图形符号
   ----- */
#include <hanenv.h>
extern void _Cdecl _ClearPrompt(int clear_mode);
static void _Cdecl _DisplayFigurePrompt(int section,int position);

unsigned _Cdecl _GetFigure(h)
unsigned h;                /* 进入该输入法时的编码串首码 */
{
    /* -- 如果已在图形符号方式,可以连续选择输入图形符号 ---- */
    if(_PositionCode)
    {

        /* ---- 如果是数字键直接确定图形符号的机内码 ---- */
        if(h>='0' && h<='9')
            h = (_SectionCode+160)|(_PositionCode+160+(h=='0'? 9:h-'1')<<8);

        /* ---- 如果是 - 或 + 重码区向前或向后翻页 ---- */
        else if(h=='-' || h=='+')
        {
            if(h=='-')
                _PositionCode = _PositionCode<91? _PositionCode+10: _PositionCode;
            else
                _PositionCode = _PositionCode>1? _PositionCode-10:1;
            _DisplayFigurePrompt(_SectionCode,_PositionCode);
            h = 0;
        }
    }

    /* ---- 如果是第一次进入图形符号方式,输入区码 ---- */
    else if(h>='1' && h<='9')
    {
        _ClearPrompt(1);
        outxyc(16,_PromptLine+7,_PmtColor,h);
        _SectionCode = h-'0';
        _PositionCode = 1;
        _DisplayFigurePrompt(_SectionCode,_PositionCode);
        h = 0;
    }
}

```



```

    return h;
}

```

我们将图形符号输入法模块中组织提示行显示的部分从函数 `_GetFigure` 中独立出来, 作为一个内部函数:

```

/* --- 内部函数_DisplayFigurePrompt: 显示图形符号输入法的提示 --- */
static void _Cdecl _DisplayFigurePrompt(section, position)
int section;          /* 区号 (1-9) */
int position;         /* 该页首位号 (每页 10 个图形符号) */
{
    int i, j = 29;
    char rest[6];

    /* ----- 显示该页的图形符号提示 ----- */
    _ClearPrompt(2);
    for(i=1; i<=(position==91? 4:10); i++)
    {
        outxyc(j++, _PromptLine+7, _PmtColor, i%10-'0');
        outxyc(j++, _PromptLine+7, _PmtColor, ':');
        outxyh(j, _PromptLine+7, _PmtColor, (section+160)|(position+i+159)<<8);
        j+=2;
    }

    /* ----- 显示该区剩余图形符号数目 ----- */
    sprintf(rest, "%03d", position==91? 0:(85-position));
    outxys(70, _PromptLine+7, _PmtColor, rest);
}

```

这样在设计函数 `_GetFigure` 时就比较简单了, 整个函数显得相当简洁。其实这就是所谓的结构化程序设计方法中最重要的一环: 模块化程序设计。熟悉软件工程的读者会发现, 整个 HANENV 系统都是用软件工程方法设计, 并用结构化程序设计方法编程的。希望读者在应用 HANENV 系统编写应用程序时, 也能采用软件工程的基本思想, “自顶向下, 逐步求精”, 设计出性能良好的应用程序来。

全角/半角 ASCII 码的处理分散在几个不同的地方: 在函数 `gethan` 中, 根据全局变量 `_AscMode` 的值确定是否将从键盘输入的半角 ASCII 码转换为全角 ASCII 码; 程序 8-6 中的函数 `_SetAscMode` 确定一旦使用全角/半角转换热键后应该作哪些操作; 而函数 `init_mode` (见程序 8-9) 将函数 `_SetAscMode` 和热键 `Ctrl+F9` 通过触发器联结起来。

#### 程序 8-6 设置西文全角/半角输入方式。

切换西文字符的全角和半角状态被设计为无论提示行是否弹出, 只要按下热键就可以

切换西文字符的输入状态。如果提示行已经弹出,则修改提示行左端的西文输入状态提示信息。

```

/* -----
   函数 _SetAscMode ; 设置输入西文字符的模式
   ----- */

#include <hanenv.h>

extern int _AscMode;

unsigned _Cdecl _SetAscMode(unsigned h)
{
    _AscMode = !_AscMode;

    /* -- 如果提示行已弹出,显示西文字符的全角/半角模式 -- */
    if(_DispPrompt)
        putxys(2, _PromptLine+7, _PmtColor, _PmtBk, _AscMode?"全角":"半角");
    return h;
}

```

以上几个函数是 HANENV 的基本输入法模块。

细心的读者会发现, gethan 函数中对各输入法模块的调用是通过一个函数指针间接进行的。而该函数指针又是一个 MODE\_TYPE 类型的全局数组变量的一个分量。因此, 在使用 gethan 函数之前, 应该首先将某输入法的各项参数(包括其处理函数的地址) 加入全局数组变量 \_Mode。这项工作由库函数 set\_han\_mode 完成。

#### 程序 8-7 设置一个输入法的有关参数。

对一种汉字输入法来说, 需要明确的参数有以下几个:

1. 该输入法的名称, 最多 4 个汉字(用于提示行显示);
2. 切换热键, 可在复合功能键 Alt+F1 到 Alt+F10 之间取值;
3. 该输入法的处理函数的地址, 要求输入法处理函数取下面的形式:

```

unsigned <函数名>(unsigned h)
{
    ...
}

```

即参数 h 为输入码字符串中的第一个字符, 返回值为汉字机内码;

如果该输入法需要码表文件, 那么还需要两个参数:

4. 码表文件名;
5. 码表装载何处。该参数可以取值:

XMS : 扩充存储器(使用 XMS 规范);

EMM : 扩充存储器(使用中断 INT15H);  
 DSK : 硬盘上;  
 MEM : 内存中(只限于不用码表的输入法)。

使用码表的原因是为应用程序提供一个方便、通用的输入法接口。应用程序的开发者甚至用户只要按规定的格式提供相应的码表文件,就可以为使用 HANENV 系统的应用程序挂接一种新的汉字输入法。当然,也可以任意修改已有的码表文件以适应自己工作的需要。使用码表的通用输入方法见 8.4。

```

/* -----
   函数 set_han_mode : 设置一个输入法
   ----- */
#include <hanenv.h>
#include <string.h>
#include <alloc.h>

int _Cdecl set_han_mode(altfn,modename,get_han,codefile,wherecodes)
int      altfn;                /* 该输入法使用的切换键 */
char     * modename;           /* 该输入法的名称 */
unsigned (* get_han)();        /* 该输入法的操作函数指针 */
char     * codefile;           /* 该输入法使用的码表文件 */
int      wherecodes;           /* 该输入法的码表装载何处 */
{
    char tmp[81];
    int i = 0;

    /* ----- 如果该输入法无需码表文件 ----- */
    if(codefile == NULL)
    {
        memcpy(_Mode[altfn].modename,modename,8);
        (_Mode[altfn].modename)[8] = 0;
        _Mode[altfn].codeset      = NULL;
        _Mode[altfn].wildchar     = 0;
        _Mode[altfn].maxcodes     = 0;
        _Mode[altfn].haveioad     = YES;
        _Mode[altfn].get_han      = get_han;
        return YES;
    }

    /* ----- 以下处理需要装入码表的输入法 ----- */
    if((_Mode[altfn].codefile = fopen(codefile,"rb")) == NULL)
        return NO;
}

```

```

do
{
    /* ----- 如果是[Text],转去处理码表表体 ----- */
    fgets(tmpline,80,_Mode[altfn].codefile);
    if(strncmp(tmpline,"[Text]",6)==0)
        goto Success;

    /* ----- 处理输入法名称 ----- */
    if(strncmp(tmpline,"Name=",5)==0)
    {
        memcpy(_Mode[altfn].modename,tmpline+5,8);
        (_Mode[altfn].modename)[8] = 0;
    }

    /* ----- 处理最大码长 ----- */
    if(strncmp(tmpline,"MaxCodes=",9)==0)
        _Mode[altfn].maxcodes = atoi(tmpline+9);

    /* ----- 处理该输入法所用键码集合 ----- */
    if(strncmp(tmpline,"UsedCodes=",10)==0)
    {
        int len = strlen(tmpline+10)-2;
        _Mode[altfn].codeset = malloc(41);
        memcpy(_Mode[altfn].codeset,tmpline+10,len);
        (_Mode[altfn].codeset)[len] = 0;
    }

    /* ----- 处理通用代换码 ----- */
    if(strncmp(tmpline,"WildChar=",9)==0)
        _Mode[altfn].wildchar = tmpline[9];
    i++;
}while(i<7);

/* ----- 如果缺少码表表体部分,装载失败 ----- */
if(_Mode[altfn].codeset)
{
    free(_Mode[altfn].codeset);
    _Mode[altfn].codeset = NULL;
}
return NO;

/* ----- 继续装载码表表体 ----- */
Success;

```

```

/* ----- 如果定义了通用代换码,将其加入字符集 ----- */
if(_Mode[altfn].wildchar)
{
    i = strlen(_Mode[altfn].codeset);
    (_Mode[altfn].codeset)[i] = _Mode[altfn].wildchar;
    (_Mode[altfn].codeset)[i+1] = 0;
}

/* -- 根据 wherecodes 将码表装入扩充内存或留在硬盘上 -- */
_Mode[altfn].wherecodes = wherecodes;
_Mode[altfn].filefront = ftell(_Mode[altfn].codefile);
fseek(_Mode[altfn].codefile, 0L, SEEK_END);
_Mode[altfn].filelen = ftell(_Mode[altfn].codefile);
fseek(_Mode[altfn].codefile, 0L, SEEK_SET);
switch(wherecodes)
{
    case XMS:
        _Mode[altfn].handle = _LoadXMS(_Mode[altfn].codefile, _Mode[altfn].filelen
            - _Mode[altfn].filefront);
        fclose(_Mode[altfn].codefile);
        break;
    case EMM:
        _Mode[altfn].EMM_addr = _LoadEMM(_Mode[altfn].codefile,
            _Mode[altfn].filelen - _Mode[altfn].filefront);
        fclose(_Mode[altfn].codefile);
        break;
}
_Mode[altfn].get_han = get_han;
_Mode[altfn].haveload = YES;
return _Mode[altfn].haveload;
}

```

### 程序 8-8 卸去某输入法。

```

/* -----
    函数 free_han_mode : 卸掉某输入法
    ----- */

#include <hanenv.h>
#include <alloc.h>

void _Cdecl free_han_mode(altfn)
int altfn;                /* 该输入法使用的切换键 */

```

```

{
    /* --- 如果该输入法带有码表,首先释放其占用的存储空间 --- */
    if(_Mode[altfn].haveload)
    {
        if(_Mode[altfn].codeset)
            free(_Mode[altfn].codeset);
        if(_Mode[altfn].wherecodes == XMS)
            _FreeXMS(_Mode[altfn].handle);
        if(_Mode[altfn].wherecodes == DSK)
            fclose(_Mode[altfn].codefile);
    }

    /* -----恢复输入法模块参数变量的初始值 ----- */
    _Mode[altfn].modename[0]= 0;
    _Mode[altfn].maxcodes    = 0;
    _Mode[altfn].codeset     = NULL;
    _Mode[altfn].haveload    = NO;
    _Mode[altfn].get_han     = NULL;
    _Mode[altfn].wildchar    = 0;
}

```

有些应用程序只要求能够显示汉字,不要求汉字的输入功能,因此只要使用前几章所介绍的内容就够了。因此我们在设计 HANENV 系统的汉字输入功能时,将汉字输入模块的安装分为三个级别:第一级别是不安装汉字输入功能,此时在应用程序中可以使用函数 `geth` 接收键盘和鼠标的按钮信息;第二级别是直接使用 `gethan` 函数代替 `geth` 函数,此时可以使用区位码和图形符号等基本输入法;第三级别是应用函数 `set_han_mode` 安装拼音、五笔字形(通过为通用输入法配备五笔字型码表来完成)等高级汉字输入法和使用函数 `set_asc_mode` 安装全角西文字符输入法。

#### 程序 8-9 安装全角 ASCII 码输入模块。

```

/* -----
    函数 set_asc_mode : 安装全角/半角汉字转换切换键
----- */
#include <hanenv.h>

void _Cdecl set_asc_mode(void)
{
    _Triggers[_TriggerCount].press_key = KEY_Ctr_F9;
    _Triggers[_TriggerCount].trigger   = _SetAscMode;
    _TriggerCount++;
}

```

}

### 8.3 拼音输入法模块的设计

拼音输入法其实是最普及的汉字输入法,几乎所有的汉字系统都配有一、两种拼音输入法。拼音输入法的最大优点是不需要培训,一般人只要上过学,没有不会汉语拼音的。而拼音输入法的缺点是输入速度比较慢。造成输入速度慢的主要原因有两个:一是重码率高,一是码长较长。如果不考虑四声的话,汉语共有不到400个不同的音节;而光是收录在国标GB2312-80《信息交换用汉字编码字符集》中的常用汉字和次常用汉字就有6,763个之多,平均每音节有16个重码字。当然,重码字不会分布得这么均匀,有些音节如“wai”、“run”等只有两、三个重码字,而象音节“ji”、“yi”等竟有上百个重码字之多!

重码字多就需要经常在提示行的重码区翻页、搜寻,必然影响输入速度。一个常用的弥补方法是对同音字按使用频度进行排序,将常用字排在前面,这样就可以尽量减少查找和翻页的时间。

另外,拼音码是非定长码,最长的音节如“zhuang”等由6个字母组成,而最短的音节“a”只有1个字母。同时,由于码长不固定,对于短码还需要在其后加上一个结束码(通常使用空格键),更增加了输入汉字时的平均击键次数。常用的解决方法是采用简拼、双拼等简化拼音法,用单个元音字母代替双码声母,单个辅音字母代替多码韵母。但是这样又增加了操作人员学习上的负担,因此这类输入法并不能完全代替全拼拼音输入法。

此外,打破按字输入的框框,使用词组、联想、双拼等技术都可以提高拼音输入法的输入速度,使之适应需要快速汉字输入的场所。

HANENV系统的拼音输入法模块同时提供全拼和双拼两种拼音输入法,其中双拼输入法的编码规则和金山系统的双拼输入法类似,详见第15章“HANENV系统的库函数”中有关函数\_GetPY的说明。这两种拼音输入法共用一套代码对照表,其逻辑结构如表8-1所示。

表 8-1 拼音输入法的代码对照表

双拼码	全拼码	重 码 表
a	a	阿啊呵吖啊腌嘎
aa	zha	扎炸渣闸榨诈乍轧眨栅札铡喳咋查蜡楂楂
ab	zhua	抓爪挝
ac	zhuan	转专砖赚撰篆传颧啮饕饨
ad	zhao	照找赵召招罩兆昭沼肇朝着嘲爪啁钊棹策诏
...	...	...

拼音输入法的基本思路就是在上述对照表中的第一或第二列中查找输入码,如果查到了就在提示行的重码区显示重码表的前10个重码,然后等待用户在其中挑选自己所需的汉字。

## 程序 8-10 拼音输入法。

拼音输入法是 HANENV 系统中最重要的程序模块之一,其编程质量直接影响到应用程序人机界面的实用程度。从便于使用和提高程序效率的角度出发,我们在设计\_GetPY 模块时考虑了以下几项功能:

1. “高频先见”,即最常用的汉字位于重码表的最前面,这一点通过精心排列代码对照表来实现;
2. 允许多音字,通过在代码对照表中相应行设置多个汉字码实现;
3. 设置重复选择键 F1—F10,可以再次选择提示行重码选择区中的汉字;
4. 尽量减少内存使用量。为了做到这一点,我们将拼音输入法的代码对照表分为三部分:第一部分是数组\_PYdata,内容为国标 GB2312—80《信息交换用汉字编码字符集》中 6763 个汉字的 HANENV 系统的统一键盘码(机内码),按拼音序和“高频先见”的原则排列,多音字多次出现;第二部分是二维数组\_PYindex,其第一列是双拼码,第二列是该双拼码对应的同音字重码串中第一个汉字在数组\_PYdata 中的位置;第三部分是字符串指针数组\_QPindex,存放着全拼拼音码和双拼拼音码的对照表。

拼音输入法模块的基本工作过程为

如果参数 h(输入码字符串的首字符)是重复选择键,则

直接查出对应的汉字的机内码并返回;

如果 h 不是小写字母键(拼音码输入字符串由小写字母打头),则

直接返回其键码;

否则

清提示行及输入编码缓冲区,首码(h)进缓冲区;

循环:继续处理输入码余下的部分:

{

从键盘输入一个键码 h;

如果 h 是重复选择码,则

将该键码送回键盘缓冲区并返回;

如果 h 是退格键,则

调整提示行和输入码缓冲区;

如果 h 是向前翻页键(“-”)或向后翻页键(“+”),则

调整代码对照表指针和提示行;

如果 h 是重码选择键,则

查出对应的汉字的机内码并返回;

如果 h 是小写字母或空格,则

如果已经输入了两个字符并且在双拼输入法下,则

在\_PYindex 查找双拼输入码;

否则(即在全拼输入法下)

先查双拼—全拼对照表\_QPindex,再查\_PYindex;

}

下面是程序中几个全局变量的说明:



\_PYdata —— 汉字音序表;  
 \_PYindex —— 双拼码索引;  
 \_QPindex —— 全拼码索引;  
 \_PositionCode —— 提示行重码区中第一个汉字在 \_PYdata 中的位置。

```

/* -----
   程序模块 PYMODE.C : 拼音输入法
   ----- */

#include <stdlib.h>
#include <alloc.h>
#include <ctype.h>
#include <string.h>
#include <hanenv.h>

/* -----
   Extern variables from gethan.
   ----- */

extern unsigned _PYindex[399][2];
extern unsigned _PYdata[];
extern char      *_QPindex[];

static void _Cdecl _DisplayPYPrompt (int pos,int len);
static int  _Cdecl _PYCodeCmp      (int *p1,int *p2);
static int  _Cdecl _PYsrch         (char *dic[],int h,char *key,int size);

/* -----
   函数 _GetPY : 拼音输入法
   ----- */

unsigned _Cdecl _GetPY(h)
unsigned h;          /* 进入该输入法时的编码串首码 */
{
    int i      = 0;          /* 输入码位置指针 */
    int ready  = NO;         /* 输入码是否完成 */
    int b      = 0;          /* 重码串首汉字在索引中的位置 */
    int n      = 0;          /* 重码个数 */
    int *ip    = NULL;       /* 查找索引用的临时变量 */
    int pos    = 0;          /* 查找全拼索引用的临时变量 */
    char code[14];           /* 拼音输入码缓冲区 */

    /* ----- 如果是重复选择键则返回相应的汉字 ----- */
    if(b>=KEY_Alt_1 && h<=KEY_Alt_0 && _PositionCode)
        return _PYdata[_PositionCode+(h-KEY_Alt_1)%10];

```

```

/* ----- 如果不是小写字母键则直接返回其键码 ----- */
if(h<'a' || h>'z')
    return h;

/* ----- 否则清提示行及输入编码缓冲区,首码进缓冲区 ----- */
_ClearPrompt(1);
memset(code,0,14);
code[i++] = h;
outxyc(15+i,_PromptLine+7,_PmtColor,h);

/* ----- 继续处理输入码余下的部分 ----- */
while(1)
{
    /* ----- 取一键盘码 ----- */
    h = geth();

    /* ----- 如果为作废码则退出 ----- */
    if(k == KEY_ENTER)
    {
        _ClearPrompt(1);
        return 0;
    }

    /* ----- 如果是重复选择码 ----- */
    if(h>=KEY_Alt_F1 && h<=KEY_Alt_F10)
    {
        ungeth(h);
        return 0;
    }

    /* ----- 如果是退格键 ----- */
    else if(i>0 && (h==Backspace || h==KEY_Lcft))
    {
        _Block(16+(-i),_PromptLine+7,1,CHAR_HIGH,_PmtBk);
        code[i] = 0;
        ready = NO;
        if(i==0)
            return 0;
    }

    /* ----- 如果是向后翻页键 ----- */
    else if(ready && h==' ' && n>10)
    {
        n
            -= 10;
    }
}

```

```

    _PositionCode += 10;
    _DisplayPYPrompt(_PositionCode,n);
}
/* ----- 如果是向前翻页键 ----- */
else if(ready && h=='-' && _PositionCode>b)
{
    _PositionCode -= 10;
    n += 10;
    _DisplayPYPrompt(_PositionCode,n);
}
/* ----- 如果是重码选择键 ----- */
else if(ready && (isdigit(h) || h==' '))
    return _PYdata[_PositionCode+(h=='0'?9:(h==' '?0:h-'1'))];

/* ----- 继续输入拼音输入码 ----- */
else if(i<=6 && ((h>='a' && h<='z') || h==' '))
{
    code[i++] = h;
    outxyc(15+i,_PromptLine+7,_PmtColor,h);

    /* ----- 处理双拼拼音输入法 ----- */
    if(strcmp(_Mode[_HanMode].modename,"双拼拼音")==0 && i==2)
    {
        h = code[0]*256+code[1];
        if((ip=(int *)bsearch(&h,_PYindex,sizeof(_PYindex)/4,4,
            _PYCodeCmp))!=NULL)
        {
            b = ip==(int *)_PYindex? 0: *(ip-1);
            n = ip==(int *)_PYindex? _PYindex[0][1]: *(ip+1)-*(ip-1);
            _PositionCode = b;
            _DisplayPYPrompt(_PositionCode,n);
            ready = YES;
        }
        else
        {
            i = 0;
            putch(Beep);
            _Block(16+i,_PromptLine+7,2,CHAR_HIGH,_PmtBk);
        }
    }

    /* ----- 处理全拼拼音输入法 ----- */
    else if(strcmp(_Mode[_HanMode].modename,"全拼拼音")==0)
    {

```

```

        if((pos = _PYsrch(_QPindex,399,code,0))>=0)
        {
            h = (*_QPindex[pos])*256+*(_QPindex[pos]+1);
            if((ip=(int *)bsearch(&h,_PYindex,sizeof(_PYindex)/4,4,
                _PYCodeCmp))!=NULL)
            {
                b=ip==(int *)_PYindex? 0:*(ip-1);
                n=ip==(int *)_PYindex? _PYindex[0][1]:*(ip+1)-*(ip-1);
                _PositionCode = b;
                _DisplayPYPrompt(_PositionCode,n);
                ready = YES;
            }
        }
        else if(_PYsrch(_QPindex,399,code,i)==-1)
        {
            putch(Beep);
            code[--i]=0;
            _Block(16+i,_PromptLine+7,1,CHAR_HIGH,_PmtBk);
        }
    }
}
}
}

```

内部函数 `_DisplayPYPrompt` 用于处理拼音输入法下提示行信息的显示。

```

/* —— 内部函数 _DisplayPYPrompt : 显示拼音输入法的提示行 —— */
static void _Cdecl _DisplayPYPrompt(pos,len)
int pos;
int len;
{
    int i,j = 29;
    char rest[6];

    /* ----- 显示该页的同音字提示 ----- */
    _ClearPrompt(2);
    for(i=1;i<=(len>=10? 10:len);i++)
    {
        outxyc(j++,_PromptLine+7,_PmtColor,i%10+'0');
        outxyc(j++,_PromptLine+7,_PmtColor,',');
        outxyh(j,_PromptLine+7,_PmtColor,_PYdata[pos++]);
        j+=2;
    }
}

```

```

/* ----- 显示该区剩余同音字数目 ----- */
sprintf(rest,"%03d",len>10? len-10:0);
outxys(70,_PromptLine+7,_PmtColor,rest);
}

```

内部函数\_PYCodeCmp 用于在\_PYindex 中查找双拼码。该函数是 Turbo C 的二分法查找库函数 bsearch 所要求的,可参看 Turbo C 的库函数手册。

```

/* --- 内部函数 _PYCodeCmp : 查表所必需的比较函数 --- */
static int _PYCodeCmp(int *p1,int *p2)
{
    return *p1 - *p2;
}

```

内部函数\_PYsrch 用于在\_QPindex 中查找全拼拼音码所对应的双拼码。

```

/* ----- 内部函数 _PYsrch : 查拼音索引表 ----- */
static int _Cdecl _PYsrch(dic,h,key,size)
char *dic[];                /* 拼音码索引表          */
int h;                      /* 拼音码索引表长度      */
char *key;                  /* 汉字输入码缓冲区      */
int size;                   /* 汉字输入码缓冲区长度 */
{
    int l=0,m,c;
    while(l<=h)
    {
        m = (l+h)/2;
        if(size)
            c = strncmp(key,dic[m]+2,size);
        else
            c = strcmp(key,dic[m]+2);
        if(c==0)
            return m;
        else if(c>0)
            l = m+1;
        else
            h = m-1;
    }
    return -1;
}

```

为了使程序相对简洁一些,上面的拼音输入法模块中没有加入对词组、联想、双拼等功

能的处理。有人将中文操作系统中的汉字输入法的研究工作划分为三个阶段：第一个阶段是按字输入阶段；第二个阶段是按词输入阶段，此时出现了词组、联想、双拼等输入法，高频词先现也是这一时期的产物；第三阶段是整句输入阶段，目前已有一些基于句法的整句智能输入法出现，但应用尚不广泛。读者也可以在上述拼音输入法的基础上开发功能更强大的新型输入法。在随本书提供的软盘中有一个双拼输入模块Spmode，提供了按词组输入能力，可供读者参考。

在研制汉字系统时，尽量减少系统程序对内存的要求是最重要的目标。因此，使用汇编语言编程、充分利用扩充存储器、仔细分析码表结构和查表方法等都是设计输入法模块的基本方法。由于输入模块的特殊性，对速度的要求不是很高，只要能够跟上操作人员击键的速度就可以了。我们在为HANENV系统开发输入法模块时也应按上述原则进行。如果输入法模块占用了太多的内存，就会影响到使用HANENV系统的应用程序的开发，这是系统程序设计人员所要尽力避免的。

## 8.4 通用输入法模块的设计

本节我们介绍通用输入法模块的设计与编程。设计通用输入法有两个原因：其一是当前已有数百以至上千种汉字输入法问世，而且新的输入法还在不断出现。为这些输入法都编写出相应的程序模块显然是不可能的。因此除了配备几种基本的汉字输入法模块以外，最好再给应用程序的设计者一个通用的输入法接口，使其可以用比较简便的方法为应用程序扩充新的汉字输入法模块；其二是对于具体的应用程序的操作员来说，现有的汉字输入法模块的码表内容可能不尽适用，例如根据专业的不同所用汉字的使用频率会有所不同，词组内容也会有很大差别。因此，希望能够根据实际情况自行调整码表的内容。

基于以上考虑，我们设计了一个通用输入法模块，其特点为

1. 使用文件型外挂码表，码表内容可以自行调整；
2. 允许使用词组，在码表中单字和词组混排，输入方法完全相同；
3. 输入时可以使用万能代替键，实现模糊输入；
4. 只要提示行上有所需汉字或词组，可以使用重复选择键重复输入；
5. 码表可以装入扩充存储器以提高速度。

码表是通用输入法模块中最重要的部分。我们设计的通用输入法的码表文件是文本文件，由两部分组成。第一部分是码表的描述部分，由描述部分说明行引导。描述部分有若干描述语句构成，每句一行。描述部分的语句有：

[Description] : 描述部分说明行，必需是码表文件的第一行；

Name=〈输入法名称〉：输入法名称说明语句，输入法名称为4个汉字；

MaxCodes=〈码长〉：输入码长度说明语句，如果输入码长度不定，按最大码长输入；

UsedCodes=〈输入码集合输入码中所用键码说明语句；

WildChar=〈万能大体键〉万能代替键说明语句。

第二部分是码表表体，由表体说明行([Text])引导，其后即为输入码—汉字对照表。输

人码—汉字对照表的格式为

〈输入码〉〈汉字或词组〉

其中输入码长度应该等于描述部分给出的输入码长度,不足部分以空格补足。码表表体应按输入码进行排序。例如,HANENV 系统所配的五笔字形输入法码表文件的开头部分是:

```
[Description]
Name=五笔字型
MaxCodes=4
UsedCodes=abcdefghijklmnopqrstuvwxyz
WildChar=z
[Text]
a    工
aaaa 工
aaan 工艺
.....
```

参看程序 8-7(设置汉字输入法模块),可以更清楚地了解码表的处理工程。

#### 程序 8-11 通用汉字输入法模块。

通用汉字输入法模块的主要函数是\_GetNewCode,其基本处理流程为

如果参数 h(输入码的首字符)是重复选择键,则  
 返回直接在重码缓冲区中找到的相应汉字或词组;  
 如果 h 不在该输入法用键集合中,则  
 返回原来的 h;

否则

{

清提示行及输入编码缓冲区,首码进缓冲区;

循环:继续处理输入码余下的部分

{

如果 h 为空格键,则

结束输入码(跳出循环);

如果 h 为退格键,则

调整输入码缓冲区和提示行;

如果 h 在输入法用键集合中,则

输入码进缓冲区并在提示行显示;

如果输入码长度不够,则

```

        继续输入(h=geth());
        如果输入码已足够长,则
            退出循环;
    }
    用二分法在码表中查找输入码;
    如果查找失败,则
        蜂鸣器响,退出通用输入法模块;
    否则(查找成功)
    {
        在提示行上显示重码序列并选择(h=被选择的汉字);
        返回被选中的汉字;
    }
}

```

由于码表文件的尺寸较大,通常不能全部装入常规内存,所以程序中的查找算法比较复杂。我们在常规内存中开辟了一个512字节的码表缓冲区,查找工作是通过该缓冲区分块进行的。程序中有大量细节用于处理跨越缓冲区边界的不完整码表条目。

```

/* -----
   程序模块 NEWCODE : 通用汉字输入法
   ----- */

#include <hanenv.h>
#include <stdlib.h>
#include <alloc.h>
#include <ctype.h>
#include <string.h>

/* -- 将自定义输入法码表装入扩充内存时缓冲区尺寸定义 -- */
#define BUFFER_SIZE 512

/* ----- 重码提示符的字模点阵 ----- */
unsigned char _ArrowFonts[] =
{
    0x01,0xFF81,0xFF80,0x07,0xFF81,
    0xFFE0,0x1F,0xFFFF,0xFFF8,0x1F,0xFFFF,0xFFF8,0x07,
    0xFF81,0xFFE0,0x01,0xFF81,0xFF80,
    0x00,0xFFC0,0x00,0x03,0xFFC0,0x00,
    0x0F,0xFFFF,0xFFF0,0x0F,0xFFFF,0xFFF0,0x03,0xFFC0,0x00,
    0x00,0xFFC0,0x00,
    0x00,0x03,0x00,0x00,0x03,0xFFC0,
    0x0F,0xFFFF,0xFFF0,0x0F,0xFFFF,0xFFF0,0x00,0x03,0xFFC0,

```



```

    0x00,0x03,0x00
};

/* ----- 全局静态变量说明 ----- */
static char _CodeBuffer[BUFFER_SIZE]; /* 输入码表缓冲区 */
static char _CodePrompt[41];          /* 代码提示缓冲区 */
static char _DupPrompt[120];          /* 重码缓冲区 */
/* ----- 内部函数说明 ----- */
static int _Cdecl _SrchCode (unsigned char *front, char *inputcode,
                             int maxcodes, int wildchar);

static int _Cdecl _NextDup (unsigned char *buff, unsigned char
                             *code, int len, int wild);

static void _Cdecl _DispInputCode (int i, int len);
static unsigned _Cdecl _OutCode (unsigned h);
static char *_Cdecl _GetCodeBlock (long offset, MODE_TYPE *mode);
static unsigned char *_Cdecl _GetString (unsigned char *ptr, unsigned char
                                         *tmpline);

/* -----
    函数 _GetNewCode : 使用自定义输入法码表输入汉字
----- */
unsigned _Cdecl _GetNewCode(h)
unsigned h; /* 进入该输入法时的编码串首码 */
{
    unsigned char *front = NULL;
    unsigned char *p = NULL;
    unsigned char *q = NULL;
    int code_pos = 0;
    int pos = 0;
    int maxcodes = _Mode[_HanMode].maxcodes; /* 输入码最大长度 */
    int wildchar = _Mode[_HanMode].wildchar; /* 万能替换键码 */
    char *codeset = _Mode[_HanMode].codeset; /* 输入法用键集合 */
    FILE *codefile = _Mode[_HanMode].codefile; /* 输入法码表文件 */
    int forward = NO;
    int pageno = 0;
    int havewild = NO;
    long low, high, mid;
    static int i = 0;
    char tmpline[17];
    char tmpstr[17];
    char inputcode[13]; /* 输入码缓冲区 */

    /* ----- 如果是重复选择键则返回相应的汉字或词组 ----- */

```

```

if(h>=KEY_Alt_1 && h<=KEY_Alt_1+i)
{
    h -= KEY_Alt_1;
    h = h==9?'0':h+'1';
    return _OutCode(h);
}

/* --- 如果不在该输入法用键集合中则直接返回其键码 --- */
if(strchr(codeset,h) == NULL)
    return h;

/* --- 否则清提示行及输入编码缓冲区,首码进缓冲区 --- */
memset(inputcode,' ',maxcodes);
inputcode[maxcodes] = 0;
_ClearPrompt(1);

/* ----- 继续处理输入码余下的部分 ----- */
while(1)
{
    /* ----- 如果为作废码则退出 ----- */
    if(k == KEY_ENTER)
    {
        _ClearPrompt(1);
        return 0;
    }

    /* ----- 如果是重复选择码 ----- */
    if(h>=KEY_Alt_F1 && h<=KEY_Alt_F10)
    {
        _ClearPrompt(1);
        ungeth(h);
        return 0;
    }

    /* --- 空格键: 结束输入码 ----- */
    if(h == KEY_SPACE)
        break;

    /* --- 回车键: 作废当前输入码 ----- */
    if(h == KEY_ENTER)
    {
        _ClearPrompt(1);
        code_pos = 0;
    }
}

```

```

    }

    /* ----- 如果是退格键 ----- */
    else if (code_pos && (h == Backspace || h == KEY_Left))
    {
        _Block(16 + (code_pos), 457, 1, CHAR_HIGH, _PmtBk);
        inputcode[code_pos] = ' ';
    }

    /* ----- 如果在输入法用键集中 ----- */
    else if (strchr(codeset, h))
    {
        inputcode[code_pos] = h;
        outxyc(15 + code_pos++, _PromptLine + 7, _PmtColor, h);
    }

    /* ----- 如果输入码长度不够则继续输入 ----- */
    if (code_pos < maxcodes)
        h = getch();

    /* ----- 如果输入码已足够长则退出循环 ----- */
    else
        break;
}

/* ----- 如果输入码中有万能替换键 ----- */
if (strchr(inputcode, wildchar))
    havewild = YES;

/* ----- 如果输入码的第一键即为万能替换键 ----- */
if (inputcode[0] == wildchar)
{
    mid = _Mode[_HanMode].filefront;
    front = _GetCodeBlock(mid, _Mode + _HanMode);
    pos = 0;
    goto Search_Success;
}

/* ----- 用二分法在码表中查找输入码 ----- */
low = _Mode[_HanMode].filefront;
high = _Mode[_HanMode].filelen;
while (high > low)
{

```

```

if(high <= BUFFER_SIZE)
    low = _Mode[_HanMode].filefront;
if(high-low <= BUFFER_SIZE)
    mid = low;
else
    mid = (high+low-BUFFER_SIZE)/2;
front = _GetCodeBlock(mid, _Mode+_HanMode);
pos = _SrchCode(front, inputcode, maxcodes, havewild? wildchar:0);
if(pos < 0)
    high = mid-1;
else if(pos > strlen(front))
    low = mid+BUFFER_SIZE-maxcodes-16;
else
    goto Search_Success;
}
/* ----- 查找失败 ----- */
putch(Beep);
return 0;

/* ----- 查找成功 ----- */
Search_Success:

/* ----- 找到重码的第一个 ----- */
while(1)
{
    p = front+pos-1;
    while(p > front && ! strchr(codeset, *p))
        p--;
    if(p < front || p == front && mid > _Mode[_HanMode].filefront)
    {
        mid -= strlen(front);
        if(mid < _Mode[_HanMode].filefront)
            mid = _Mode[_HanMode].filefront;
        front = _GetCodeBlock(mid, _Mode+_HanMode);
        pos = _SrchCode(front, inputcode, maxcodes, havewild? wildchar:0);
        if(pos > strlen(front))
        {
            mid += strlen(front);
            fseek(codefile, mid, SEEK_SET);
            front = _GetCodeBlock(mid, _Mode+_HanMode);
            pos = _SrchCode(front, inputcode, maxcodes, havewild? wildchar:0);
            break;
        }
    }
}

```

```

        else if (mid > _Mode[_HanMode].filefront)
            continue;
        else
            break;
    }
    else
        break;
}
/* ----- 在提示行上显示重码序列并选择 ----- */
q = front + pos + maxcodes;
while(1)
{
    p = _CodePrompt;
    i = 0;
Loop:
    if (havewild)
        memcpy(_DupPrompt + i * maxcodes, q - maxcodes, maxcodes);
    q = _GetString(q, tmpline);
    if (strlen(tmpline) < (40 - (p - _CodePrompt)))
    {
        sprintf(p, "%d: %s", i == 9 ? 0 : i + 1, tmpline);
        p += strlen(tmpline) + 2;
        i++;
    }
    else
    {
        forward = YES;
        goto Display;
    }
NextPage:
    if (strlen(q) == 0)
    {
        mid += strlen(front);
        if (mid > _Mode[_HanMode].filelen)
            mid = _Mode[_HanMode].filelen - BUFFER_SIZE;
        front = _GetCodeBlock(mid, _Mode + _HanMode);
        q = front;
    }
    if (_NextDup(q, inputcode, maxcodes, wildchar))
    {
        q += maxcodes;
        goto Loop;
    }
}

```

```

/* ----- 显示重码序列 ----- */
Display:
_ClearPrompt(2);
outxys(29, _PromptLine+7, _PmtColor, _CodePrompt);
if(pageno && forward)
    _DrawF(70*8, _PromptLine+13, 3, 6, _PmtColor, _ArrowFonts);
else if(pageno)
    _DrawF(70*8, _PromptLine+13, 3, 6, _PmtColor, _ArrowFonts+18);
else if(forward)
    _DrawF(70*8, _PromptLine+13, 3, 6, _PmtColor, _ArrowFonts+36);

/* ----- 如果没有重码则直接返回结果 ----- */
if(i == 1)
{
    if(havewild)
        _DispInputCode(0, maxcodes);
    p = takehan(tmpline, &h);
    if(strlen(p))
        ungetstr(p);
    return h;
}

/* ----- 在重码中选择 ----- */
AcceptSelect:
h = geth();

/* ----- 如果输入的是数字键则选择 ----- */
if(h >= '1' && h <= i+'1' || i == 9 && h == '0')
{
    if(havewild)
        _DispInputCode(h == '0' ? 9, h-'1', maxcodes);
    p = strchr(_CodePrompt, h)+2;
    p = takehan(p, &h);
    q = tmpstr;
    while(ishan(p))
    {
        *q++ = *p++;
        *q++ = *p++;
    }
    *q = 0;
    if(strlen(tmpstr))
        ungetstr(tmpstr);
}

```

```

    return h;
}
/* ----- 如果是作废键则退出 ----- */
else if(h == KEY_ENTER)
{
    _ClearPrompt(1);
    return 0;
}

/* ----- 如果是向前翻页键 ----- */
else if( pageno && h == '-' )
{
    unsigned char *r = tmpline;
    int len = 0, j = 0;
    q = _CodePrompt+2;
    while( *q != '\0' )
        *r++ = *q++;
    *r = 0;
    if((q = strstr(front, tmpline)) != 0)
    {
        mid = strlen(front);
        if(mid < _Mode[_HanMode].filefront)
            mid = _Mode[_HanMode].filefront;
        front = _GetCodeBlock(mid, _Mode+_HanMode);
        q = strstr(front, tmpline);
    }
    q--;
    do
    {
        q = maxcodes;
        if(q <= front)
        {
            mid = strlen(front);
            if(mid < _Mode[_HanMode].filefront)
                mid = _Mode[_HanMode].filefront;
            front = _GetCodeBlock(mid, _Mode+_HanMode);
            q = front+strlen(front)-1;
        }
        j = 0;
        while( *q > 0xa0 || *q == '\r' || *q == '\n' )
        {
            j++;
            q--;
        }
    } while(1);
}

```

```

    }
    if(_NextDup(q-3,inputcode,maxcodes,wildchar))
        len += j;
    else
    {
        q++;
        while(ishan(q) || *q=='\r')
            q+=2;
        while(strchr(codeset,*q) || *q==' ')
            q++;
        q--;
        break;
    }
}while(len<40);
q++;
if(len>40)
{
    while(ishan(q) || *q=='\r')
        q+=2;
    q += maxcodes;
}
pageno--;
continue;
}

/* ----- 如果是向后翻页键 ----- */
else if(forward && h == '=' && forward)
{
    if(havewild)
        memcpy(_DupPrompt,_DupPrompt+i*maxcodes,maxcodes);
    i = 0;
    p = _CodePrompt;
    sprintf(p,"%d:%s",i==9? 0:i+1,tmpline);
    p += strlen(tmpline)+2;
    i++;
    forward = NO;
    pageno++;
    goto NextPage;
}

/* ----- 其他键不起作用 ----- */
else
{
    putchar(Beep);

```



```

        goto AcceptSelect;
    }
}
/* ----- 返回汉字机内码 ----- */
return h;
}
/* ----- 内部函数 _SrchCode : 查表 ----- */
static int _Cdecl _SrchCode(front, inputcode, maxcodes, wildchar)
unsigned char * front;
char * inputcode;
int maxcodes;
int wildchar;
{
    unsigned char * p = front;
    char tmpcode[13];
    int i;
    while(*p)
    {
        if(wildchar)
        {
            memcpy(tmpcode, p, maxcodes);
            tmpcode[maxcodes] = 0;
            for(i=0; i<maxcodes; i++)
                if(inputcode[i] == wildchar)
                    tmpcode[i] = wildchar;
            i = memcmp(inputcode, tmpcode, maxcodes);
        }
        else
            i = memcmp(inputcode, p, maxcodes);
        if(i > 0)
            return -1;
        if(i == 0)
            return p - front;
        p += maxcodes;
        while((*p) > 0xa0 || *p == '\r' || *p == '\n')
            p++;
        if(*p == 0)
            break;
    }
    return strlen(front) + 1;
}

/* ----- 内部函数 _GetString : 从码表中取一串 ----- */

```

```

static unsigned char * _Cdecl _GetString(ptr,tmpline)
unsigned char * ptr, * tmpline;
{
    while(ishan(ptr))
    {
        *tmpline++ = *ptr++;
        *tmpline++ = *ptr++;
    }
    *tmpline = 0;
    if( *ptr == '\r' )
        ptr += 2;
    return ptr;
}

/* ----- 内部函数 _NextDup : 取下一重码 ----- */
static int _Cdecl _NextDup(buff,code,len,wild)
unsigned char * buff;
unsigned char * code;
int len;
int wild;
{
    int i;
    for(i=0;i<len;i++)
        if(code[i] != wild && code[i] != buff[i])
            break;
    return i == len;
}

/* ----- 内部函数 _GetCodeBlock : 取代码块 ----- */
static char * _Cdecl _GetCodeBlock(offset,mode)
long offset;
MODE_TYPE * mode;
{
    unsigned char * ptr;
    long len;
    long addr;
    int i = 0;

    switch(mode->whererecodes)
    {
        case DSK:
            fseek(mode->codefile,offset,SEEK_SET);
            len = fread(_CodeBuffer,1,BUFFER_SIZE,mode->codefile);

```

```

    break;
case XMS:
    _Emb.len      = BUFFER_SIZE;
    _Emb.sour_han = mode->handle;
    _Emb.sour_off = offset;
    _Emb.dest_han = 0;
    _Emb.dest_off = FP_SEG(_CodeBuffer);
    _Emb.dest_off <<= 16;
    _Emb.dest_off += FP_OFF(_CodeBuffer);
    _MoveDataXMS();
    len = mode->filelen - offset;
    len = len > BUFFER_SIZE ? BUFFER_SIZE : len;
    break;
case EMM:
    addr = FP_SEG(_CodeBuffer);
    addr = (addr << 4) + FP_OFF(_CodeBuffer);
    _SetDestAddr(addr, BUFFER_SIZE);
    addr = mode->EMM_addr + offset;
    _SetSourAddr(addr, BUFFER_SIZE);
    _MoveDataEMM(BUFFER_SIZE);
    len = mode->filelen - offset;
    len = len > BUFFER_SIZE ? BUFFER_SIZE : len;
    break;
}
ptr = _CodeBuffer + len - 1;
while(( * ptr) > 0xa0 || * ptr == '\r' || * ptr == '\n')
    ptr--;
while(strchr(mode->codeset, * ptr) || * ptr == ' ')
    ptr--;
* (ptr - 1) = 0;
ptr = _CodeBuffer;
while(strchr(mode->codeset, * (ptr + i)) || * (ptr + i) == ' ')
    i++;
if(i < mode->maxcodes)
{
    while(strchr(mode->codeset, * ptr) || * ptr == ' ')
        ptr++;
    while(( * ptr) > 0xa0 || * ptr == '\r' || * ptr == '\n')
        ptr++;
}
return ptr;
}

```

```

/* ----- 内部函数 _OutCode : 输出代码 ----- */
static unsigned _Cdecl _OutCode(unsigned h)
{
    char tmpstr[17];
    char * ptr, * qtr;

    ptr = strchr(_CodePrompt,h)+2;
    ptr = takehan(ptr,&h);
    qtr = tmpstr;
    while(ishan(ptr))
    {
        *qtr++ = *ptr++;
        *qtr++ = *ptr++;
    }
    *qtr = 0;
    if(strlen(tmpstr))
        ungetstr(tmpstr);
    return h;
}

/* ----- 内部函数 _DispInputCode : 显示输入码 ----- */
static void _DispInputCode(int i,int len)
{
    char tmpline[17];
    memcpy(tmpline,_DupPrompt+i*len,len);
    tmpline[len] = 0;
    putxys(16,_PromptLine+7,_PmtColor,_PmtBk,tmpline);
}

```

# 光标、时钟和定时器

在设计各种数据、文本编辑软件时经常要用到光标。光标用来标明当前的编辑位置,要求醒目而又不单独占据屏幕位置。在西文文本显示方式下的光标有两种形式:一种是下划线式,一种是矩形块式。为了醒目起见,光标以一定的频率闪烁;为了不影响操作人员辨认屏幕上原来的显示内容,光标并不覆盖其背景,而是以某种“透明”的方式和背景字符叠加在一起。遗憾的是 HANENV 系统和大多数中文操作系统相同,工作于图形方式下,不能直接利用 BIOS 的文本光标。因此在本章中我们首先介绍如何设计一个可以工作于图形模式下的闪烁光标。

设计闪烁光标必然要用到机内时钟。BIOS 的时钟中断 1CH 可以直接挂接用户的应用程序,很适合于用来设计与时间有关的应用程序。因此,除了光标以外,在本章中我们还要介绍 HANENV 系统中的时钟显示和报时、定时器(闹钟)等应用 BIOS 中断 1CH 的库函数。

## 9.1 BIOS 的时钟中断 1CH

我们知道, BIOS 的 1CH 号软中断由实时时钟中断(08H)服务程序触发,每秒钟中断约 18.2 次。一般来说,该中断的服务程序中只有一条中断返回语句,因而是一个空中断。实际上,该中断正是 BIOS 为用户预留的时钟中断接口。我们可以用编写自己的中断服务程序来代替原来的 1CH 服务程序,这样我们自己的中断服务程序也会以每秒 18.2 次的频率被调用。编写中断服务程序是比较复杂的,首先要考虑的是现场保护问题,除了标志寄存器、指令指针 IP、代码段寄存器 CS 由硬件隐含操作自动压栈保护以外,其余的段寄存器和通用寄存器的内容都需要由用户自己的中断服务程序实施保护。另外,由 1CH 中断的频率可以算出两次中断之间只有 55ms 左右,这是利用 1CH 中断的服务程序的运行时间长度极限。如果 1CH 号中断服务程序运行时间超过这个期限,有可能会引起丢失时钟计数或者程序死锁等现象,因此用户服务程序必须编得短小精悍。

在 Turbo C 中,编写的用户中断服务函数可以使用修饰符 interrupt,例如:

```
void interrupt intfun()
```

```

{
    .....
}

```

使用这种格式编写的中断服务函数会自动保存和恢复中断点的现场信息,例如各寄存器的内容等,并且可以使用库函数 `servect` 安装到某个中断向量上去,使用相当方便。为了实现闪烁光标、时钟显示和报时、定时器等功能,在 HANENV 系统中我们设计了一个新的中断 1CH 的服务程序,见程序 9-1。

#### 程序 9-1 新中断 1CH 服务程序。

为了在中断 1CH 中挂接多项内容,我们定义了一个全局函数指针数组,共有 10 个分量,每个分量都可以指向一个要求挂在 1CH 中断上的功能函数:

```
void far (* _Int1CHfun[10])();
```

在 HANENV 系统的初始化函数 `init_hanenv` 中已经将该数组清零:

```
for(i=0;i<10;i++)
    _Int1CHfun[i] = NULL;
```

我们编写的新中断 1CH 的服务程序非常简单,就是逐个扫描该数组的各个分量,如果哪个分量指向一个实际的功能函数,则执行之。扫描完所有的分量之后,再执行原来的 1CH 中断服务程序。

```

/* -----
   函数 _Newint1CH : 新 1CH 中断处理
   ----- */
#include(hanenv.h)
void interrupt _Newint1CH(void)
{
    static int i;
    for(i = 0; i < 10; i++)
        if(_Int1CHfun[i])
            (* _Int1CHfun[i])();
    (* _Oldint1CH)();
}

```

编写好了中断服务程序之后,还要将其安装到相应的中断向量上去。DOS 将 0—3FFH 的内存空间开辟为中断向量区,一共可以安装 256 个中断向量,每个中断向量占用 4 个字节,内容就是中断服务程序的地址(段地址和偏移量)。安装用户自定义的中断服务程序的方法也很多,下面我们只介绍利用 Turbo C 的库函数 `getvect` 和 `setvect` 安装我们的新 1CH

中断服务程序的方法。

库函数 `getvect` 的功能为取某号中断的服务程序地址,其调用格式为

```
void interrupt (* getvect(int intr_num))();
```

参数 `intr_num` 为欲查询的中断号(在 0—255 之间),返回值是该中断号对应的中断服务程序的地址。

库函数 `setvect` 的功能是将一个中断服务程序安装到某个中断号上去,其调用格式为

```
void setvect(int intr_num, void interrupt (* isr)());
```

其中参数 `intr_num` 为欲安装的中断号, `isr` 为中断服务程序的地址。

使用以上两个库函数时应用程序中应该包含头文件 `dos.h`。在 HANENV 系统的库函数 `init_hanenv` 中我们定义了一个指向中断函数的全局指针变量 `_Oldint1CH`:

```
void interrupt far (* _Oldint1CH)();
```

用于存放原来的 1CH 中断服务程序的地址:

```
_Oldint1CH = getvect(0x1c);
```

然后再将我们编写的新的时钟中断服务程序安装到中断向量 1CH 上去:

```
setvect(0x1c, _Newint1CH);
```

## 9.2 光标与汉字的文本工作方式设计

由 9.1 节可知,我们在库函数 `init_hanenv` 中已经用自己编写的新时钟中断服务程序代替了原来的 1CH 中断服务程序。现在的问题就是如何编写能够安装到全局数组变量 `_Int1CHfun` 的某个分量上的实际执行中断功能的函数了。首先我们考虑光标函数的设计。

我们设计光标的目标是:

1. 是闪烁光标;
2. 光标与背景做“异或”运算,这样无论背景是什么颜色,都能同时看清背景和光标;
3. 光标的大小、颜色可调;
4. 可以随时打开或关闭光标;
5. 光标不应影响屏幕上其他内容的显示。

以上五条说起来容易做起来难,尤其是第 5 条。由于 HANENV 系统采用直接操作 VGA 显示卡上的显示寄存器组的方式实现屏幕输出,显然在设计中断服务程序时这些寄存器也有一个现场保护问题。Turbo C 中的中断类型函数只对段寄存器和通用寄存器实施保护,因此 VGA 显示寄存器的保护只能另想办法。实际上,由于 VGA 卡上的显示寄存器大多数采用两步操作才能完成其设置工作,第一步首先设置索引寄存器,第二步才是为指

定的寄存器赋值(参看第1章的有关内容),所以其现场保护工作非常困难。因此,我们采用了另一种方法解决这个问题。我们在 HANENV 系统的初始化函数 `init_hanenv` 中定义了一个全局变量 `_VideoBusy`,用于标志是否正在操作 VGA 卡上的显示寄存器:

```
int _VideoBusy = NO;
```

编写显示汉字、画线、屏幕拷贝等库函数时,在对显示卡寄存器操作之前首先置 `_VideoBusy` 为 YES,然后再进行屏幕显示的有关操作,然后再恢复 `_VideoBusy` 的值为 NO。在设计有显示输出的中断服务程序时,首先检查全局变量 `_VideoBusy` 的值,如果为 NO(即显示卡寄存器不忙)则继续执行中断服务的内容,否则立即退出。这样就避免了中断服务程序中的显示操作影响屏幕其他地方的显示效果。

### 程序 9-2 闪烁光标。

为了便于调整光标的颜色、大小和闪烁速度,以及记录光标的工作状态,我们在光标程序模块之首定义了一组全局静态变量:

```
/* -----
   程序模块 Cursor.c : 实现闪烁光标的函数模块
   ----- */
#include (hanenv.h)
/* ----- 全局静态变量定义: 光标的参数及工作变量 ----- */
static int _CursorStatus = 0; /* 光标状态:1 = 显示 */
static int _CursorWidth = 1; /* 光标宽度(字节) */
static int _CursorHigh = 18; /* 光标整体高度(象素) */
static int _CursorLine = 2; /* 光标线数 */
static int _CursorSpeed = 1; /* 光标速度:1 = 最快 */
static int _CursorColor = 15; /* 光标颜色 */
static int _LastCol = 0; /* 光标原来列(字节) */
static int _LastLine = 0; /* 光标原来行(象素) */
```

把这些全局变量说明为静态变量是为了避免应用程序修改它们。在光标活动期间随意修改光标参数必然会在屏幕上留下痕迹。

首先我们设计一个用于光标显示的内部函数,该函数只完成在屏幕上指定位置显示一个小光棒(光标)的任务。要注意的是光标应该以“异或”方式写向屏幕,这样如果向已有光标的地方再次写光标就会抹去光标,恢复原来的背景。以一定的频率反复写光标就会造成光标“闪烁”的现象。

该函数中调用了许多 VGA 显示卡上的显示寄存器,这些寄存器全部都在前面几章的程序中出现过。如果读者还不清楚其具体含义和操作的话,可以参看第1章的有关内容。

```
/* -----
   内部函数 _Cursor : 显示光标
   ----- */
```



```

----- */
static void _Cdecl _Cursor(int col,int line)
{
    int i,j;
    char far *addr;
    unsigned char old03,old05;

    /* ----- 置显示寄存器标志为“忙” ----- */
    _VideoBusy = YES;

    /* ----- 计算光标的显示位置 ----- */
    addr = MK_FP(0xa000,(line+_CursorHigh-1+_ScreenTop)*_ScreenWidth+
        col);

    /* -- 保存方式寄存器和数据移位与函数选择寄存器的内容 -- */
    outportb(0x3ce,0x05);
    old05 = inportb(0x3cf);
    outportb(0x3ce,0x03);
    old03 = inportb(0x3cf);

    /* ----- 设置显示寄存器组 ----- */
    outportb(0x3ce,0x05);
    outportb(0x3cf,old05&0xfc);
    outportb(0x3ce,0x03);
    outportb(0x3cf,0x18);
    outportb(0x3ce,0x00);
    outportb(0x3cf,_CursorColor);
    outportb(0x3ce,0x01);
    outportb(0x3cf,0x0f);
    outportb(0x3ce,0x08);
    outportb(0x3cf,0xff);

    /* ----- 显示光标 ----- */
    for(i=0;i<_CursorLine;i++)
    {
        for(j=0;j<_CursorWidth;j++)
            *(addr+j) &= 0xff;
        addr -= _ScreenWidth;
    }

    /* ----- 恢复显示寄存器组的省缺值 ----- */
    outportb(0x3ce,0x03);
    outportb(0x3cf,old03);

```

```

    outportb(0x3ce,8);
    outportb(0x3cf,0xff);
    outportb(0x3ce,1);
    outportb(0x3cf,0);
    outportb(0x3ce,0);
    outportb(0x3cf,0x0f);
    outportb(0x3ce,0x05);
    outportb(0x3cf,old05);
    /* ----- 翻转光标状态 ----- */
    _CursorStatus = !_CursorStatus;
    /* ----- 置显示寄存器标志为“闲” ----- */
    _VideoBusy = NO;
}

```

内部函数 `_CursorFun` 用于实现光标的闪烁和控制光标的移动。在该函数中定义了一个局部静态变量 `count`，其作用是控制光标的闪烁速度。我们知道，软中断 1CH 以每秒 18.2 次的速度被调用，如果每调用一次 1CH 中断服务程序光标就闪烁一次的话则光标闪烁过快，特别是当光标比较大时更是如此。因此我们设置了一个全局变量 `_CursorSpeed` 控制光标的闪烁速度：该变量的值越大，光标闪烁的速度就越慢。

光标的移动是通过修改全局变量 `_TextCol` 和 `_TextLine` 实现的。在 HANENV 系统中，光标总是显示在全局变量 `_TextCol` 和 `_TextLine` 所指定的位置上。在本函数模块中我们另外定义了两个静态全局变量 `_LastCol` 和 `_LastLine`，通常情况下它们的值分别与 `_TextCol` 和 `_TextLine` 的值相同。如果为了移动光标而修改了 `_TextCol` 和 `_TextLine` 的值，那么 `_CursorFun` 函数首先查看光标是否正显示在屏幕上，如果是，则在原处再写一次光标，由于我们在写光标时使用了“异或”模式，此时原来位置的光标正好消失。然后就可以在新的位置上显示光标了。此时应该修改变量 `_LastCol` 和 `_LastLine` 的值使其与变量 `_TextCol` 和 `_TextLine` 保持一致。

```

/* -----
    内部函数 _CursorFun : 实现光标的闪烁功能
    ----- */
void far _CursorFun(void)
{
    static count = 0;

    count++;
    if(count > _CursorSpeed && !_VideoBusy && !_MouseBusy)
    {
        disable();

        /* ----- 如果光标位置有变动则移动光标 ----- */

```

```

    if(_TextCol != _LastCol || _TextLine != _LastLine)
    {
        if(_CursorStatus)
            _Cursor(_LastCol, _LastLine);
        _LastCol = _TextCol;
        _LastLine = _TextLine;
    }
    count = 0;
    _Cursor(_TextCol, _TextLine);
    enable();
}
}

```

上述光标控制函数还需要和全局函数指针数组 `_Int1CHfun` 的某个分量联系起来才能真正实现闪烁的光标。下列函数用于在屏幕上指定位置显示一个闪烁光标：

```

/* -----
   函数 lightcursor : 打开光标显示开关
   ----- */
void _Cdecl lightcursor(void)
{
    _Int1CHfun[1] = _CursorFun;
}

```

移动光标可以使用函数 `movecursor`。实际上, `movecursor` 是一个定义于头文件 `hanenv.h` 中的带参宏：

```
#define movecursor(x,y) (_TextCol=(x), _TextLine=(y))
```

下面的 `delightcursor` 函数用于关闭光标。此时也要检查光标的状态以决定是否要再写一次光标。

```

/* -----
   函数 delightcursor : 关闭光标显示开关
   ----- */
void _Cdecl delightcursor(void)
{
    _Int1CHfun[1] = 0;
    if(_CursorStatus)
        _Cursor(_TextCol, _TextLine);
}

```

下面的 `setcursor` 用于设置光标的大小、速度和颜色。为了不在屏幕上留下痕迹,设置光标的参数应在光标关闭期间进行。

```
/* -----
   函数 setcursor : 设置光标的形状、颜色和大小等参数
   ----- */
void _Cdecl setcursor(width,high,line,speed,color)
int width;           /* 光标宽度(字节)      */
int high;            /* 光标整体高度(像素) */
int line;            /* 光标线数          */
int speed;           /* 光标速度:1=最快    */
int color;           /* 光标颜色           */
{
    _CursorWidth = width;
    _CursorHigh  = high;
    _CursorLine   = line;
    _CursorSpeed  = speed;
    _CursorColor  = color;
}
```

应该注意的是如果要在光标所在位置显示汉字和图形时一定要先关闭光标,待显示完成以后再打开光标。因为在光标位置覆盖其他图形会使变量 `_CursorStatus` 所记录的光标状态不准确,从而可能造成在光标移动后在原来的位置上留下光标的影子。

### 9.3 正文工作方式

在设计文本编辑器、文件阅读器或数据编辑等方面的程序时,往往需要一种正文工作方式。这种工作方式以光标为核心,用户的操作都在光标处进行,满行时的折行和满屏时的滚屏都是自动处理的。在本节中我们介绍 HANENV 系统的正文工作方式的设计、构成和编程。

HANENV 系统的正文工作方式由光标、正文窗口和一组正文方式汉字显示函数构成。在上一节中我们已经详细地介绍了闪烁光标的设计与编程,因此下面我们首先介绍 HANENV 系统的正文窗口。

在库函数 `init_hanenv` 中定义了一组用于说明正文窗口界限的全局变量:

```
int _TextWinLeft   = 0;    /* 正文窗口左边界 */
int _TextWinTop    = 0;    /* 正文窗口上边界 */
int _TextWinRight  = 79;   /* 正文窗口右边界 */
int _TextWinBottom = 480;  /* 正文窗口下边界 */
```

即初始窗口的大小等于屏幕大小。在程序中可以通过函数 `set_window` 改变正文窗口的大小,也可以使用函数 `get_window` 了解当前窗口的设置情况;这两个函数都是定义于头文

件 hanenv.h 中的带参宏:

```
#define set_window(l,t,r,b) (_TextWinLeft=(l),_TextWinTop=(t),\
                             _TextWinRight=(r),_TextWinBottom=(b))
#define get_window(l,t,r,b) (l=_TextWinLeft,t=_TextWinTop,\
                             r=_TextWinRight,b=_TextWinBottom)
```

HANENV 的正文状态汉字显示函数以窗口为边界。当被显示的字符串已经到达窗口右边界时会自动回车换行;当被显示的字符串已经到达窗口右下角时,整个窗口的内容会向上滚一行,然后接着显示字符串的剩余部分。不过,正文窗口对于在第6章中介绍的 drawxy...、outxy...和 putxy...等定位汉字显示函数族不起作用。

由于正文方式下通常要进行大量汉字显示工作,所以汉字显示函数的速度很重要。为了提高显示速度,正文状态的汉字显示函数均使用调用 \_OutF 或 \_OutFont 函数的方法实现。HANENV 系统中的正文状态汉字显示函数族包括下列函数:

```
void outc(int ch);
void outchar(int ch);
void outh(unsigned han);
void outhan(unsigned han);
void outh1(unsigned h,int whichhalf);
void outhan1(unsigned h,int whichhalf);
void outs(char *string);
void outstr(char *string);
void putnstr(char *string,int width);
```

其中前四个是定义于头文件 hanenv.h 中的带参宏,其定义可以在第14章“HANENV系统的头文件”中找到;第4个和第5个函数分别用于带或不带放大倍数地显示半个汉字,主要用于字符串显示时位于窗口边界的半个汉字的显示;最后三个函数用于在正文窗口中光标处显示一个字符串。由它们的函数名即可看出:outs 用于快速显示字符串,不带放大倍数;outstr 用于放大显示字符串;putnstr 用于显示给定字符串的前 n 个字符,可以带放大倍数,并且在显示之前会清除显示区域下面原来的屏幕显示内容。

由上面这些正文方式下的汉字显示函数的说明可以看出,这些函数中都没有关于显示位置和颜色的参数。前面我们已经说过,在正文方式下汉字和字符在光标处显示,而光标位置实际上由全局变量 \_TextCol 和 \_TextLine 确定,在应用程序中可以通过函数(宏) movecursor 修改,例如:

```
movecursor(20,0);
```

将光标移至(20,0)处。另外,汉字和字符的显示颜色由全局变量 \_TextColor 确定,函数 putnstr 清屏幕背景所使用的颜色由全局变量 \_Background 确定。这两个全局变量的定义

在函数 `init_hanenv` 中:

```
int _TextColor = LIGHTGRAY;
int _Background = BLACK;
```

这两个变量的值可以通过函数 `set_text_color` 和 `set_background` 修改。实际上,这两个函数是定义于头文件 `hanenv.h` 中的带参宏:

```
#define set_text_color(color) (_TextColor=(color))
#define set_background(color) (_Background=(color))
```

下面我们通过函数 `outs` 说明正文方式的字符串显示函数的设计方法。

### 程序 9-3 在正文方式下输出一个字符串。

在正文方式下输出字符串时要考虑两个问题:一是调整光标的位置;一是如何将输出局限在正文窗口中。其实这两个问题是紧密联系在一起的:如果光标是在窗口的中间,则在显示了字符串中的一个汉字或字符之后,光标向右移动一个汉字或字符的距离;如果在显示了一个汉字或字符之后光标的位置恰好位于窗口右边界之外,则光标应该移至窗口中下一行的最左端;如果已经是在窗口中的最后一行,光标无法下移时,则应该首先将窗口中的所有内容向上翻滚一行,即原来的第二行覆盖第一行的内容,第三行覆盖第二行的内容,……,最后一行覆盖倒数第二行的内容,腾出来的最后一行的内容用 `_Background` 指定的颜色清除为一个空行,然后再将光标移到该行之首(最左端)。

当光标已经到达窗口的右边界,即当前行中只能显示得下一个字符(半个汉字)时,当前要显示内容却是一个汉字,事情还要麻烦些。首先我们要确定此时的处理原则。传统的处理方法是将整个汉字都移到下一行显示,而在当前行末留下一个空格。但是这样处理有一个问题,就是无法区别该空格是因为显示不下汉字而产生的还是原来字符串中就有的。所以我们采用了另一种方式,在只能显示半个汉字的地方就只显示半个汉字,另外半个汉字放在下一行的开始显示。这样就完全避免了上面出现的二义性,而且窗口显示也显得比较整齐。至于将一个汉字分成两半显示,一般情况下并不会影响对该汉字的识别。

```
/* -----
   程序 outs : 以正文方式输出一个字符串
   ----- */
#include <hanenv.h>

void _Cdecl outs(string)
char * string;          /* 待显示的字符串 */
{
    unsigned h;
    int width           = _CurrentHZK->fontwidth;
```

```

int high          = _CurrentHZK->fonthigh+2;
int half          = NO;
int have_cursor   = havecursor();

/* ----- 关闭光标 ----- */
if(have_cursor)
    delightcursor();

/* -- 循环:顺序显示字符串中的每一个汉字或 ASCII 码 -- */
while((string=takehan(string,&h))!=NULL)
{

    /* ----- 如果是 ASCII 码 ----- */
    if(h<256)
    {

        /* ----- 显示 ASCII 字符 ----- */
        outc(h);

        /* -- 调整光标位置:未到窗口右边界则光标右移 -- */
        if(_TextCol<_TextWinRight)
            _TextCol++;

        /* ----- 已到窗口右边界光标下移一行 ----- */
        else
        {
            _TextCol = _TextWinLeft;

            /* -- 如果不是窗口最下一行则光标下移一行 -- */
            if(_TextLine<_TextWinBottom)
                _TextLine += high;

            /* ----- 否则整个窗口的内容上移一行 ----- */
            else
                scroll_up(high);
        }
    }

    /* ----- 如果是汉字或全角字符或图形符号 ----- */
    else if(isSP(h))
    {
        _TextLine++;
    }
}

```

```

/* ----- 如果只能显示半个汉字 ----- */
if(_TextCol == _TextWinRight)
{
    takefont(h);
    outh1(0);
    half = ! half;
}
else
    outh(h);

/* -- 调整光标位置:未到窗口右边界则光标右移 ----- */
_TextCol += width;
_TextLine--;

/* ----- 已到窗口右边界光标下移一行 ----- */
if(_TextCol > _TextWinRight)
{
    _TextCol = _TextWinLeft;

    /* ----- 如果不是窗口最下一行则光标下移一行 ----- */
    if(_TextLine < _TextWinBottom)
        _TextLine += high;

    /* ----- 否则整个窗口的内容上移一行 ----- */
    else
        scroll_up(high);

    /* ----- 如果上一行最后还剩半个汉字 ----- */
    if(half)
    {
        _TextLine++;
        outh1(1);
        _TextLine--;
        _TextCol -= width/2;
        half = ! half;
    }
}

/* ----- 打开光标 ----- */
if(have_cursor)
    lightcursor();
}

```



在上面的 outs 函数中为了将字符串的显示局限在正文窗口中,还调用了两个库函数:用于显示半个汉字的函数的 outh1 和用于窗口上滚的函数 scroll\_up。下面我们首先介绍函数 outh1 的设计方法。

#### 程序 9-4 在正文方式下显示半个汉字。

其实函数 \_OutF 阅读函数 outh1 就可以发现,显示半个汉字的关键在于读字模缓冲区时要正确地读出所需要的字节。

```

/* -----
   程序 outh1 : 显示半个汉字
   ----- */
#include (hanenv.h)

void _Cdecl outh1(whichhalf)
int whichhalf;      /* 哪半个汉字:0=前半个汉字,1=后半个汉字 */
{
    int i,j;
    int width        = _CurrentHZK->fontwidth/2;
    unsigned char *p  = _HanFont;
    char far *addr    = MK_FP(0xa000,(_TextLine+_ScreenTop)*_ScreenWidth
                               +_TextCol);

    _VideoBusy = YES;

    /* ----- 设置 VGA 的显示寄存器 ----- */
    outportb(0x3ce,0);
    outportb(0x3cf,_TextColor);
    outportb(0x3ce,1);
    outportb(0x3cf,0x0f);
    if(whichhalf)
        p += width;
    for(i=0;i<_CurrentHZK->fonthigh;i++)
    {
        for(j=0;j<width;j++)
        {
            outportb(0x3ce,0x08);
            outportb(0x3cf,*p++);
            *(addr+j) &= 0xff;
        }
        p += width;
        addr += _ScreenWidth;
    }
}

```

```

    }

    /* ----- 恢复被修改的寄存器 ----- */
    outportb(0x3ce,8);      /* 恢复位屏蔽寄存器      */
    outportb(0x3cf,0xff);   /* 恢复位屏蔽寄存器      */
    outportb(0x3ce,1);      /* 恢复允许置位/复位寄存器 */
    outportb(0x3cf,0);      /* 恢复允许置位/复位寄存器 */
    outportb(0x3ce,0);      /* 恢复置位/复位寄存器      */
    outportb(0x3cf,0x0f);   /* 恢复置位/复位寄存器      */
    _VideoBusy = NO;
}

```

HANENV 系统中一共有两个函数用于移动文本窗口的内容：上滚函数 `scroll_up` 和下滚函数 `scroll_down`。下面我们以上滚函数 `scroll_up` 为例介绍这类函数的设计方法。

#### 程序 9-5 正文窗口上滚函数。

该函数的实现方法其实很简单，首先利用 `_MoveImage` 函数将文本窗口中从第二行开始直到最后一行的部分拷贝到从第一行开始的地方，然后再用 `_Block` 函数将文本窗口的最后一行内容清除。

```

/* -----
   函数 scroll_up : 正文窗口上滚
   ----- */
#include<hanenv.h>

void _Cdecl scroll_up(high)
int high;          /* 正文窗口上滚线数      */
{
    _MoveImage(_TextWinLeft, _TextWinTop+high, _TextWinRight
               - _TextWinLeft+1, _TextWinBottom - _TextWinTop, _TextWinLeft,
               _TextWinTop);
    _Block(_TextWinLeft, _TextLine, _TextWinRight - _TextWinLeft+1, high, _Background);
}

```

用于文本方式的函数还有一些，其设计方法大同小异，我们就不一一举例了。只是要注意 HANENV 系统的文本方式并不是一个和图形工作方式相对应、相排斥的独立的显示模式，而只是为了方便设计诸如文字编辑器等需要文本窗口的应用程序而提供的一组全局变量和库函数。这些库函数完全可以和 HANENV 系统的其他库函数同时使用，相得益彰。实际上，就以文本编辑器为例，正文编辑部分可以使用本章介绍的文本方式处理函数，而封面、编辑提示和菜单当然还是要使用坐标定位的汉字显示函数。

## 9.4 时钟与定时器

在屏幕上显示一个实时时钟,不仅实用,而且能够美化应用程序的用户界面。我们设计的时钟函数非常灵活,可以指定其颜色、字体大小、在屏幕上的显示位置等参数,并具有整点和半点报时功能。另外,我们的时钟还带有“闹钟”功能,可以设置多达100项的定时执行的事件。HANENV系统的工具软件hantool就带有一个这样的时钟,见图5-2中hantool版面的右上角。

时钟模块的设计思路和光标模块很相似。一个时钟显示内部函数\_ClockFun,处理时钟的显示、整点和半点报时以及定时器事件的触发。该函数必需通过运行函数run\_clock才能和全局数组变量\_IntlCHfun的第0个分量联系起来。为了使定时器功能独立于时钟的显示,另外编写了两个函数light\_clock和close\_clock,分别用于打开和关上时钟的显示。和设置光标的方法类似,函数set\_clock用于修改时钟显示的各项参数。最后,函数set\_alarm和del\_alarm用于设置和取消一个定时事件。

### 程序 9-6 时钟和定时器程序模块。

首先我们定义了一组说明时钟和定时器工作状态的全局静态变量。请注意为这些变量所赋的初值。如果不用set\_clock函数修改这些参数,那么执行run\_clock以后将在屏幕左上角以正常字体显示一个黑底白字的数字式时钟。

```
/* -----
   函数模块 clock.c 时钟和定时器模块
   ----- */
#include (hanenv.h)
#include (alloc.h)

/* -- 局部静态变量定义: 时钟和定时器的参数及工作变量 -- */
static int    _ClockColor  = WHITE;    /* 时钟数码显示色 */
static int    _ClockBk    = BLACK;     /* 时钟背景显示色 */
static int    _ClockCol    = 0;        /* 时钟显示列(字节) */
static int    _ClockLine   = 0;        /* 时钟显示行(象素) */
static int    _ClockXtimes = 1;        /* 时钟横向放大倍数 */
static int    _ClockYtimes = 1;        /* 时钟纵向放大倍数 */
static int    _ClockDisp   = YES;      /* 时钟是否显示标志 */
static int    _AlarmCount  = 0;        /* 定时器个数 */
static ALARM *_Alarm[100];             /* 定时器队列 */
static struct time _Clock;             /* 时钟变量 */
static struct date _Calendar;          /* 日历变量 */
```

我们注意到在上述全局静态变量的定义中出现了一个新的类型ALARM,其定义可以

在头文件 `hanenv.h` 中找到:

```
typedef struct                                /* 定时器类型的定义 */
{
    struct date d;                            /* 日期 */
    struct time t;                            /* 时间 */
    int tag;                                  /* 定时器激活方式 */
    unsigned (*fun)();                        /* 事件函数 */
}ALARM;
```

即 `ALARM` 类型的变量说明了一个定时器事件:定时器被激活的日期、时间和方式,以及该定时器被激活后应该执行的功能(即事件函数的功能)。其中定时器激活方式可以取下列值:

```
PER_SECOND  —— 每秒激活
PER_MINATE  —— 每分钟激活
PER_HOUR    —— 每小时激活
PER_DAY     —— 每日激活
PER_MONTH   —— 每月激活
PER_YEAR    —— 每年激活
ONCE_ONLY   —— 只在指定时间激活一次
```

内部函数 `_ClockFun` 用于处理时钟的显示、逢整点和半点报时以及定时器的触发。报时使用 Turbo C 的库函数 `sound` 和 `nosound`。函数 `sound` 使机内扬声器以指定的频率发声,函数 `nosound` 关闭扬声器。需要注意的是函数 `sound` 只是启动了机内可编程定时器芯片 8253,一旦启动,扬声器即按指定的频率发声,直到使用函数 `nosound` 关闭扬声器。在扬声器发声期间 CPU 仍可进行其他工作,即发声相当于“后台”工作。

```
/* -----
   内部函数 _ClockFun : 处理时钟显示和整点报时
   ----- */
void far _ClockFun(void)
{
    static int time = 0, count = 0;
    char info[8];
    int i;

    /* ----- 处理整点报时 ----- */
    if(_Clock.ti_min == 59 && _Clock.ti_sec >= 56)
    {
        if(time == 0)
            sound(800);
        if(time == 6)
```

```

        nosound();
    }

    /* -----处理半点报时 ----- */
    if((_Clock.ti_min==0 || _Clock.ti_min==30) && _Clock.ti_sec==0)
    {
        if(time==0)
            sound(2000);
        if(time==6)
            nosound();
    }

    /* -----每次中断计数值加一 ----- */
    time++;

    /* ----- 计数值满 18 则秒值加一 ----- */
    if(time>=18)
    {
        count++;
        _Clock.ti_sec++;

        /* ----- 如果满 60 秒钟分值加一 ----- */
        if(_Clock.ti_sec>=60)
        {
            _Clock.ti_sec=0;
            _Clock.ti_min++;

            /* ----- 如果满 60 分钟小时值加一 ----- */
            if(_Clock.ti_min>=60)
            {
                _Clock.ti_min = 0;
                _Clock.ti_hour =_Clock.ti_hour>=23? 0:_Clock.ti_hour+1;
            }
        }

        /* ----- 修改时钟的显示 ----- */
        if(_ClockDisp && !_VideoBusy)
        {
            /* ----- 保存原来的放大倍数和显示寄存器值 ----- */
            int old_xt = _Xtimes;
            int old_yt = _Ytimes;
            unsigned char old03,old05;

            /* ----- 修改放大倍数为时钟显示放大倍数 ----- */
            _Xtimes = _ClockXtimes;

```

```

_Ytimes = _ClockYtimes;

/* ----- 修改显示寄存器 ----- */
outportb(0x3ce,0x05);
old05 = inportb(0x3cf);
outportb(0x3ce,0x03);
old03 = inportb(0x3cf);
outportb(0x3ce,0x03);
outportb(0x3cf,0x00);
outportb(0x3ce,0x05);
outportb(0x3cf,old05&0xfc);
info[0] = _Clock.ti_hour/10+'0';
info[1] = _Clock.ti_hour%10+'0';
info[2] = ':';
info[3] = _Clock.ti_min/10+'0';
info[4] = _Clock.ti_min%10+'0';
info[5] = ':';
info[6] = _Clock.ti_sec/10+'0';
info[7] = _Clock.ti_sec%10+'0';
/* ----- 显示时钟值 ----- */
for(i=0;i<8;i++)
{
    _Block(_ClockCol+i*_ClockXtimes,_ClockLine,_ClockXtimes,
           _ClockYtimes*CHAR_HIGH,_ClockBk);
    outxychar(_ClockCol+i*_ClockXtimes,_ClockLine,_ClockColor,info[i]);
}
outportb(0x3ce,0x03);
outportb(0x3cf,old03);
outportb(0x3ce,0x05);
outportb(0x3cf,old05);
_Xtimes = old_xt;
_Ytimes = old_yt;
}

/* -- 如果某定时器到期则触发之 ----- */
for(i=0;i<_AlarmCount;i++)
{
    if(_Alarm[i]->tag>0 && _Alarm[i]->t.ti_sec!=_Clock.ti_sec)
        continue; /* per minate */
    if(_Alarm[i]->tag>1 && _Alarm[i]->t.ti_min!=_Clock.ti_min)
        continue; /* per hour */
    if(_Alarm[i]->tag>2 && _Alarm[i]->t.ti_hour!=_Clock.ti_hour)
        continue; /* per day */
    if(_Alarm[i]->tag>3 && _Alarm[i]->d.da_day!=_Calendar.da_day)

```

```

        continue; /* per month */
    if(_Alarm[i]->tag>4 && _Alarm[i]->d.da_mon!= _Calendar.da_mon)
        continue; /* per year */
    if(_Alarm[i]->tag>5 && _Alarm[i]->d.da_year!= _Calendar.da_year)
        continue; /* once only */
    /* ----- 触发器 0 是专为定时器保留 ----- */
    _Triggers[0].trigger = _Alarm[i]->fun;
    ungeth(1);
}
/* ----- 修正秒误差 ----- */
if(count == 5)
{
    count = 0;
    time = -1;
}
else
    time = 0;
}
}

```

上述函数最后有一段“修正秒误差”，用于修正按 18 次中断算 1 秒钟这样简单计数所造成的时间误差。

函数 run\_clock 用于将时钟处理函数 ClockFun 的地址赋给全局数组变量 \_Int1CHfun 的第 1 个分量。只有这样，时钟处理函数才能作为新的中断 1CH 的服务程序运行。

```

/* -----
   函数 run_clock : 激活时钟
   ----- */

void _Cdecl run_clock(void)
{
    getdate(&_Calendar);
    gettime(&_Clock);
    _Int1CHfun[0] = _ClockFun;
}

```

函数 set\_clock 用于修改时钟显示的位置、大小和颜色。为了保证屏幕显示效果，这种修改应该在时钟显示被关闭的情况下进行。

```

/* -----
   函数 set_clock : 设置时钟显示模式
   ----- */

```

```

void _Cdecl set_clock(col,line,xtimes,ytimes,color,bk)
int col;                /* 时钟显示列(字节) */
int line;               /* 时钟显示行(像素) */
int xtimes;             /* 时钟横向放大倍数 */
int ytimes;             /* 时钟纵向放大倍数 */
int color;              /* 时钟数码显示色 */
int bk;                 /* 时钟背景显示色 */
{
    _ClockColor = color;
    _ClockBk    = bk;
    _ClockCol   = col;
    _ClockLine  = line;
    _ClockXtimes = xtimes;
    _ClockYtimes = ytimes;
}

```

函数 light\_clock 用于打开时钟显示开关。

```

/* -----
   函数 light_clock : 显示时钟
   ----- */
void _Cdecl light_clock(void)
{
    _ClockDisp = YES;
}

```

函数 close\_clock 用于关闭时钟显示开关。

```

/* -----
   函数 close_clock : 关闭时钟显示
   ----- */
void _Cdecl close_clock(void)
{
    _ClockDisp = NO;
}

```

函数 set\_alarm 用于设置定时器。在设置定时器之前首先应该编写定时器事件函数,其内容可以是显示一段信息,发出某种声音,或者是其他处理如文件存盘等。

```

/* -----
   函数 set_alarm : 设置定时器
   ----- */

```



```

int _Cdecl set_alarm(date,time,type,fun)
struct date date;           /* 定时日期          */
struct time time;           /* 定时时间          */
int type;                   /* 定时器类型        */
unsigned (*fun)();          /* 定时器操作函数指针 */
{
    int alarm_no = NO;
    if((_Alarm[_AlarmCount]==malloc(sizeof(ALARM)))!=NULL)
    {
        _Alarm[_AlarmCount]-->d      = date;
        _Alarm[_AlarmCount]-->t      = time;
        _Alarm[_AlarmCount]-->tag    = type;
        _Alarm[_AlarmCount]-->fun    = fun;
        alarm_no                      = ++_AlarmCount;
    }
    return alarm_no;
}

```

为了让应用程序设计人员设计一个可以动态编辑的定时器,我们编写了删除定时器事件的函数。

```

/* -----
   函数 del_alarm : 删除一个定时器
   ----- */
void _Cdecl del_alarm(int alarm_no)
{
    if(alarm_no>0 && alarm_no<=_AlarmCount)
    {
        free(_Alarm[alarm_no-1]);
        for(;alarm_no<=_AlarmCount;alarm_no++)
            _Alarm[alarm_no-1]=_Alarm[alarm_no];
        _AlarmCount--;
    }
}

```

## 9.5 鼠标、光标和时钟函数应用小结

鼠标光标、文本光标和时钟的显示都是由时钟中断控制的,它们都有强行显示的共同特点。在编写应用程序时要充分注意这种情况。例如,直接向鼠标光标所在区域写字符、画图形会覆盖鼠标光标,但是只要移动鼠标则鼠标光标就会重新出现;为了防止用户一时找不到鼠标光标而感到困惑,在设计应用程序时应养成先关闭鼠标光标再进行屏幕显示工作的习惯。事实上,我们在设计 HANENV 系统的大型功能函数如菜单、代码表、全屏幕数据编

辑等时就是这样做的。

如果在文本光标上覆盖其他内容,问题就更严重了。HANENV 系统是通过翻转全局变量 `CursorStatus` 的状态来控制光标显示的,在光标上覆盖其他内容会改变光标的显示状态但并未同时修改该变量的值,因此在移动光标之后会在原处留下一个光标的痕迹,相当难看。尽管在时钟上覆盖其他内容的后果没有这么严重(时钟通常不会移动),但是时钟的再次显示却会破坏覆盖于时钟上面的内容。因此,在应用程序中要注意不要在时钟上覆盖其他显示内容以及如果要在文本光标处进行显示工作的话应该首先关闭光标,当显示工作完成以后再打开光标。

## 第 3 篇

# 用户界面程序设计

## 屏 幕 作 图

由于 WINDOWS 的流行,使得窗口式图形用户界面在基于 DOS 的应用程序中也流行起来。HANENV 系统提供了丰富的作图手段和可以直接用于窗口式图形用户界面设计的组件,如点、直线和曲线、矩形块、框以及按键式和图标式菜单、滚动条、提问框、代码表等。

### 10.1 HANENV 系统和 Turbo C 的图形库联合编程

HANENV 系统可以和 Turbo C 中的图形库联合使用。一般来说,打开和关闭图形库和 HANENV 系统时最好按程序 10-1 中的顺序执行。

**程序 10-1** HANENV 系统和 Turbo C 的图形库的打开和关闭的顺序。

```
/* -----
   程序举例：HANENV 系统和 Turbo C 图形库的打开和关闭的顺序
   ----- */
#include <hanenv.h>
#include <graphics.h>

main()
{
    int driver = DETECT;
    int mode = 0;

    /* ---*/ 初始化 HANENV 系统 -----
    init_hanenv();

    /* ---*/ 初始化 Turbo C 的图形库 -----
    initgraph(&driver,&mode,"\\tc");
```

```

...

/* ---*/ 关闭 Turbo C 的图形库 ---
closegraph();

/* ---*/ 关闭 HANENV 系统 ---
close_hanenv();
}

```

关于 Turbo C 的图形库的装载及其库函数的应用请参看 Turbo C 的有关手册。必须说明的是 Turbo C 的图形库中的绘图函数大多只能工作于正常屏幕范围,即(0,0)至(639,479)之间,所以如果我们使用了 set\_screen\_width 函数和 set\_screen\_high 函数设置了一个大于实际屏幕的虚屏时,Turbo C 中的绘图函数就不适用了。此时只能使用 HANENV 系统提供的各种绘图函数。

## 10.2 横竖线、框和矩形块

在所有绘图函数中最重要的就是画点函数,我们已经在第6章中介绍过。实际上,在制作图形用户界面时最有用的绘图函数是画横线和竖线的函数。为了提高画横线和竖线的速度,我们对这两种应用单独编程,没有使用可以画任意角度的直线函数。

### 程序 10-2 画横线。

画横线时我们使用直接对 VGA 的显示寄存器和 VRAM 操作的方式进行。由于 VGA 图形模式 12H 中 VRAM 与屏幕像素对应关系比较特殊(见第1章),所以在画横线时我们分以下几种情况考虑:

1. 如果线很短,可以在一个字节中画出;
2. 如果线的两端跨字节,要分别考虑两端不满一个字节部分的输出;
3. 如果线足够长,线的中段按字节画出。

我们在画线函数中设计了一个线型参数 pattern,用来表示所画出的线是实线还是虚线。该参数是一个字节,如果某位为1则显示一点,否则为空。例如,该参数的所有位均为1(即参数值为 0xff)则画出一条实线,而如果该参数值为 0xaa 或 0x55 则画出一条虚线。当然,也可以将参数 pattern 设置为其他值,例如 0xfa,那么所画出的就是一条点划线。

```

/* ---
    函数 hline : 横线
    --- */
#include <hanenv.h>
void _Cdecl _H_Line(x,y,len,color,pattern)
int x;          /* 横线左端列坐标(以像素为单位) */
int y;          /* 横线左端行坐标(以像素为单位) */

```

```

int len;                /* 横线长度(以像素为单位) */
int color;              /* 线颜色 */
unsigned char pattern;  /* 线型 */
{
    register char far *addr;
    unsigned mask=pattern;
    register int i;

    /* ---- 计算线左端在显示存储器中的地址 ---- */
    /* -- 置显示寄存器组标志为“忙” ---- */
    _VideoBusy = YES;

    /* -- 设置显示寄存器组 ---- */
    outportb(0x3ce,0);
    outportb(0x3cf,color);
    outportb(0x3ce,1);
    outportb(0x3cf,0x0f);

    /* ---- 如果线局限于一个字节内则直接输出 ---- */
    if( x%8 && (x+len)/8 == x/8)
    {
        mask = (pattern>>x%8) & (pattern<<(8-(x+len)%8));
        outportb(0x3ce,0x08);
        outportb(0x3cf,mask);
        * (addr) &= 0xff;
    }
    else
    {
        /* ---- 如果线左端不满一个字节 ---- */
        if(x%8)
        {
            mask=pattern>>x%8;
            outportb(0x3ce,0x08);
            outportb(0x3cf,mask);
            * (addr)&= 0xff;
            addr++;
            len -= 8-x%8;
            x=x/8*8+8;
        }

        /* ---- 如果线长大于1个字节 ---- */
        if((x+len)/8 > x/8)
        {

```

```

        outportb(0x3ce,0x08);
        outportb(0x3cf,pattern);
        for(i=0;i<len/8;i++)
            *(addr++)&=0xff;
        len+=len/8*8;
    }

    /* ----- 如果线尾不满1个字节 ----- */
    if((x+len)%8)
    {
        mask=(pattern<<(8-(x+len)%8));
        outportb(0x3ce,0x08);
        outportb(0x3cf,mask);
        *(addr)&=0xff;
    }
}

/* ----- 恢复显示寄存器组的值 ----- */
outportb(0x3ce,8);
outportb(0x3cf,0xff);
outportb(0x3ce,1);
outportb(0x3cf,0);
outportb(0x3ce,0);
outportb(0x3cf,0x0f);

/* ----- 恢复显示寄存器组标志为“闲” ----- */
_VideoBusy = NO;
}

```

### 程序 10-3 画竖线。

画竖线比较简单,只需注意正确计算其在显示存储器 VRAM 中的地址,并设置好位屏蔽寄存器即可。

```

/* -----
    函数 _V_Line : 画竖线
    ----- */
#include <hanenv.h>

void _Cdecl _V_Line(x,y,len,color,pattern)
int x;                /* 横线左端列坐标(以象素为单位) */
int y;                /* 横线左端行坐标(以象素为单位) */
int len;              /* 横线长度(以象素为单位) */

```

```

int color;          /* 线颜色 */
unsigned char pattern; /* 线型 */
{
    register char far *addr;
    register int i;
    unsigned char mask=0x80 >> x%8;

    /* ----- 计算线左端在显示存储器中的地址 ----- */
    addr = MK_FP(0xa000,(y+_ScreenTop)*_ScreenWidth+x/8);

    /* ----- 置显示寄存器组标志为“忙” ----- */
    _VideoBusy = YES;

    /* -- 设置显示寄存器组 ----- */
    outportb(0x3ce,0);
    outportb(0x3cf,color);
    outportb(0x3ce,1);
    outportb(0x3cf,0x0f);
    outportb(0x3ce,8);
    outportb(0x3cf,mask);

    /* -- 画竖线 ----- */
    for(i=0;i<len;i++)
    {
        if(pattern & (0x80 >> i%8))
            *(addr) &= 0xff;
        addr += _ScreenWidth;
    }

    /* ----- 恢复显示寄存器组的值 ----- */
    outportb(0x3ce,8);
    outportb(0x3cf,0xff);
    outportb(0x3ce,1);
    outportb(0x3cf,1);
    outportb(0x3ce,0);
    outportb(0x3cf,0x0f);

    /* ----- 恢复显示寄存器组标志为“闲” ----- */
    _VideoBusy = NO;
}

```



## 程序 10-4 画矩形。

画矩形只需调用画横线和画竖线的函数画出边框即可。

```

/* -----
   函数 draw_rectangle : 画矩形
   ----- */
#include (hanenv.h)

void _Cdecl draw_rectangle(x,y,width,high,color,pattern)
int x;           /* 矩形左上角列坐标(以像素为单位) */
int y;           /* 矩形左上角行坐标(以像素为单位) */
int width;       /* 矩形宽度(以像素为单位) */
int high;        /* 矩形高度(以像素为单位) */
int color;       /* 矩形颜色 */
unsigned char pattern; /* 矩形线型 */
{
    _H_Line(x, y,width,color,pattern);
    _V_Line(x, y, high,color,pattern);
    _V_Line(x+width-1, y, high,color,pattern);
    _H_Line(x,y+high-1,width,color,pattern);
}

```

以上几个函数与屏幕背景的作用可以通过函数 set\_write\_mode 改变。通常写模式被设置为覆盖类型,即所画之线覆盖屏幕上原来的背景,也可以设置为其他类型,例如异或类型等。

## 程序 10-5 画矩形块。

该函数用于在屏幕上指定位置显示一个矩形块。和画线函数类似,该函数也有一个参数 pattern,用于确定所画矩形的图案。恰当设置该函数可以绘制出精巧漂亮的背景。该参数共由 4 个字节构成,其意义(由高位字节向低位字节数)分别为横向线型、纵向线型、纵向移位线、移位位数。例如,参数值 0xffff0000 显示实心矩形,而图 10-1 显示了几中常用的图案,可以用作背景花纹。

```

/* -----
   函数 _Bar : 在屏幕上指定位置显示一个矩形块
   ----- */
#pragma inline
#include (hanenv.h)

/* ----- 内部函数 ror : 对字节进行循环移位 ----- */
static char _Cdecl ror(byte,n)

```

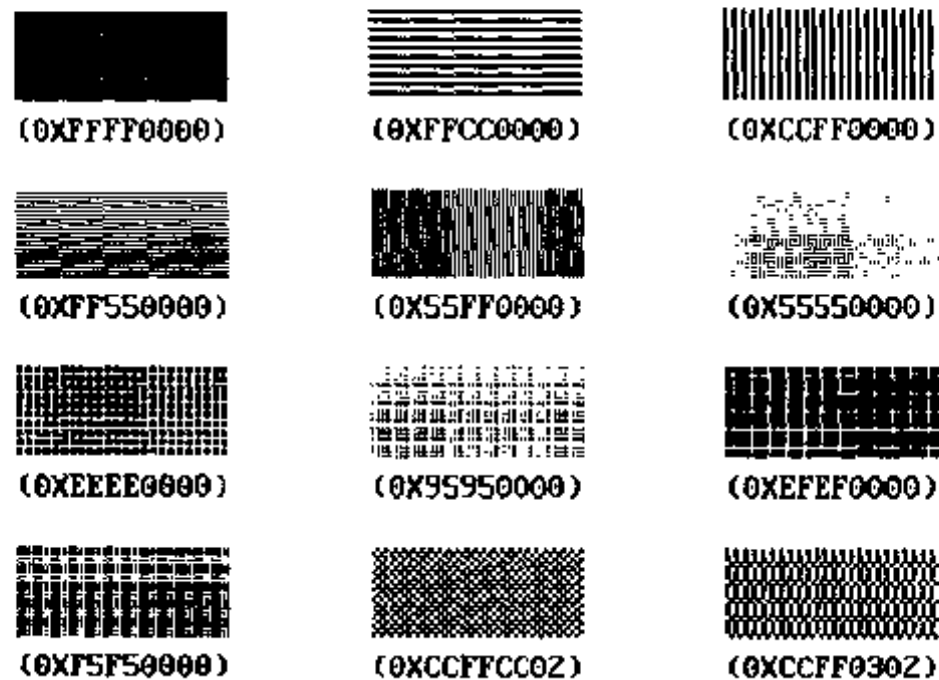


图 10-1 \_Bar 函数的显示图案

```

char byte; /* 要进行循环位的字节 */
int n; /* 循环移位的位数 */
{
    asm mov cx,n;
    asm mov al,byte;
    asm rol al,cl;
    asm mov byte,al;
    return byte;
}

/* --- 函数 _Bar : 在屏幕上指定位置显示一个矩形块 --- */
void _Cdecl _Bar(x,y,w,h,color,pattern)
int x,y; /* 矩形块左上角坐标,均以像素为单位 */
int w,h; /* 矩形块宽和高,均以像素为单位 */
int color; /* 矩形块颜色 */
unsigned long pattern; /* 填充模式 */
{
    register unsigned i;
    unsigned char hp1 = pattern>>24;
    unsigned char hp2 = ror(hp1,(int)(pattern & 0xff));
    unsigned char vp = (pattern >> 16) & 0xff;
    unsigned char cp = (pattern >> 8) & 0xff;

    for(i=0;i<h;i++)
        if(vp & (0x80 >> i % 8))
        {

```

```

        if(cp & (0x80 >> i%8))
            _H_Line(x,y+i,w,color,hp2);
        else
            _H_Line(x,y+i,w,color,hp1);
    }
}

```

除了上面介绍的画线、框函数以外, HANENV 中还有一组使用扩展 ASCII 码中的制表符号画线、框的函数。这些函数经常用于制作文本方式下的屏幕表格。

#### 程序 10-6 用扩充 ASCII 码画横线。

使用扩展 ASCII 码画线函数实质上是写扩展 ASCII 码字符串, 因此确定字符或汉字的放大倍数的全局变量 `_Xtimes` 和 `_Ytimes` 仍然起作用。在画线之前还要先清理画线用字符所占用的屏幕空间的背景。线的颜色及背景色由全局变量 `_TextColor` 和 `_Background` 之值决定。

```

/* -----
   函数 char _h_line : 用扩充 ASCII 码画横线
   ----- */
#include (hanenv.h)

void _Cdecl char _h_line(col,line,len,type)
int col;                /* 横线左端列坐标(以字节为单位) */
int line;               /* 横线左端行坐标(以像素为单位) */
int len;               /* 横线长度(以字节为单位) */
int type;              /* 横线类型 */
{
    int i;
    int c = type == DOUBLE? 205:196;

    for(i=0;i<len;i++)
    {
        _Block(col+i*_Xtimes,line,_Xtimes,CHAR_HIGH*_Ytimes,_Background);
        outxychar(col+i*_Xtimes,line,_TextColor,c);
    }
}

```

#### 程序 10-7 用扩充 ASCII 码画竖线。

在使用扩充 ASCII 码画竖线时值得一提的是扩充 ASCII 码中的竖向制表符占满了一个字符行(18 线), 这样, 在文本方式下使用扩充 ASCII 码画竖线时就不会在行间留下间隙。

```

/* -----
   函数 char_v_line : 用扩充 ASCII 码画竖线
   ----- */
#include <hanenv.h>

void _Cdecl char_v_line(col,line,len,type)
int col;           /* 竖线上端列坐标(以字节为单位) */
int line;          /* 竖线上端行坐标(以像素为单位) */
int len;           /* 竖线长度(以字符为单位) */
int type;          /* 竖线类型 */
{
    int i;
    int c = type==DOUBLE? 186:179;
    int char_high = CHAR_HIGH*_Ytimes;
    for(i=0;i<len;i++)
    {
        _Block(col,line+i*char_high,_Xtimes,char_high,_Background);
        outxychar(col,line+i*char_high,_TextColor,c);
    }
}

```

程序 10-8 用扩充 ASCII 码画矩形。

```

/* -----
   函数 char_rectangle : 用扩充 ASCII 码画矩形
   ----- */
#include <hanenv.h>

void _Cdecl char_rectangle (col,line,width,height,type)
int col;           /* 矩形左上角列坐标(以字节为单位) */
int line;          /* 矩形左上角行坐标(以像素为单位) */
int width;         /* 矩形宽度(以字节为单位) */
int height;        /* 矩形高度(以字符为单位) */
int type;          /* 线类型 */
{
    int char_high = CHAR_HIGH*_Ytimes;

    _Block(col,line,_Xtimes,char_high,_Background);
    outxychar(col,line,_TextColor,type==DOUBLE? 201:218);
    char_h_line(col+_Xtimes,line,width-2,type);
    _Block(col+(width-1)*_Xtimes,line,_Xtimes,char_high,_Background);
    outxychar(col+(width-1)*_Xtimes,line,_TextColor,type==DOUBLE?
        187:191);
}

```

```

char _v_line(col,line+high,high-2,type);
char _v_line(col+(width-1)*_Xtimes,line+char_high,high-2,type);
_Block(col,line+(high-1)*char_high,_Xtimes,char_high,_Background);
outxychar(col,line+(high-1)*char_high,_TextColor,type==DOUBLE?
200;192);
_Block(col+(width-1)*_Xtimes,line+(high-1)*char_high,_Xtimes,char
_high,_Background);
outxychar(col+(width-1)*_Xtimes,line+(high-1)*char_high,
_TextColor,type==DOUBLE? 188;217);
char _h_line(col+_Xtimes,line+(high-1)*char_high,width-2,type);
}

```

在以上三个函数中所能画出线型有单线、双线两种。虽然它们是画线、框的函数，但实际上使用方法和 putxy... 族的汉字或字符串显示函数差不多。

### 10.3 构造对话框和按键

本节介绍几个用于构造立体框和按键的函数的设计。

#### 程序 10-9 画立体框。

该函数用于画出一个向外凸出或向内凹进的窗口。

```

/* -----
   函数 _Hole : 画立体框
   ----- */
#include <hanenv.h>

void _Hole(x,y,width,high,color1,color2,thickness)
int x;           /* 立体框左上角列坐标(以像素为单位) */
int y;           /* 立体框左上角行坐标(以像素为单位) */
int width;       /* 立体框宽度(以像素为单位) */
int high;        /* 立体框高度(以像素为单位) */
int color1;      /* 立体框左上边颜色 */
int color2;      /* 立体框右下边颜色 */
int thickness;   /* 立体框厚度(以像素为单位) */
{
    int i;
    for(i=0;i<thickness;i++)
    {
        _H_Line(x+i,y+i,width-i*2,color1,0xff);
        _V_Line(x+i,y+i,high-i*2,color1,0xff);
    }
}

```

```

    _V_Line(x+width-1-i,y+1+i,high-1-i*2,color2,0xff);
    _H_Line(x+1+i,y+high-1-i,width-1-i*2,color2,0xff);
}
}

```

该函数所绘之框究竟是向外凸出还是向内凹进全靠背景颜色和框颜色的搭配。例如，如下程序段绘出一个凸出的按键：

```

    _Bar(x,y,50,26,LIGHTGRAY,0xffff0000);
    draw_rectangle(x,y,50,26,BLACK,0xff);
    _Hole(x+1,y+1,48,24,WHITE,DARKGRAY,2);

```

而程序段

```

    _Bar(x,y,50,26,LIGHTGRAY,0xffff0000);
    draw_rectangle(x,y,50,26,BLACK,0xff);
    _Hole(x+1,y+1,48,24,WHITE,DARKGRAY,2);

```

画出一个凹进的窗口，如图 10-2 所示。

#### 程序 10-10 画一带有边框的矩形。

带有薄边的对话框已经成为 WINDOWS 式用户界面的特征(如图 5-1 所示)。我们设计的画框函数使用了如下几个全局变量：



图 10-2 框、凸出的按键与凹进的窗口

\_Background : 框内背景色;  
 \_BoxLineColor: 画框所用的线色,初始值为黑色;  
 \_BarColor : 这是一个 unsigned 类型的全局变量,用来表示一个立体块的颜色。第 0—3 位为立体块上的字符或汉字的颜色;第 4—7 位为立体块表面的颜色;第 8—11 位为立体块左、上边的颜色;第 12—15 位为立体块右、下边的颜色。该立体块可能是一按键,也可能是一凹进的窗口。该变量的初值为黑、浅灰、白色和深灰色,我们用其中的立体块表面颜色作为框的颜色。

```

/* -----
   函数 _Box : 画框
   ----- */
#include <hanenv.h>

void _Cdecl _Box(x,y,width,high)

```

```

int x;                /* 框左上角列坐标(以像素为单位) */
int y;                /* 框左上角行坐标(以像素为单位) */
int width;            /* 框宽度(以像素为单位) */
int high;             /* 框高度(以像素为单位) */
{
    int i;
    int color = (_BarColor>>4) & 0x0f;

    _Bar(x,y,width,high,_Background,0xffff0000);
    for(i=0;i<4;i++)
        draw_rectangle(x+i,y+i,width-2*i,high-2*i,i==0 || i==3? _BoxLine
                        Color;color,0xff);
}

```

#### 程序 10-11 画按键。

该函数用于显示一个立体的按键。由于决定一个按键的参数很多,所以我们在头文件 hanenv.h 中定义了一个按键类型,其中包括按键的大小、位置、颜色、键名、键名放大倍数等分量(参看第 14 章:“HANENV 系统的头文件”)。在显示按键时,要注意的是按键类型中的按键位置是相对于菜单的位置,按键有弹起和按下两种状态以及如果按键处于被锁住状态时显示键名要用虚体字等。

```

/* -----
   函数 _DrawButton: 在指定坐标画按键
   ----- */
#include (hanenv.h)

void _Cdecl _DrawButton(button,col,line,chameleon,op)
BUTTON_TYPE button;    /* 按键参数 */
int col;                /* 菜单左上角列坐标(以字节为单位) */
int line;               /* 菜单左上角行坐标(以像素为单位) */
int chameleon;          /* 按下按键时是否改变颜色 */
int op;                 /* 按键状态(0=弹起,1=按下) */
{
    int color;           /* 按键名颜色 */
    int bk;              /* 按键面颜色 */
    int lightedge;       /* 按键阳面边缘颜色 */
    int darkedge;        /* 按键阴面边缘颜色 */
    int x;               /* 按键绝对列坐标 */
    int y;               /* 按键绝对行坐标 */
    int xtimes = _Xtimes; /* 保存原来的横向放大倍数 */
    int ytimes = _Ytimes; /* 保存原来的纵向放大倍数 */
    int namex;           /* 键名显示列坐标 */

```

```

int namey;                                /* 键名显示行坐标          */

/* —— 取出显示按键的各种颜色 —— */
if(chameleon)
{
    color = (button.pushcolor) & 0x0f;
    bk = (button.pushcolor>>4) & 0x0f;
    lightedge = (button.pushcolor>>8) & 0x0f;
    darkedge = (button.pushcolor>>12) & 0x0f;
}
else
{
    color = button.butncolor & 0x0f;
    bk = (button.butncolor>>4) & 0x0f;
    lightedge = (button.butncolor>>8) & 0x0f;
    darkedge = (button.butncolor>>12) & 0x0f;
}

/* —— 取键名显示放大倍数 —— */
_Xtimes = button.fonttimes&0xff;
_Ytimes = button.fonttimes>>8;

/* —— 计算显示按键的各种坐标 —— */
x = col * 8 + button.x;
y = line + button.y;
namex = x + button.w/2 - strlen(button.name) * 4 * _Xtimes;
namey = y + (button.h - _CurrentHZK ->fonthigh * _Ytimes)/2 - 1;

/* —— 显示按键 —— */
_Bar(x,y,button.w,button.h,bk,0xffff0000);
if(button.high)
    draw_rectangle(x,y,button.w,button.h,_BoxLineColor,0xff); if(op==0)
    _Hole(x+1,y+1,button.w-2,button.h-2,lightedge,darkedge,button.high);
if(button.lock)
    set_font_bk_style(DOTTED_LINE);
drawxystr(namex,namey+((op&&button.high)?1:0),color,button.name);
set_font_bk_style(SOLIDLINE);
_Xtimes = xtimes;
_Ytimes = ytimes;
}

```

该函数除了应用于按键式菜单以外,也可以直接用于其他只需一两个键的应用场合,如提问函数 ask 中。



## 10.4 直线和曲线

为了满足某些特殊需要，例如在设置了尺寸比实际屏幕要大的虚屏上作图，我们为 HANENV 系统编写了两个功能较强的绘图函数：一个是绘制任意直线的函数，一个是功能弧线绘制函数。特别是后一个函数，功能十分强大，可以绘制圆或椭圆弧线和扇形等多种图形，并因此派生了若干比较方便使用的函数(宏)。图 10-3 中给出了一个利用 HANENV 系统制作的软件封面。

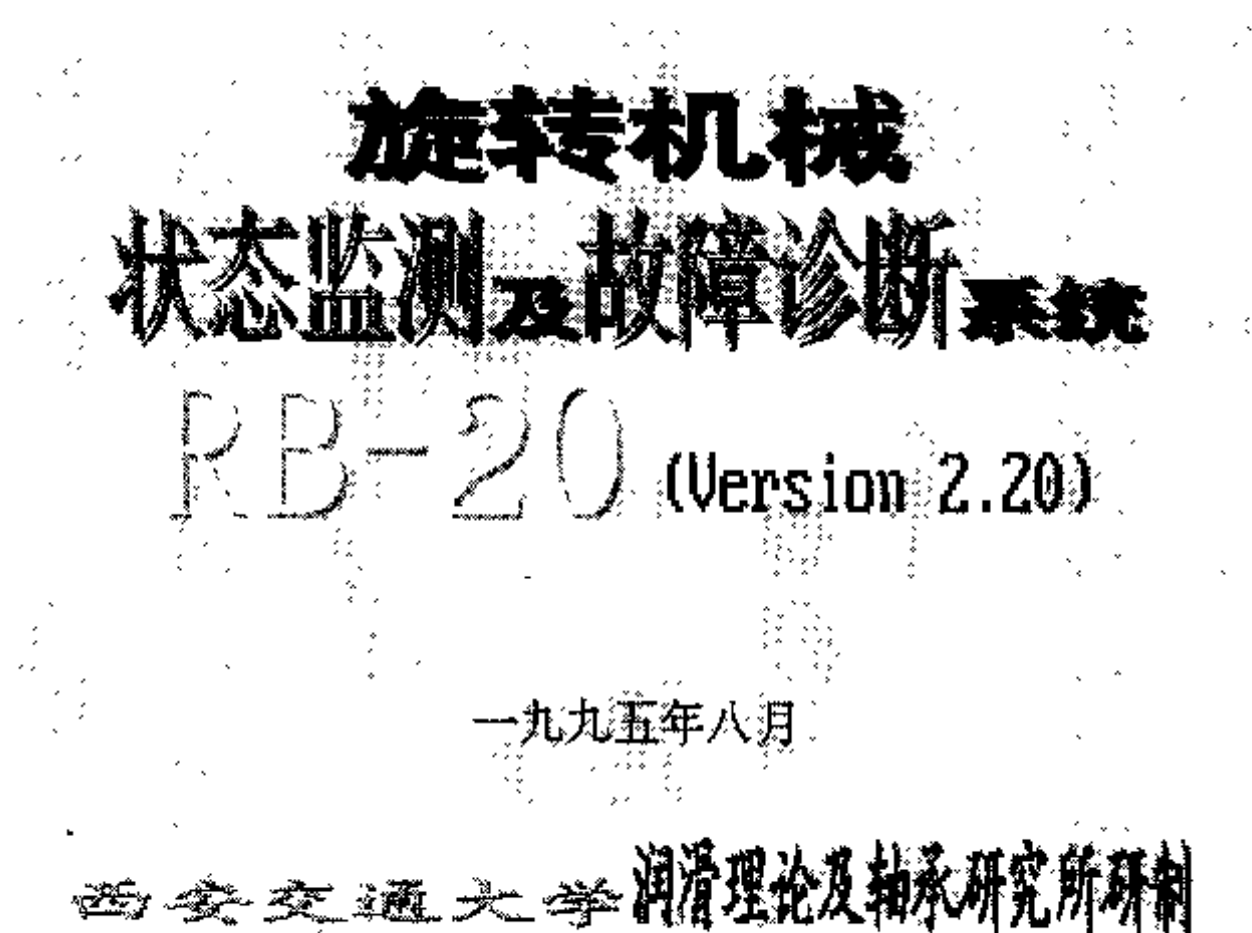


图 10-3 软件封面举例

程序 10-12 画任意直线。

我们采用 Bresenham 的直线插补算法求出直线段上每点的坐标。迄今为止，这是最好的画直线算法，无需用到浮点运算，甚至也无需用到乘法和除法，所以速度很快。

```
/* -----
   函数 draw_line : 画直线
   ----- */
#include <hanenv.h>

void _Cdecl draw_line(x1,y1,x2,y2,color)

int x1;          /* 直线始端列坐标(以象素为单位) */
```

```

int y1;          /* 直线始端行坐标(以像素为单位) */
int x2;          /* 直线末端列坐标(以像素为单位) */
int y2;          /* 直线末端行坐标(以像素为单位) */
int color;       /* 直线颜色 */
{
    register int t,d;
    int xer = 0;
    int yer = 0;
    int deta_x;
    int deta_y;
    int incx;
    int incy;

    /* --- 计算线在 X 轴和 Y 轴上投影的长度 --- */
    deta_x = x2-x1;
    deta_y = y2-y1;

    /* --- 确定第二点的增量方向 --- */
    if(deta_x>0)
        incx = 1;
    else if(deta_x==0)
        incx = 0;
    else
        incx = -1;
    if(deta_y>0)
        incy = 1;
    else if(deta_y==0)
        incy = 0;
    else
        incy = -1;

    /* --- 确定线在 X 轴和 Y 轴上投影的长度 --- */
    deta_x = abs(deta_x);
    deta_y = abs(deta_y);

    /* --- 确定线中点数 --- */
    if(deta_x>deta_y)
        d = deta_x;
    else
        d = deta_y;

    /* --- 循环绘出线中所有点 --- */
    for(t=0;t<=d;t++)

```

```

{
    _PutPixel(x1,y1,color);
    xer += deta_x;
    yer += deta_y;
    if(xer>d)
    {
        xer -= d;
        x1 += incx;
    }
    if(yer>d)
    {
        yer -= d;
        y1 += incy;
    }
}
}
}

```

#### 程序 10-13 画弧。

我们采用 Bresenham 的画圆算法实现我们的画弧函数。和画直线的 Bresenham 插补算法一样,该算法在画弧的过程中也无需用到浮点运算、乘法和除法(但在计算弧的起点坐标时我们使用了浮点运算和乘、除法)。我们设计的这个画弧函数的功能很强,不仅能够画出圆、椭圆以及圆弧和椭圆弧,还可以画出空心或实心的扇形,非常适合绘制用于数据统计的饼图和立体饼图。

```

/* -----
   函数 _DrawArc : 画弧
   ----- */
#include<hanenv.h>
#include<math.h>
#include<stdlib.h>

void _Cdecl _DrawArc(col,row,a,b,t1,t2,color,fill)
int col;          /* 圆心列坐标(以像素为单位) */
int row;          /* 圆心行坐标(以像素为单位) */
int a;            /* 椭圆横半径(以像素为单位) */
int b;            /* 椭圆纵半径(以像素为单位) */
int t1;           /* 弧起始角度(以度为单位) */
int t2;           /* 弧终止角度(以度为单位) */
int color;        /* 弧颜色 */
int fill;         /* 填充模式 */
{
    int dx,dy,f,fx,fy,x,y,j,mx,my,x1,y1,dx,ly,r,jf;

```

```

double t;

/* —— 计算弧起始点的坐标 —— */
t = (double)t1 * M_PI/180.0;
r = max(a,b);
x1 = (double)r * cos(t)+0.5;
y1 = (double)r * sin(t)+0.5;
t = (double)(t2-t1);
while(fabs(t)>360.0)
    t += t>0? -360.0:360.0;
if(x1==0 && y1==0)
    return;
if(fabs(t)+1.0==1.0)
    t = 360.0;
if(t>0.0)
{
    dx = (y1>0 || y1==0 && x1>0)? -1:1;
    dy = (x1<0 || x1==0 && y1>0)? -1:1;
}
else
{
    dx = (y1<0 || y1==0 && x1>0)? -1:1;
    dy = (x1>0 || x1==0 && y1>0)? -1:1;
}
lx = 0;
ly = 0;
j1 = 0;
t2 = t1;
while(t2>=90)
{
    j1 += 2 * r;
    t2 -= 90;
}
x = r * cos((double)t2 * M_PI/180.0);
y = r * sin((double)t2 * M_PI/180.0);
j1 += y + (r-x);
j = 0;
t2 = t;
t += t1;

if(t2>0)
{
    while(t>=90.0)

```

```

    {
        j += 2 * r;
        t -= 90.0;
    }
    x = r * cos(t * M_PI/180.0) + 0.5;
    y = r * sin(t * M_PI/180.0) + 0.5;
    j += y + (r - x);
    j -= j1;
}
else
{
    if(t > 0.0)
        t2 = 1;
    else
    {
        t2 = 0;
        t = -t;
    }
    while(t >= 90.0)
    {
        j += 2 * r;
        t -= 90.0;
    }
    x = r * cos(t * M_PI/180.0) + 0.5;
    y = r * sin(t * M_PI/180.0) + 0.5;
    j += y + (r - x);
    if(t2 == 1)
        j = j1 - j;
    else
        j = j + j1;
}
j -= 1;
f = 0;
fx = 2 * x1 * dx + 1;
fy = 2 * y1 * dy + 1;
mx = 0;
my = 0;
t = (double)t1 * M_PI/180.0;
x = (int)((double)a * cos(t) + 0.5);
y = (int)((double)b * sin(t) + 0.5);

/* -- 如果是实心扇形则画一半径 ----- */
if(fill)

```

```

draw_line(col+x,row-y,col,row,color);

/* -- 如果是弧则画起始点 ----- */
else
    _PutPixel(col+x,row-y,color);

/* -- 画弧 ----- */
while(j! =0)
{
    if(f>=0)
    {
        lx += a;
        if(lx>=r)
        {
            /* -- 如果是实心扇形则填充 -- */
            if(fill==2)
            {
                x += dx;
                lx -= r;
                draw_line(col+x,row-y,col,row,color);
            }

            /* -- 画弧上的点 ----- */
            else
            {
                if(mx! =0)
                {
                    _PutPixel(col+x,row-y,color);
                    mx = 0;
                }
                x += dx;
                lx -= r;
                if(my! =0)
                {
                    _PutPixel(col+x,row-y,color);
                    my = 0;
                }
                else
                    mx = 1;
            }
        }
        f -= abs(fx);
        fx += 2;
    }
}

```

```

if(fx==0 || fx<0 && fx-2>0 || fx>0 && fx-2<0)
{
    dy = -dy;
    fy = -fy+2;
    f = -f;
}
}
else
{
    ly += b;
    if(ly>=r)
    {
        if(fill==2)
        {
            y += dy;
            ly -= r;
            draw_line(col+x,row-y,col,row,color);
        }
        else
        {
            if(my!=0)
            {
                _PutPixel(col+x,row-y,color);
                my = 0;
            }
            y += dy;
            ly -= r;
            if(mx!=0)
            {
                _PutPixel(col+x,row-y,color);
                mx = 0;
            }
            else
            {
                my = 1;
            }
        }
    }
    f += abs(fy);
    fy += 2;
    if(fy==0 || fy<0 && fy-2>0 || fy>0 && fy-2<0)
    {
        dx = -dx;
        fx = -fx+2;
        f = -f;
    }
}

```

```
    }  
    }  
    j--;  
}  
if(fill)  
    draw_line(col,row,col+x,row-y,color);  
if(mx || my)  
    _PutPixel(col+x,row-y,color);  
}
```

由于上面的画弧函数功能很强,参数太多,所以不易使用。因此我们根据此函数又设置了一组比较简单的绘图函数:

```
void draw_circle(int x,int y,int r,int color);  
void draw_arc(int x,int y,int r,int t1,int t2,int color);  
void draw_pie(int x,int y,int r,int color);  
void draw_fan(int x,int y,int r,int t1,int t2,int color);  
void draw_ellipse(int x,int y,int a,int b,int color);
```

这5个绘图函数分别用于绘制圆、圆弧、实心圆、扇形和椭圆。因此,只要不是画椭圆扇形等比较复杂的图形,就可以使用上面这组比较简单的绘图函数来代替函数 DrawArc。实际上,这组函数都是定义在头文件 hanenv.h 中的带参宏,其定义可以参看第14章的有关部分。



# 图形用户界面设计

在设计图形用户界面时,除了鼠标的使用、绘图和各种规格尺寸的汉字和字符的显示以外,还要考虑几种常用图形部件:提示和对话框、滚动条、代码表、数据录入窗口以及按键式和图标式菜单等的应用。在本章中我们先介绍前面几种部件的设计方法,而把数据录入窗口和菜单的设计分别留到第 12 章和第 13 章解决。

### 11.1 提示和对话框

我们首先介绍一个弹出式提示框的设计。对该框的性能要求是在调用函数时首先在屏幕上的指定位置显示一个提示框,框中有预先设计的说明信息。待用户按下键盘上的任一键或鼠标按钮时即退出提示,自动恢复原来的屏幕背景。要求该提示框的显示不和鼠标光标和文本光标发生任何冲突,同时该提示框的位置、大小、颜色等参数均可随意设置。图 11-1 给出了一个提示框的实际例子。

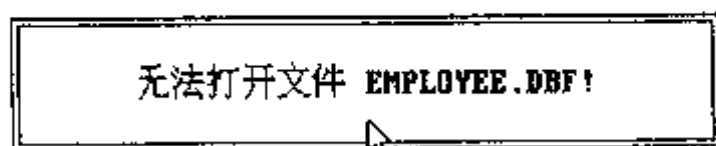


图 11-1 提示框

#### 程序 11-1 显示提示信息。

在函数中如果需要设置的参数太多会给程序员带来不便,所以我们将提示框中所用到的所有颜色参数以及提示信息的放大倍数等都改由全局变量控制。这些全局变量是:

- \_TextColor : 提示信息中的汉字或字符的显示颜色;
- \_Background : 提示框内的背景色;
- \_BarColor : 其中的背景色决定提示框框边的颜色(浅灰);
- \_BoxLineColor: 画提示框轮廓线的颜色(黑色);
- \_Xtimes : 提示信息中汉字或字符的横向放大倍数(1);
- \_Ytimes : 提示信息中汉字或字符的纵向放大倍数(1);

除非要得到某种特殊显示效果,否则其中后 4 个变量的值通常保持不变。实际上 HANENV 系统的多数应用部件都是这样设计的。

显示提示信息函数的工作流程很简单:

1. 首先关闭鼠标,保存提示框下面的屏幕背景;
2. 然后显示提示框;
3. 显示提示信息,显示提示信息有两种模式:如果提示信息长度不足一行则将提示信息显示在行的正中间,如果提示信息的长度大于一行的长度则按文本窗口显示方式显示提示;
4. 打开鼠标,等待用户操作;
5. 一旦用户按下键盘上的任何键或鼠标按钮,则关闭鼠标,恢复原来的背景图象。

```

/* -----
   函数 dispinfo : 显示提示信息
   ----- */

#include <hanenv.h>
#include <string.h>

void _Cdecl dispinfo(col,line,width,height,prompt)
int col;                /* 提示框左上角列坐标(以字节为单位) */
int line;               /* 提示框左上角行坐标(以像素为单位) */
int width;              /* 提示框宽度(以字节为单位) */
int height;             /* 提示框高度(以像素为单位) */
char * prompt;          /* 提示信息 */
{
    char ** block;       /* 存储提示框背景的缓冲区指针 */

    int pmtwidth = _CurrentHZK->fontwidth * _Xtimes * strlen(prompt)/2;
    int pmthigh  = (_CurrentHZK->fonthigh+2) * _Ytimes;
    int pmtcol   = col+(width-pmtwidth)/2;
    int pmthline = line+(height-pmthigh)/2;
    delight _mouse();
    block = getblock(col,line,width,height);
    _Box(col*8,line,width*8,height);
    if(pmtwidth<=width-2)
        outxystr(pmtcol,pmthline,_TextColor,prompt);
    else
    {
        int old_wl,old_wt,old_wr,old_wb;
        int old_col = _TextCol;
        int old_row = _TextLine;
        get_window(old_wl,old_wt,old_wr,old_wb);
        pmthigh = (pmtwidth/(width-2)+2) * _CurrentHZK->fonthigh * _Ytimes;
    }
}

```

```

    pmtwidth  = width-2;
    pmtcol    = col+1;
    pmiline   = line+8
    set _window(pmtcol,pmtline,pmtcol+pmtwidth-1,pmtline+pmthigh-1);
    movecursor(pmtcol,pmtline);
    outstr(prompt);
    set _window(old _wl,old _wt,old _wr,old _wb);
    movecursor(old _col,old _row);
}
light _mouse();
till _mouse _pop(LEFT _BUTTON);
geth();
delight _mouse();
putblock(col,line,width,high,block);
light _mouse();
}

```

在设计应用程序时经常需要一种提问模块,用于向操作人员提出一个问题,并得到一个肯定或否定的回答。这就是逻辑提问框。逻辑提问框的设计比上面的提示框稍微复杂些。我们设计的逻辑提问框中有两个立体按键,上面标有“是”和“否”,操作人员可以通过使用鼠标点选其中的某一个按键,或者直接使用键盘上的字母键“Y”和“N”作出回答。另外,在逻辑提问框中除了问题以外,应用程序的程序设计人员还可以另外选用一个标题项。该标题项中字符的显示颜色和背景色由全局变量 \_TitleColor(白色)和 \_TitleBk(蓝色) 的值决定。一个实际的逻辑提问框如图 11-2 所示。

#### 程序 11-2 逻辑提问框。

逻辑提问框的简单流程为

1. 关闭鼠标,保存提问框下面的屏幕背景;

2. 显示提问框;

3. 如果有标题的话,显示标题;

4. 如果有问题的话,显示问题;

5. 显示按键;

6. 打开鼠标,等待用户的操作;

7. 如果用户作出了回答,则关闭鼠标,恢复原来的背景;

8. 返回用户的回答。

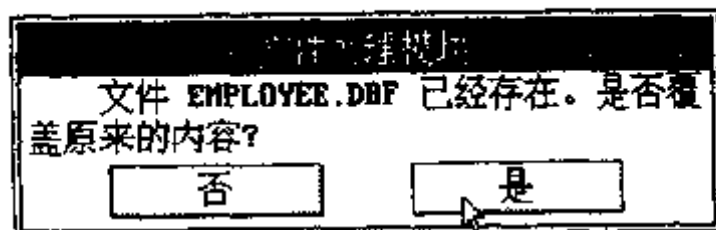


图 11-2 逻辑提问框

```

/* -----
   函数 ask : 逻辑提示框
   ----- */

```

```

#include <hanenv.h>
#include <string.h>

/* ----- 宏定义：两个按钮的鼠标范围 ----- */
#define YES_l (col * 8 + buttons[YES].x)
#define YES_t (pmtline + buttons[YES].y)
#define YES_r (col * 8 + buttons[YES].x + buttons[YES].w)
#define YES_b (pmtline + buttons[YES].y + buttons[YES].h)
#define NO_l (col * 8 + buttons[NO].x)
#define NO_t (pmtline + buttons[NO].y)
#define NO_r (col * 8 + buttons[NO].x + buttons[NO].w)
#define NO_b (pmtline + buttons[NO].y + buttons[NO].h)

int _Cdecl ask(col, line, width, high, title, prompt)
int col; /* 提示框左上角列坐标(以字节为单位) */
int line; /* 提示框左上角行坐标(以像素为单位) */
int width; /* 提示框宽度(以字节为单位) */
int high; /* 提示框高度(以像素为单位) */
char * title; /* 提示框名称 */
char * prompt; /* 提示信息 */
{
    char ** block; /* 存储提示框背景的缓冲区指针 */
    unsigned key;
    int pmtwidth;
    int pmthigh;
    int pmtcol;
    int pmtline = line + 8;
    int old_l = _TextWinLeft;
    int old_t = _TextWinTop;
    int old_r = _TextWinRight;
    int old_b = _TextWinBottom;
    BUTTON_TYPE buttons[2];

    delight_mouse();

    /* ----- 存储提示框的背景 ----- */
    block = getblock(col, line, width, high);
    /* ----- 画提示框 ----- */
    _Box(col * 8, line, width * 8, high);

    /* ----- 如果有标题,则显示标题并调整有关参数 ----- */
    if(title)
    {

```

```

    pmtcol      = col * 8 + 4;
    pmtwidth    = (width - 1) * 8;
    pmthigh     = (_CurrentHZK -> fonthigh + 2) * _Ytimes;

    _Bar(pmtcol, line + 4, pmtwidth, pmthigh + 4, _TitleBk, 0xffff0000);
    if(prompt)
        _H_Line(pmtcol, pmtline + pmthigh, pmtwidth, _BoxLineColor, 0xff);
    pmtwidth    = strlen(title) * _CurrentHZK -> fontwidth * _Xtimes / 2;
    pmtcol      = col + (width - pmtwidth) / 2;
    outxstr     (pmtcol, pmtline, _TitleColor, title);
    pmtline +   = (pmthigh + 1);
}

/* ----- 如果有提示,则显示提示并调整有关参数 ----- */
if(prompt)
{
    pmtcol      = col + 1;
    pmtwidth    = strlen(prompt) * _CurrentHZK -> fontwidth * _Xtimes / 2;
    pmthigh     = (pmtwidth / (width - 1) + 1) * (_CurrentHZK -> fonthigh + 2)
                * _Ytimes;
    pmtwidth    = width - 2;
    set_window(pmtcol, pmtline, pmtcol + pmtwidth - 1, pmtline + pmthigh - 1);
    _TextCol    = pmtcol;
    _TextLine   = pmtline;
    outstr(prompt);
    set_window(old_l, old_t, old_r, old_b);
    pmtline += pmthigh;
}

/* ----- 设置按钮的有关参数 ----- */
pmtwidth      = _CurrentHZK -> fontwidth * _Xtimes;
pmtwidth      += 2 * (width - pmtwidth - 3) / 7;
pmtwidth      *= 8;
buttons[NO].x  = (width * 8 - pmtwidth * 2 - 8) / 3 + 4;
buttons[NO].y  = 4 * _Ytimes;
buttons[NO].w  = pmtwidth;
buttons[NO].h  = (_CurrentHZK -> fonthigh + 8) * _Ytimes;
buttons[NO].butncolor = _BarColor;
buttons[NO].pushcolor = _BarColor;
buttons[NO].fonttimes = _Ytimes << 8 | _Xtimes;
buttons[NO].name  = "否";
buttons[NO].key   = 'N';
buttons[NO].pressed = NO;

```

```

buttons[NO].high      = 1;
buttons[NO].lock      = NO;
buttons[NO].fun       = NULL;

buttons[YES].x        = (width * 8 - pmtwidth * 2 - 8) / 3 * 2 + 4 + pmtwidth;
buttons[YES].y        = 4 * _Ytimes;
buttons[YES].w        = pmtwidth;
buttons[YES].h        = (_CurrentHZK - > fonthigh + 8) * _Ytimes;
buttons[YES].butncolor = _BarColor;
buttons[YES].pushcolor = _BarColor;
buttons[YES].fonttimes = _Ytimes << 8 | _Xtimes;
buttons[YES].name     = "是";
buttons[YES].key      = 'Y';
buttons[YES].presscd  = YES;
buttons[YES].high     = 1;
buttons[YES].lock     = NO;
buttons[YES].fun      = NULL;

/* ----- 显示按钮 ----- */
_DrawButton(buttons[YES], col, pmtline, NO, 0);
_DrawButton(buttons[NO], col, pmtline, NO, 0);

/* ----- 主循环: 根据键盘和鼠标信息进行逻辑选择 ----- */
light_mouse();
do
{
    key = geth();
    switch(key)
    {
        case LEFT_BUTTON:
            if(mouse_enter(YES_l, YES_t, YES_r, YES_b))
            {
                delight_mouse();
                _DrawButton(buttons[YES], col, pmtline, NO, 1);
                light_mouse();
                till_mouse_pop(LEFT_BUTTON);
                key = 'Y';
            }
            else if(mouse_enter(NO_l, NO_t, NO_r, NO_b))
            {
                delight_mouse();
                _DrawButton(buttons[NO], col, pmtline, NO, 1);
                light_mouse();
            }
        }
    }

```

```

        till_mouse_pop(LEFT_BUTTON);
        key = 'N'
    }
    break;
case 'y':
case 'Y':
    delight_mouse();
    _DrawButton(buttons[YES],col,pmtline,NO,1);
    light_mouse();
    delay(200);
    break;
case 'n':
case 'N':
    delight_mouse();
    _DrawButton(buttons[NO],col,pmtline,NO,1);
    light_mouse();
    delay(200);
    break;
case RIGHT_BUTTON:
    key = KEY_ESC;
    break;
}
}while(! strchr"YyNn",key) && key!=KEY_ESC);
/* ----- 恢复提示框处原来的背景 ----- */
delight_mouse();
putblock(col,line,width,high,block);
light_mouse();

/* ----- 将键码转换为逻辑值并返回 ----- */
if(key=='Y' || key=='y')
    key = YES;
else if(key=='N' || key=='n')
    key = NO;
else
    /* key==KEY_ESC || key==RIGHT_BUTTON */
    key = -1;
return key;
}

```

## 11.2 滚动条

滚动条是 WINDOWS 窗口式用户界面的重要特征。通过滚动条,应用程序的操作人员可以利用鼠标方便地操作表格数据的选择、光条的移动和窗口的翻页。在 HANENV 系统中,我们仿照 WINDOWS 中的滚动条,为应用程序的设计人员提供了两个使用方便的滚动条函数。

### 程序 11-3 滚动条模块。

我们提供的滚动条函数共有两个,一个是横向滚动条函数 `scroll_h_box`,一个是纵向滚动条函数 `scroll_v_box`,其实例可参看图 5-1,在编辑器窗口的右面和下面分别有一个纵向和横向滚动条,用来控制当前编辑位置。这两个函数共享一组字模数据和内部显示函数,功能基本相同,均为控制某个量(参数 `n`)在指定范围(0—参数 `range`)内变化。滚动条使用鼠标操作,其操作方法有以下三种:

1. 用鼠标左键点选滚动条两端的按键,可以使量 `n` 的值增大或减小 1,同时修改滑标的位置;
2. 用鼠标左键点击滚动条条身(不包括滑标),可以使量 `n` 的值加或减一个固定的数值(页面长度),同时修改滑标的位置;
3. 使鼠标光标指向滑标,然后按下鼠标左键不松手,可以拖动滑标在滚动条上滑动,待松手后即可按比例调整量 `n` 的值。

```
/* -----
   函数模块 scroll_box : 滚动条
   ----- */
#include <hanenv.h>
#include <ctype.h>

/* ----- 滚动条中箭头的字模 ----- */
static unsigned char _Fonts[] =
{
    /* -- 向上箭头的字模 ----- */
    0x18,0x3C,0x7E,0xFFFF,0xFFFF,0x3C,0x3C,0x3C,

    /* -- 向下箭头的字模 ----- */
    0x3C,0x3C,0x3C,0xFFFF,0xFFFF,0x7E,0x3C,0x18,

    /* -- 向左箭头的字模 ----- */
    0x18,0x1C,0xFFFE,0xFFFF,0xFFFF,0xFFFE,0x1C,0x18,

    /* -- 向右箭头的字模 ----- */
}
```



```

0x18,0x38,0x7F,0xFFFF,0xFFFF,0x7F,0x38,0x18
};

/* --- 内部函数 _DrawBar : 绘制滚动条中的头、尾滑标 --- */
static void _Cdecl _DrawBar(x,y,type,color,bk,lightedge,darkedge)
int x;          /* 块标左上角列坐标(以像素为单位) */
int y;          /* 块标左上角行坐标(以像素为单位) */
int type;       /* 块标类型(0-4,滑标和上下左右标) */
int color;      /* 块标上箭头的颜色 */
int bk;         /* 块标颜色 */
int lightedge;  /* 块标亮边颜色 */
int darkedge;  /* 块标暗边颜色 */
{
    int pressed = NO;

    /* -- 按下时的滑标类型为 11-14 ----- */
    if(type>4)
    {
        pressed = YES;
        type -= 10;
    }

    /* -- 显示滑标 ----- */
    delight_mouse();
    draw_rectangle(x,y,16,16,_BoxLineColor,0xff);
    _Bar(x+1,y+1,14,14,bk,0xffff0000);
    if(! pressed)
        _Hole(x+1,y+1,14,14,lightedge,darkedge,1);
    if(type)
        _DrawF(x+4+pressed,y+4+pressed,1,8,color,_Fonts+8*(type-1));
    light_mouse();
}

/* --- 内部函数 _MoveBarH : 在横滚动条中移动滑标 --- */
static void _Cdecl _MoveBarH(x1,x2,y,color,bk,lightedge,darkedge)
int x1;          /* 滑标原来的列坐标 */
int x2;          /* 滑标新的列坐标 */
int y;           /* 滑标的行坐标 */
int color;       /* 滑标上箭头的颜色 */
int bk;          /* 滑标颜色 */
int lightedge;   /* 滑标亮边颜色 */
int darkedge;    /* 滑标暗边颜色 */
{
    delight_mouse();

```

```

/* —— 消除原来的滑标 —— * /
_Bar(x1+16,y+1,16,14,bk,0xffff0000);
/* —— 在指定位置画出新的滑标 —— * /
_DrawBar(x2+16,y,0,color,bk,lightedge,darkedge);
light_mouse();
}

/* —— 内部函数 _MoveBarV : 在竖滚动条中移动滑标 —— * /
static void _Cdecl _MoveBarV(x,y1,y2,color,bk,lightedge,darkedge)
int x;          /* 滑标的列坐标 */
int y1;         /* 滑标原来的行坐标 */
int y2;         /* 滑标新的行坐标 */
int color;      /* 滑标上箭头的颜色 */
int bk;         /* 滑标颜色 */
int lightedge;  /* 滑标亮边颜色 */
int darkedge;   /* 滑标暗边颜色 */
{
    delight_mouse();
    /* —— 消除原来的滑标 —— * /
    _Bar(x+1,y1+16,14,16,bk,0xffff0000);
    /* —— 在指定位置画出新的滑标 —— * /
    _DrawBar(x,y2+16,0,color,bk,lightedge,darkedge);
    light_mouse();
}

/* —— 函数 scroll_v_box : 竖向滚动条 —— * /
void _Cdecl scroll_v_box(x,y,len,pagesize,key,range,n,oldpos)
int x;          /* 滚动条左上角列坐标(以像素为单位) */
int y;          /* 滚动条左上角行坐标(以像素为单位) */
int len;        /* 滚动条长度 */
int pagesize;   /* 页面大小 */
unsigned key;   /* 滚动条驱动键码 */
int range;      /* 范围 */
int *n;         /* 位置 */
int *oldpos;    /* 滑标原来的位置 */
{
    int color    = _BarColor & 0x0f;

    int bk       = (_BarColor >> 4) & 0x0f;
    int lightedge = (_BarColor >> 8) & 0x0f;
    int darkedge  = (_BarColor >> 12) & 0x0f;

```

```

int p = range? (long)( * n ) * (len-48)/range:0;

/* --- 如果尚未画出滚动条,画滚动条后退出 --- */
if( * oldpos == -1)
{
    _Bar(x,y,16,len,bk,0xffff0000);
    draw_rectangle(x,y,16,len,_BoxLineColor,0xff);
    _DrawBar(x,y,1,color,bk,lightedge,darkedge);
    _DrawBar(x,y+len-16,2,color,bk,lightedge,darkedge);
    _DrawBar(x,y+16+p,0,color,bk,lightedge,darkedge);

    /* --- 存储当前滑标位置 --- */
    * oldpos = p;
    return ;
}

/* --- 如果无法移动滚动条则退出 --- */
if(range == 0)
    return ;

/* --- 如果使用键盘控制滚动条 --- */
if(key==KEY_Up+1536)
    ( * n ) --;
else if(key==KEY_Down+1536)
    ( * n ) ++;
else if(key==KEY_PgUp+1536)
    ( * n ) -= pagesize;
else if(key==KEY_PgDn+1536)
    ( * n ) += pagesize;
else if(key==KEY_Home+1536)
    ( * n ) = 0;
else if(key==KEY_End+1536)
    ( * n ) = range;

/* --- 如果位置参数小于 0,按 0 对待 --- */
if(( * n ) <= 0)
    ( * n ) = p = 0;

/* --- 如果位置参数大于范围,按范围值对待 --- */
else if(( * n ) >= range)
{
    ( * n ) = range;
    p = len-48;
}

```

```

}

/* --- 按位置参数计算出滑标的位置 --- */
else
    p = (long)(*n) * (len-48)/range;

/* --- 移动滑标 --- */
_MoveBarV(x,y+*oldpos,y+p,color,bk,lightedge,darkedge);

/* --- 存储当前滑标位置 --- */
*oldpos = p;

/* --- 如果驱动键不是鼠标左键,则退出 --- */
if(key != LEFT_BUTTON)
    return ;

/* --- 如果鼠标指向滑标,则可拖动滑标 --- */
if(mouse_enter(x,y+16+*oldpos,x+15,y+16+*oldpos+15))
{
    int lastmy = mouserow();
    int old_ml = _MouseLeft;
    int old_mt = _MouseTop;
    int old_mr = _MouseRight;
    int old_mb = _MouseBottom;
    int mx,my;

    set_mouse_range(x+1,lastmy-*oldpos,x+15,lastmy-*oldpos+len-48);
    while(get_mouse_status(&mx,&my) == LEFT_BUTTON)
        if(my != lastmy && y+lastmy-my>0)
        {
            p = *oldpos+my-lastmy;
            _MoveBarV(x,y+*oldpos,y+p,color,bk,lightedge,darkedge);
            lastmy = my;
            *oldpos = p;
        }
    set_mouse_range(old_ml,old_mt,old_mr,old_mb);
    (*n) = (long)p * range/(len-48);
    p = (long)(*n) * (len-48)/range;
    _MoveBarV(x,y+*oldpos,y+p,color,bk,lightedge,darkedge);

    /* --- 存储当前滑标位置 --- */
    *oldpos = p;
    return ;
}

```

```

}

/* -- 如果鼠标指向滚动条条身,翻页 ----- */
if(mouse_enter(x,y+16,x+15,y+len-16))
{
    p = mouserow()-y-22;
    if(p<*oldpos)
        (*n) -= pagesize;
    else
        (*n) += pagesize;
    if((*n)<0)
        (*n) = 0;
    else if(*n>=range)
        (*n) = range;
    p = (long)(*n)*(len-48)/range;
    _MoveBarV(x,y+*oldpos,y+p,color,bk,lightedge,darkedge);
    delay(_MouseSpeed);
}

/* -- 如果鼠标指向滚动条上块标,滑标向上移动一个单位 -- */
else if(mouse_enter(x,y,x+15,y+15))
{
    _DrawBar(x,y,11,color,bk,lightedge,darkedge);
    if((*n)>0)
        (*n)--;
    p = (long)(*n)*(len-48)/range;
    _MoveBarV(x,y+*oldpos,y+p,color,bk,lightedge,darkedge);
    delay(_MouseSpeed);
    _DrawBar(x,y,1,color,bk,lightedge,carkedge);
}

/* -- 如果鼠标指向滚动条下块标,滑标向下移动一个单位 -- */
else if(mouse_enter(x,y+len-15,x+15,y+len))
{
    _DrawBar(x,y+len-16,12,color,bk,lightedge,darkedge);
    if((*n)<range)
        (*n)++;
    if((*n)==range)
        p = len-48;
    else
        p = (long)(*n)*(len-48)/range;
    _MoveBarV(x,y+*oldpos,y+p,color,bk,lightedge,darkedge);
    delay(_MouseSpeed);
}

```

```

    _DrawBar(x,y+len-16,2,color,bk,lightedge,darkedge);
}
/* -- 存储当前滑标位置 ----- */
*oldpos = p;
}

/* -----
   函数 scroll_h_box : 横向滚动条
   ----- */
void _Cdecl scroll_h_box(x,y,len,pagesize,key,range,n,oldpos)
int x;           /* 滚动条左上角列坐标(以像素为单位) */
int y;           /* 滚动条左上角行坐标(以像素为单位) */
int len;         /* 滚动条长度 */
int pagesize;    /* 页面大小 */
unsigned key;    /* 滚动条驱动键码 */
int range;       /* 范围 */
int *n;          /* 位置 */
int *oldpos;     /* 滑标原来的位置 */
{
    int color     = _BarColor & 0x0f;
    int bk        = (_BarColor>>4) & 0x0f;
    int lightedge = (_BarColor>>8) & 0x0f;
    int darkedge  = (_BarColor>>12) & 0x0f;
    int p         = range? (long)(*n)*(len-48)/range:0;

    /* -- 如果尚未画出滚动条,画滚动条后退出 ----- */
    if(*oldpos == -1)
    {
        _Bar(x,y,len,16,bk,0xffff0000);
        draw_rectangle(x,y,len,16,_BoxLineColor,0xff);
        _DrawBar(x,y,4,color,bk,lightedge,darkedge);
        _DrawBar(x+len-16,y,3,color,bk,lightedge,darkedge);
        _DrawBar(x+16+p,y,0,color,bk,lightedge,darkedge);

        /* -- 存储当前滑标位置 ----- */
        *oldpos = p;
        return ;
    }

    /* -- 如果无法移动滚动条则退出 ----- */
    if(range == 0)
        return ;

    /* -- 如果使用键盘控制滚动条 ----- */

```

```

if(key==KEY_Left+2048)
    (*n)--;

else if(key==KEY_Right+2048)
    (*n)++;
else if(key==KEY_PgUp+2048)
    (*n)-=pagesize;
else if(key==KEY_PgDn+2048)
    (*n)+=pagesize;
else if(key==KEY_Home+2048)
    (*n)=0;
else if(key==KEY_End+2048)
    (*n)=range;

/* -- 如果位置参数小于0,按0对待 ----- */
if((*n)<=0)
    (*n)=p=0;

/* -- 如果位置参数大于范围,按范围值对待 ----- */
else if((*n)>=range)
{
    (*n)=range;
    p=len-48;
}

/* -- 按位置参数计算出滑标的位置 ----- */
else
    p=(long)(*n)*(len-48)/range;

/* -- 移动滑标 ----- */
_MoveBarH(x+*oldpos,x+p,y,color,bk,lightedge,darkedge);

/* -- 存储当前滑标位置 ----- */
*oldpos=p;

/* -- 如果驱动键不是鼠标左键,则退出 ----- */
if(key!=LEFT_BUTTON)
    return ;

/* -- 如果鼠标指向滑标,则可拖动滑标 ----- */
if(mouse_enter(x+16+*oldpos,y,x+16+*oldpos+15,y+15))
{
    int lastmx=mousecol();

```

```

int old_ml = _MouseLeft;
int old_mt = _MouseTop;
int old_mr = _MouseRight;
int old_mb = _MouseBottom;
int mx, my;

set_mouse_range(lastmx - *oldpos, y + 1, lastmx - *oldpos + len - 48, y + 15);
while(get_mouse_status(&mx, &my) == LEFT_BUTTON)
    if(mx != lastmx && x + lastmx - mx > 0)
    {
        p = *oldpos + mx - lastmx;
        _MoveBarH(x + *oldpos, x + p, y, color, bk, lightedge, darkedge);
        lastmx = mx;
        *oldpos = p;
    }
set_mouse_range(old_ml, old_mt, old_mr, old_mb);
(*n) = (long)p * range / (len - 48);
p = (long)(*n) * (len - 48) / range;
_MoveBarH(x + *oldpos, x + p, y, color, bk, lightedge, darkedge);

/* -- 存储当前滑标位置 ----- */
*oldpos = p;
return ;
}

/* -- 如果鼠标指向滚动条条身,按比例移动滑标 ---- */
if (mouse_enter(x + 16, y, x + len - 16, y + 15))
{
    p = mousecol() - x - 22;
    if(p < *oldpos)
        (*n) -= pagesize;
    else
        (*n) += pagesize;
    if((*n) < 0)
        (*n) = 0;
    else if((*n) > range)
        (*n) = range;
    p = (long)(*n) * (len - 48) / range;
    _MoveBarH(x + *oldpos, x + p, y, color, bk, lightedge, darkedge);
    delay(_MouseSpeed);
}

/* -- 如果鼠标指向滚动条左块标,滑标向左移动一个单位 -- */

```



```

else if(mouse_enter(x,y,x+15,y+15))
{
    _DrawBar(x,y,14,color,bk,lightedge,darkedge);
    if((*n)>0)
        (*n)--;
    p = (long)(*n)*(len-48)/range;
    _MoveBarH(x+*oldpos,x+p,y,color,bk,lightedge,darkedge);
    delay(_MouseSpeed);
    _DrawBar(x,y,4,color,bk,lightedge,darkedge);
}
/* -- 如果鼠标指向滚动条右块标,滑标向右移动一个单位 -- */
else if(mouse_enter(x+len-15,y,x+len,y+15))
{
    _DrawBar(x+len-16,y,13,color,bk,lightedge,darkedge);
    if((*n)<range)
        (*n)++;
    if((*n)==range)
        p = len-48;
    else
        p = (long)(*n)*(len-48)/range;
    _MoveBarH(x+*oldpos,x+p,y,color,bk,lightedge,darkedge);
    delay(_MouseSpeed);
    _DrawBar(x+len-16,y,3,color,bk,lightedge,darkedge);
}

/* -- 存储当前滑标位置 -- -- -- -- -- */
*oldpos = p;
}

```

滚动条函数的使用比一般函数稍微复杂些。滚动条函数的参数表中,以下几个参数的设置最为重要且不易掌握:

pagesize——使用鼠标左键点击滚动条条身时受控制变量的增减数值,可以用来控制编辑器的翻页、快速查找等;

key——在调用程序中是使用什么键(geth、gethan等函数的返回值,包括键盘键和鼠标按钮)进入滚动条函数的。如果是下列编辑键值加上一个固定常数(对纵向滚动条来说是1536,对横向滚动条来说是2048),则可改变受控制变量的值:

KEY\_Up: 受控制变量值减1(纵向滚动条);  
 KEY\_Down: 受控制变量值加1(纵向滚动条);  
 KEY\_Left: 受控制变量值减1(横向滚动条);  
 KEY\_Right: 受控制变量值加1(横向滚动条);  
 KEY\_PgUp: 受控制变量值减页面长度;

KEY\_PgDn : 受控制变量值加页面长度;

KEY\_Home : 受控制变量值变为 0;

KEY\_End : 受控制变量值变为 range。

如果 key 之值为鼠标左键,就可能需要使用鼠标操纵滚动条,否则只需根据参数 n 的值调整滚动条上滑标的位置;

n——该参数是使用滚动条控制的整型变量的地址。为了方便起见,我们规定该变量的变化范围为 0—range 之间,即如果其值为 0,则滚动条中的滑标位于滚动条的最上(左)端,如果等于 range,则滑标位于滚动条的最下(右)端;

range——使用滚动条控制的量(参数 n)的最大值。由于该参数为整型,所以其值最大不能超过 32767,如果使用滚动条控制的量的实际变化范围大于该值,也要进行线性变换;

oldpos——这实际上是一个滚动条函数的工作变量,用来保存上次调用滚动条函数时滑标的位置。另外,我们还用这个变量表示是否是第一次调用滚动条函数。如果该参数的值为 -1,则表示是第一次调用滚动条函数,此时只是画出滚动条后即退出滚动条函数。

其实,从滚动条的应用效果就可以看出,使用滚动条必然需要如下程序结构:

```
unsigned key;
int      oldpos = -1;
... ..

    显示滚动条;
    循环((条件))
    {
        key = geth(); /* 或 key = gethan(); */
        ... ..

        调用滚动条函数以修改参数 n 的值;
    }
```

其中参数 n 可以用来控制表格的当前行和列、数据库的当前记录、当前字段等,具体应用的例子可以参看下节的代码表函数的源程序。

### 11.3 代码表

我们在设计 HANENV 系统时,为应用程序的设计人员提供了一套窗口式代码表函数。这些代码表函数使用方便,功能强大,可以构造多种其它功能部件。我们首先介绍它们的功能设计。

一个典型的代码表应用就是设计数据库系统的数据录入和查询界面。数据库中有些数据字段,例如人事数据库中的性别、职称、学历,财务数据库中的科目名称和成本项目,交通事故处理数据库中的事故类型,外事管理数据库中的国别等,只能取已知的有限种值,而且为了查询规范化,不能随意填写。例如,人事数据库中的“职称”字段,可能取值“高级工程师”、“工程师”、“助理工程师”、“技术员”等,但有人可能填写成“高工”、“助工”等,虽然不妨碍阅读理解,但使用数据库的查询语言进行检索时就可能因其写法不规范而无法检索出来。

通常的解决办法是采用编码,例如“高级工程师”的编码为1,“工程师”的编码为2,“助理工程师”的编码为3,“技术员”的编码为4等。但是这种方法不但给操作人员增加了记忆负担,而且很容易发生输入错误。如果利用代码表输入这类字段,以上问题即可迎刃而解。代码表使用鼠标进行操作,直观、方便,结果唯一。图11-3中给出了一个代码表的示例。

我们设计的代码表有以下特点和功能:

1. 代码表的设置方便、灵活,其显示位置和大小由参数表确定,而诸如表框和滚动条、文字、背景、光条、标题的颜色、标题和代码条目中汉字和字符的放大倍数以及双击鼠标按钮时的测试时间间隔等不需经常变化的指标均可以通过全局变量进行调整;

2. 代码表的功能较多。可以单选,也可以通过自行设计调用函数实现一次选择多个项目,还可以用去掉光条的方法制作在线帮助或提示模块;

3. 代码表的容量大,操作方便。由于我们设计的代码表采用分页显示,既可以使用鼠标操作滚动条,也可以使用键盘的光标移动键,特别是PgUp、PgDn、Home、End等,使得即使代码表的容量很大,也可以很快找到所需要的条目;

4. 适用面广。在代码条目内容比较规则,条目数量比较大的场合,可以使用自编代码条目生成函数的方法,以节约内存;在代码条目无法自动生成时也可以直接使用代码表数组,使用方便。

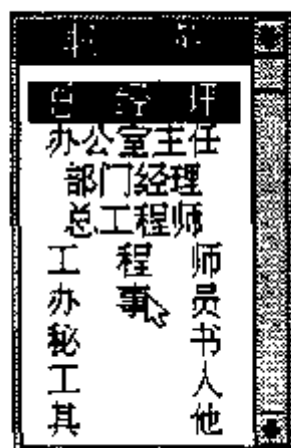


图11-3 代码表

#### 程序11-4 代码表函数。

由于代码表函数的功能强大,所以其程序结构也比较复杂。下面我们列出其简单工作流程:

保存代码表背景,显示代码表表体及其第一页的内容;

循环:

{

    根据鼠标或键盘信息修改代码表的当前页和当前条目;

    如果鼠标光标指向某条目且双击鼠标左键,则

        退出循环,该条目成为当前条目;

    如果按下回车键,则

        退出循环;

}

恢复代码表的背景;

返回当前代码条目的编号。

```
/* -----
函数 _GetCode : 代码表
----- */

#include <hanenv.h>
#include <string.h>
```

```

/* ----- 内部函数 _PutRecord : 显示代码条目 ----- */
static void _Cdecl _PutRecord(col,line,width,getrec,i,selected,key)
int col;                /* 代码条目显示列坐标(以字节为单位) */
int line;               /* 代码条目显示行坐标(以像素为单位) */
int width;              /* 代码条目显示宽度(以字节为单位) */
void (*getrec)();       /* 构造代码条目的函数(由用户提供) */
int i;                  /* 代码条目的序号 */
int selected;           /* 代码条目是否被选中 */
unsigned key;           /* 为构造代码条目的函数提供的键值 */
{
    /* -- 根据代码条目是否被选中,确定代码条目的显示颜色 -- */
    int color = selected? _EditColor: _TextColor;
    int bk    = selected? _EditBk: _Background;
    char code[81];

    /* -- 清代码条目背景,构造并显示代码条目 ----- */
    _Block(col+1,line,width, _Ytimes * CHAR _HIGH,bk);
    (*getrec)(i,code,key);
    outxystr(col+1,line,color,code);
}

/* ----- 函数 _GetCode : 窗口式代码表函数 ----- */
unsigned _Cdecl _GetCode(col,line,width,high,title,n,getrec,no)
int col;                /* 代码表左上角列坐标(以字节为单位) */
int line;               /* 代码表左上角行坐标(以像素为单位) */
int width;              /* 代码表宽度(以字节为单位) */
int high;               /* 代码表高度(以像素为单位) */
char * title;           /* 代码表标题 */
int n;                  /* 代码表长度(代码条数) */
void (*getrec)();       /* 构造代码条目的函数(由用户提供) */
int * no;               /* 选中的代码序号 */
{
    char ** block;       /* 存储代码表背景的缓冲区指针 */
    unsigned key;
    int i;
    int page_top;        /* 当前页首代码编号 */
    int curr_rec = 0;    /* 光条(当前)代码编号 */
    int have_cursor = iscursorlight();
    int mouse_light = ismouselight();
    int char_high = _Ytimes * CHAR _HIGH;
    int dispwidth = _Xtimes * width - 2;
    int oldpos = -1;     /* 滚动条用临时变量 */
    int sblen = high * _Ytimes * CHAR _HIGH - 6;

```

```

/* —— 如果取光条颜色与正文颜色相同,可构造帮助提示框 —— */
int onlyshow = (_EditColor == _TextColor && _EditBk == _Background);

/* ----- 如果代码表为空则退出 ----- */
if(n<=0 || getrec==NULL)
    return KEY_ESC;

/* ----- 关闭光标和鼠标 ----- */
if(have_cursor)
    delightcursor();
if(mouse_light)
    delight_mouse();

/* ----- 显示代码表表框 ----- */
block = getblock(col,line,dispwidth+4,high*char_high);
_Box(col*8,line,(dispwidth+4)*8,high*char_high);
scroll_v_box((col+dispwidth+2)*8-3,line+3,sblen,high,0,n,&curr_rec,
             &oldpos);
delight_mouse();

/* —— 如果有标题,则显示标题,然后调整参数 —— */
if(title)
{
    _Bar(col*8+4,line+4,(dispwidth+1)*8+1,char_high,_TitleBk,
        0xffff0000);
    outxystr(col+(dispwidth-strlen(title)*_Xtimes)/2+1,line+4*_Ytimes,
        _TitleColor,title);
    _H_Line(col*8+4,line+char_high+4,(dispwidth+1)*8+1,_BoxLineColor,
        0xff);
    high -= 2;
    line += (char_high*2-6*_Ytimes);
}
else
{
    high--;
    line += char_high-6*_Ytimes;
}

/* ----- 修正 n 的数值 ----- */
if(*no<0)
    *no = 0;
else if(*no>=n)
    *no = n-1;

```

```

/* ----- 显示代码表第一页的内容 ----- */
page_top = (*no/high) * high;
for(i=0; i<high && i+page_top<n; i++)
    _PutRecord(col, line+i * char_high, dispwidth, getrec, page_top+i, NO, 0);

/* ----- 将第一条代码设置为光条色 ----- */
i = *no%high;
if(! onlyshow)
    _PutRecord(col, line+i * char_high, dispwidth, getrec, page_top+i, YES, 0);

/* --- 主循环: 根据键盘和鼠标信息移动光条和选择代码 --- */
light_mouse();
while((key = geth()) != KEY_ESC
      && key != KEY_Left && key != KEY_Right)
{
    /* - 如果不是帮助提示框, 用回车键选择光条指向的代码 - */
    if(! onlyshow && key == KEY_ENTER)
        break;

    /* --- 如果是鼠标右键, 退出代码表 ----- */
    if(key == RIGHT_BUTTON)
    {
        till_mouse_pop(RIGHT_BUTTON);
        break;
    }
    delight_mouse();

    /* --- 取消原来的光条, 准备根据键码决定新光条的位置 --- */
    if(! onlyshow)
        _PutRecord(col, line+i * char_high, dispwidth, getrec, i+page_top, NO, 0);

    /* --- 根据输入的键码决定是选择代码还是移动光条 --- */
    switch(key)
    {
        case LEFT_BUTTON: /* 光条移至鼠标指向的代码 */
            if(! onlyshow)
            {
                int tmphigh = n-page_top>high? high:n-page_top;

                if(mouse_enter(col * 8,
                               line, (col+dispwidth) * 8, line+tmphigh * char_high))
                {
                    i = (mouserow()-line)/char_high;

```

```

    /* -- 双击鼠标左键表示选中该条目 -- -- */
    if(double_press(key))
        ungeth(KEY_ENTER);
    /* -- 单击鼠标左键使光条移向该条目并在 允许重
        复选择时向 getrec 函数送空格 */
    else
        key = ' ';
    }
}
break;
case KEY_Down:          /* 向下键:光标移向下一代码 */
    if(onlyshow)
        i = high-1;
    if(i<high-1 && i+page_top<n-1)
        i++;
    else if(page_top+high<n)
    {
        _MoveImage(col+1,line+char_high,dispwidth,(high-1)*char_high,
                    col+1,line);
        page_top++;
        if(onlyshow)
            _PutRecord(col,line+i*char_high,dispwidth,getrec,i+page_top,
                        NO);
    }
    break;
case KEY_Up:           /* 向上键:光标移向上一代码 */
    if(onlyshow)
        i = 0;
    if(i)
        i--;
    else if(page_top)
    {
        _MoveImage(col+1,line,dispwidth,(high-1)*char_high,col+1,
                    line+char_high);
        page_top--;
        if(onlyshow)
            _PutRecord(col,line+i*char_high,dispwidth,getrec,i+page_top,
                        NO);
    }
    break;
case KEY_PgUp:         /* 上翻页键:向上翻页 */
    if(page_top)

```

```

{
    page_top = page_top > high ? page_top - high : 0;
    _Block(col+1, line, dispwidth, high * char_high,
        _Background);
    for(i=0; i<high && i+page_top<n; i++)
        _PutRecord(col, line+i * char_high, dispwidth, getrec, page_top+i, NO,
            0);

    i = 0;
}
break;
case KEY_PgDn:          /* 下翻页键: 向下翻页 */
    if(page_top+high<n)
    {
        page_top += high;
        _Block(col+1, line, dispwidth, high * char_high, _Background);
        for(i=0; i<high && i+page_top<n; i++)
            _PutRecord(col, line+i * char_high, dispwidth, getrec, i+page_top, NO,
                0);

        i = 0;
    }
    break;
case KEY_Home:          /* Home 键: 光条指向代码表首代码 */
    if(page_top)
    {
        page_top = 0;
        for(i=0; i<high && i+page_top<n; i++)
            _PutRecord(col, line+i * char_high, dispwidth, getrec, i+page_top, NO,
                0);
    }
    i = 0;
    break;
case KEY_End:           /* End 键: 光条指向代码表末代码 */
    if(page_top+high<n)
    {
        _Block(col+1, line, dispwidth, high * char_high, _Background);

        page_top = n-1;
        i = 0;
    }
    else
        i = n-page_top-1;
    break;
}

```



```

/* —— 计算光条指向的代码的位置:页首位置+页内偏移 —— */
curr_rec = page_top+i;

/* —— 调用滚动条函数修正当前代码位置变量的值 —— */
light_mouse();
scroll_v_box((col+dispwidth+2)*8-3,(line-char_high*(title? 2:1)+6*_Ytimes)+3,sblen,high,key,n-1,&curr_rec,&oldpos);
delight_mouse();

/* —— 根据当前代码位置变量的值调整页首位置和页内偏移 —— */
if(curr_rec != page_top+i)
{
    if(curr_rec-page_top >= high || curr_rec-page_top < 0)
    {
        page_top = curr_rec/high*high;
        _Block(col+1,line,dispwidth,high*char_high,_Background);
        for(i=0;i<high&& i+page_top<n;i++)
            _PutRecord(col,line+i*char_high,dispwidth,getrec,i+page_top,NO,
                key,0);
    }
    i = curr_rec-page_top;
}
/* —— 将页内偏移指向的代码条目用光条颜色重新显示 —— */
if(! onlyshow)
    _PutRecord(col,line+i*char_high,dispwidth,getrec,i+page_top,YES,key);
light_mouse();
}
/* —— 将选中代码的位置=页首位置+页内偏移送入输出变量 —— */
*no = page_top+i;

/* —— 恢复代码表下面的屏幕内容 —— */
if(block)
{
    if(title)
    {
        high += 2;
        line -= 2*char_high-6*_Ytimes;
    }
    else
    {
        high ++;
        line -= char_high-6*_Ytimes;
    }
}

```

```

    delight_mouse();
    putblock(col,line,dispwidth+4,high*char_high,block);
    light_mouse();
}
/* ----- 如果原来有光标,则恢复光标显示 ----- */
if(have_cursor)
    lightcursor();
return key;
}

```

在使用上面的代码表函数时,要注意参数 `getrec` 的应用。`getrec` 是一个指向函数的指针。在应用上面的代码表函数时,必须首先编写一个生成代码条目的函数,然后将其地址作为参数 `getrec` 的值。该函数应该具有下列格式:

```

void fun(i,code,key)
int i;           /* 代码条目的序号 */
char *code;      /* 生成的代码条目,长度不超过 80 个字符 */
unsigned key;    /* 调用该函数之前接收的键盘或鼠标按钮
                  键码值(127=单击鼠标左键) */
{
    .....
    strcpy(code,...);
}

```

即在函数中要将生成的第 *i* 个代码条目的内容拷贝到 `code` 中去。

为了使读者更加清楚地了解代码表函数的使用方法,下面我们举一个例子说明上述自定义代码条目生成函数的设计方法。

#### 程序 11-5 ASCII 码表。

实际上,ASCII 码表是我们利用上面的代码表函数为 HANENV 系统设计的一个功能部件,其用途是在屏幕上显示一个代码表式提示窗口,内容为标准和扩展 ASCII 码的码值(包括 10 进制数与 16 进制数)和图象(见图 11-4)。请读者注意其中的代码条目生成函数 `_GetAsciiItem` 的设计。

```

/* -----
   模块 ascii_list : 查询 ASCII 码表
   ----- */
#include <hanenv.h>

void _Cdecl _GetAsciiItem(i,code)
int i;
char *code;

```

```

{
    sprintf(code," %3d    %02XH    %c",i,i,i);
}

unsigned _Cdecl ascii_list(col,line,no)
int col;
int line;
int * no;
{
    return _GetCode(col,line,27,10,"DEC HEX ASCII CODES",256,_GetAsciiItem,
                    no);
}

```

这个 `ascii_list` 函数除了让操作人员在屏幕上查看 ASCII 码表的内容以外,还可以通过使用鼠标左键双击表中条目或使用回车键来得到被选中的条目的编号(在本例中恰好是被选中的 ASCII 码的码值)。如果实际应用时不需要这一功能,而只需要在屏幕上查看 ASCII 码表,可以通过修改全局变量的方法解决。方法是分别将全局变量 `_EditColor` 和 `_EditBk` 的值设置为与全局变量 `_TextColor` 和 `_Background` 的值相同,使光条消失即可。其原理可以参看代码表函数 `_GetCode` 中有关变量 `onlyshow` 的处理。

DEC	HEX	ASCII CODES
0	00H	
1	01H	␣
2	02H	␣
3	03H	♥
4	04H	♦
5	05H	♠
6	06H	♣
7	07H	.

图 11-4 ASCII 码表

另外,在 IIANENV 系统提供的库函数中,还有一个查询区位码的库函数 `SP_list`,其代码条目生成函数的设计稍微复杂一些。有兴趣的读者可以直接参考本书所附软盘上该函数的源程序。

在代码条目无法自动生成的场合,可以使用下面的这个代码表函数。该函数要求提供一个存有代码条目的字符型指针的数组作为参数。

#### 程序 11-6 使用字符型指针数组参数的代码表函数。

该函数仍然是建立在代码表函数 `_GetCode` 上面的。但该函数增加了一个新的功能,即可以通过给代码条目前面加上标记的方式实现一次选择多个代码条目。只要操作人员使用鼠标左键单击某代码条目,或者按下空格键,都可以在当前代码条目之前加上(或消除)一个星号。当然,这时要注意所提供的代码条目的第一个字节应该置成空格。

```

/* -----
   函数 getcode: 代码表
   ----- */
#include <hanenv.h>

```

```

#include <string.h>

static char * *_CodeList = NULL;
static int _MultiSelect = NO;

/* —— 内部函数 _NthRecord : 构造代码表的第 i 个代码条目 —— */
static void _Cdecl _NthRecord(i, code, key)
int i;                                /* 代码条目的序号 */
char * code;                          /* 带回构造好的代码条目的参数 */
unsigned key;                         /* 检查是否多选的键码 */
{
    char * p = _CodeList[i];

    if(_MultiSelect && key == KEY_SPACE)
        *p = *p == ' '? '?' : ' ';
    strcpy(code, _CodeList[i]);
}

/* —— 函数 getcode : 窗口式代码表函数 —— */
unsigned _Cdecl getcode(col, line, width, high, title, n, codelist, no, mark)
int col;                             /* 代码表左上角列坐标(以字节为单位) */
int line;                            /* 代码表左上角行坐标(以像素为单位) */
int width;                           /* 代码表宽度(以字节为单位) */
int high;                            /* 代码表高度(以像素为单位) */
char * title;                        /* 代码表标题 */
int n;                               /* 代码表长度(代码条数) */
char * codelist[];                  /* 代码表 */
int * no;                            /* 选中的代码序号 */
int mark;                            /* 是否允许多项重复选择 */
{
    _CodeList = codelist;
    _MultiSelect = mark;
    return _GetCode(col, line, width, high, title, n, _NthRecord, no);
}

```

下面我们举例说明代码表函数 getcode 的使用方法,在这个应用场合中没有使用“多选”功能。

#### 程序 11-7 输入职称的代码表。

```

/* -----
    测试代码表的程序。

```

```

----- */
#include <hanenv.h>
char *code[] =
{
    "高级工程师",
    "工程师",
    "助理工程师",
    "技术员",
    "其他"
};

main()
{
    unsigned key;
    int n;
    char tmp[81];

    /* ----- 初始化 HANENV 系统 ----- */
    init_hanenv();
    init_XMS();
    _CurrentHZK = load_HZK(2,16,"hzk16j",NULL,94*87,XMS);
    if(_CurrentHZK == NULL)
    {
        close_hanenv();
        exit(0);
    }

    /* ----- 用绿色清屏幕 ----- */
    delight_mouse();
    _Block(0,0,80,480,GREEN);
    .light_mouse();

    /* ----- 设置代码表的工作参数 ----- */
    _TextColor = BLUE;           /* 代码条目显示颜色 */
    _Background = WHITE;        /* 代码表背景颜色 */
    _TitleColor = WHITE;        /* 标题正文颜色 */
    _TitleBk = BLUE;            /* 标题背景显示颜色 */
    _EditColor = YELLOW;        /* 光条正文颜色 */
    _EditBk = RED;              /* 光条颜色 */
    _Xtimes = 1;                /* 字体横向放大倍数 */
    _Ytimes = 1;                /* 字体纵向放大倍数 */

    /* ----- 循环测试代码表函数 ----- */
    do

```

```

{
    light_mouse();
    key = getcode(10,100,20,7,"职 称",5,code,&n,NO);
    delight_mouse();
    sprintf(tmpline,"key = %d, n = %d",key,n);
    putxys(40,100,RED,WHITE,tmpline);
    geth();
}while(key! =KEY_ESC);
/* ----- 退出 HANENV 系统 ----- */

_CurrentHZK = free_HZK(_CurrentHZK);
close_hanenv();
}

```

该程序的运行结果见图 11-3。

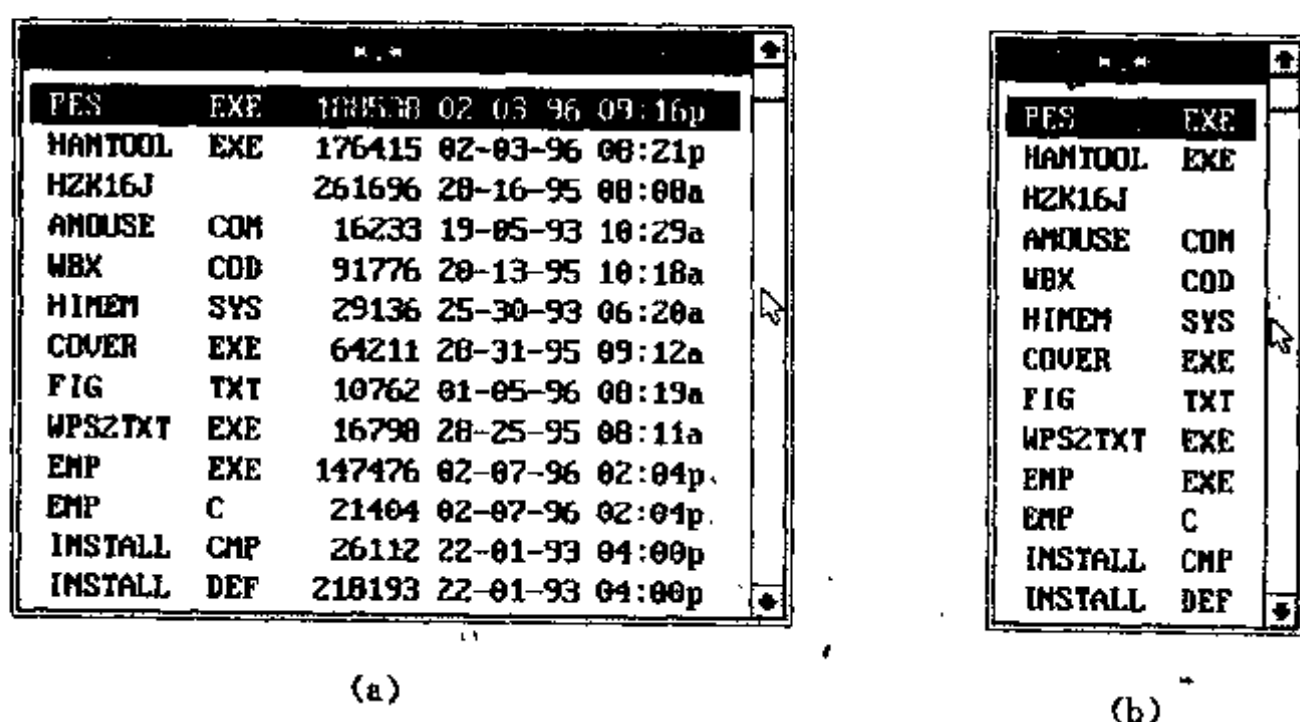


图 11-5 目录选拔菜单

在应用程序中,经常需要对磁盘中的一个或一批文件进行操作。通常的作法是要求用户输入文件名,然后再对文件操作。这种处理方式不仅操作复杂,易于出错,而且在用户不能确定需要操作的文件名的情况下还必须退出应用程序,使用 DOS 命令查看需要操作的文件名,费时费力。下面我们介绍一个使用代码表函数编写的目录选择菜单函数,使用该函数可以很方便地使用鼠标或键盘在指定路径中选择需要对之操作的文件,见图 11-5。

#### 程序 11-8 文件目录选择菜单函数。

我们在设计文件目录选择菜单函数时尽量考虑了各种应用场合的需要,功能设置相当完善。该函数的主要功能指标为

1. 可以方便地通过设置参数,或修改全局变量的值确定文件目录选择菜单的位置、大小和颜色,适应不同应用场合;
2. 代码条目的设置分为简化和完整两种格式,前者包括文件名和后缀,后者除此而外还

包括文件的长度、建立(修改)日期和时间(分别见图 11-5a 和图 11-5b);

3. 可以通过参数指定需要列文件目录的路径(文件名部分可以使用通配符)和文件属性;

4. 可以设置多选参数,在允许同时选择多个文件的情况下可以使用鼠标左键单击相应文件条目或者使用空格键在当前文件条目之前做一个星号标记(或取消星号标记),在使用鼠标左键双击最后一个选中的文件条目后由预先指定的用户自定义函数对所有被选中的文件进行处理。

```

/* -----
   函数 dirmenu : 文件目录选择菜单
   ----- */

#include <hanenv.h>
#include <dir.h>
#include <fcntl.h>
#include <alloc.h>
#include <string.h>

/* ----- 内部函数 _TranFileitem : 生成文件条目 ----- */
static void _TranFileitem(char *dest, struct ffblk f, int full_width)
{
    char *point = strchr(f.ff_name, '.');

    /* -- 初始化条目串 ----- */
    memset(dest, ' ', 14);
    dest[14] = 0;
    /* -- 将文件名及其后缀拷贝到条目中 ----- */
    if(point)
    {
        memcpy(dest+1, f.ff_name, point-f.ff_name);
        memcpy(dest+10, point+1, strlen(point+1));
    }
    else
        memcpy(dest+1, f.ff_name, strlen(f.ff_name));

    /* -- 如果是全宽度显示,将文件长度及更新时间拷贝到条目中 */
    if(full_width)
    {
        int year  = (f.ff_fdate>>9)+80;
        int month = (f.ff_fdate>>5)&0x1f;
        int day   = f.ff_fdate&0x1f;
        int hour  = (f.ffftime>>11)&0x1f;
        int minate= (f.ffftime>>5)&0x1f;
    }
}

```

```

        int noon = hour>12?'p':'a';
        hour     = hour>12? hour-12:hour;
        sprintf(dest+14,"%8ld %02d-%02d-%02d %02d:%02d%c",f.ff_ftime,
                month,day,year,hour,minate,noon);
    }
}

/* -----
   函数 dir_menu : 文件目录选择菜单
   ----- */

unsigned _Cdecl dir_menu(col,line,high,path,attrib,full_width,fun,multiselect)
int col;           /* 目录选择菜单左上角列坐标(单位为字节) */
int line;          /* 目录选择菜单左上角行坐标(单位为象素) */
int high;          /* 目录选择菜单高度(单位为正文行) */
char * path;       /* 备选文件路径(可使用通配符) */
int attrib;        /* 备选文件属性 */
int full_width;    /* 是否显示文件长度、日期等 */
void (* fun)();    /* 指向处理被选中的文件的函数指针 */
int multiselect;   /* 是否允许选择多个文件 */
{
    unsigned key;
    struct fblk fblk;
    char ** dir_list;
    char filename[MAXPATH];
    char drive[MAXDRIVE];
    char dir[MAXDIR];
    char * nameptr = filename;
    int i;
    int dir_count = 1;
    int no = 1;
    int width = full_width? 40 : 14;

    /* -- 如果没有符合条件的文件则退出 ----- */
    if(findfirst(path,&fblk,attrib))
        return NULL;

    /* -- 取符合条件的文件数目 ----- */
    while(findnext(&fblk)==0)
        dir_count++;

    /* -- 分解路径,并将其拷贝到最终文件名的前面 ---- */
    fnsplit(path,drive,dir,NULL,NULL);
    if(drive[0])
    {

```



```

    sprintf(nameptr,"%s",drive);
    nameptr += strlen(drive);
}
if(dir[0])
{
    sprintf(nameptr,"%s",dir);
    nameptr += strlen(dir);
}

/* ----- 构造文件目录表 ----- */
if(coreleft() < dir_count * (width+2))
    return KEY_ESC;
dir_list = (char **) malloc(sizeof(char *) * dir_count);
for(i=0; i < dir_count; i++)
    dir_list[i] = (char *) malloc(sizeof(char) * width);

findfirst(path,&ffblk,attrib);
i = 0;
do
{
    _TranFileitem(dir_list[i++],ffblk,full_width);
}while(findnext(&ffblk) == 0);

/* ----- 调用 getcode 函数选择文件 ----- */
key = getcode(col,line,width+2,high,path,dir_count,dir_list,&no,multiselect);

/* ----- 如果不是 ESC 键则构造选中文件表 ----- */
if(key == KEY_ENTER)
{
    memcpy(nameptr,dir_list[no]+1,12);
    nameptr[8] = '.';
    nameptr[12] = 0;
    skipspace(nameptr);
    if(fun)
    {
        *(dir_list[no]+1) = ' ';
        (*fun)(filename);
    }
}
for(i=0; i < dir_count; i++)
{
    if(multiselect && fun && *(dir_list[i]) == '*' && key == KEY_ENTER)
    {

```

```

        memcpy(nameptr, dir_list[i] + 1, 12);
        nameptr[8] = '.';
        nameptr[12] = 0;
        skip_space(nameptr);
        (*fun)(filename);
    }
    free(dir_list[i]);
}
free(dir_list);
return key;
}

```

在应用程序中使用上面的文件目录选择菜单函数时要注意参数 fun(文件处理函数)的选用。参数 fun 是一个指向函数的指针, 必须指向一个用户自己编写的文件处理函数。该函数的格式应该为

```

void processfiles(char * filename)
{
}

```

注意由函数 dir\_menu 提供的文件名中已带有路径。如果允许多选, 函数 dir\_menu 在结束文件选择之后会自动调用该函数逐个处理被选中的文件, 而第一个被处理的文件就是结束选择时所选定的那个文件。如果不允许多选, 则 dir\_menu 调用上述用户自定义函数处理被选中的文件。

## 11.4 调色板

在第 1 章中我们已经介绍过, VGA 显示卡的图形模式 12H 虽然只允许同屏显示 16 种不同的颜色, 但是我们可以通过 BIOS 的显示器中断 10H 或者 VGA 卡上的图形控制器的寄存器组修变这些颜色实际值。这样, 我们既可以通过使这些变量集中于某个色彩区域来实现丰富的色彩层次, 也可以将某种颜色修改为特别的色彩, 例如人的肤色, 以取得特殊的屏幕效果。实际上, 已经有在 16 色模式下通过修改省缺颜色配置而模拟显示 256 色图象的算法出现, 实际效果差强人意。

调整某个省缺颜色的色调是通过修改该颜色寄存器的红、绿、蓝三种原色的分量进行的。图形模式 12H 允许每种原色分量在 0—63 的范围内变化, 因此我们最多可以调出  $63 \times 63 \times 63 = 256K$  种不同的色彩来! 可以想象, 通过编写程序修改指定颜色——观察屏幕显示效果——再修改程序中的颜色参数的方法要从如此丰富的色彩中选出自己最中意的色彩是一项相当困难的工作。因此我们设计了一个调色板工具(见图 11-6)来帮助应用程序设计人员完成这项工作。该函数还可以作为热键命令(例如利用 HANENV 系统的触发器功能)直接挂到应用程序中, 使得应用程序的操作人员也可以根据自己的喜好修改屏幕的显示效果。

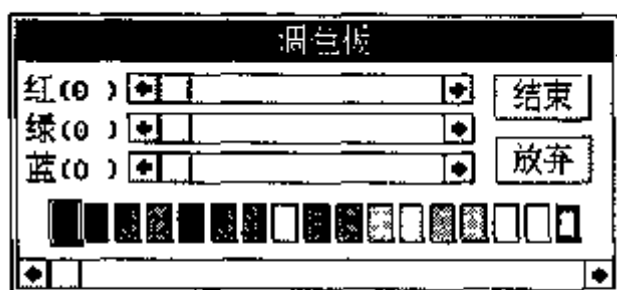


图 11-6 调色板工具

这方面的例子可以参考第 5 章有关工具软件 hantool 的说明部分。

#### 程序 11-9 调色板工具

在第 1 章中我们已经介绍过,在 VGA 的图形模式 12H 下除了可以同屏显示 16 种颜色以外,还可以通过控制屏幕边缘色彩寄存器为显示屏幕加上一个指定颜色的边框。该寄存器一般被设置为指向 16 种颜色中的第 0 号颜色,即背景色(缺省值为黑色)。在我们设计的调色板工具函数中已经考虑了将屏幕边缘颜色作为一个独立的颜色进行调整,因此在应用程序中一屏上最多可以调出 17 中不同的色彩来。

从图 11-6 中可以看出,在调色板下方有一排各种颜色的小方块,共有 17 个,前 16 个代表当前屏幕上可以使用的 16 种色彩,最右边的一个中空黑色方框代表屏幕边缘颜色。在最左端的黑色小方块外面还套着一个矩形框,使用鼠标操纵调色板最下方的滚动条就可以通过移动矩形框来选择要修改的屏幕色彩。在调色板上还有 3 个横向滚动条,分别用于调整被选中的颜色的红、绿、蓝三原色分量的强度(其分量值在滚动条左边的括号中显示),调整的效果可以通过被选中的颜色块的变化反映出来(但屏幕边缘色变化的效果只能通过直接观察屏幕边缘得出)。

```

/* -----
   函数 palette : 调色板
   ----- */

#include <hanenv.h>
#include <stdlib.h>

unsigned _Cdecl palette(col,line,buffer)
int col;           /* 调色板左上角列坐标(以字节为单位) */
int line;          /* 调色板左上角行坐标(以像素为单位) */
char * buffer;     /* 调色板数据缓冲区(共需 51 字节) */
{
    int oldposr = -1; /* 红分量调整滚动条工作变量 */
    int oldposg = -1; /* 绿分量调整滚动条工作变量 */
    int oldposb = -1; /* 蓝分量调整滚动条工作变量 */
    int oldposc = -1; /* 颜色选择滚动条工作变量 */
    int r,g,b;        /* 当前颜色的红绿蓝分量 */
    int old_r,old_g,old_b; /* 原先颜色的红绿蓝分量 */

```

```

char value[3];          /* 显示颜色分量值的字符串      */
unsigned key;           /* 键盘或鼠标信息变量      */
int i;                 /* 循环变量,当前颜色号    */
int old_i;             /* 选择颜色号前的原颜色号 */
int old_bk = _Background; /* 原来的背景色          */
BUTTON_TYPE buttons[2]; /* 结束/退出鼠标按键      */
char * * block;        /* 调色板背景存储缓冲区指针 */

/* ----- 设置鼠标按键的各种参数 ----- */
buttons[0].x          = 246;
buttons[0].y          = 30;
buttons[0].w          = 50;
buttons[0].b          = 24;
buttons[0].butncolor  = _BarColor;
buttons[0].pushcolor  = _BarColor;
buttons[0].fonttimes  = 257;
buttons[0].name       = "结束";
buttons[0].key        = KEY_ENTER;
buttons[0].pressed    = KEY_ENTER;
buttons[0].high       = 1;
buttons[0].lock       = NO;
buttons[0].fun        = NULL;

buttons[1].x          = 246;
buttons[1].y          = 62;
buttons[1].w          = 50;
buttons[1].h          = 24;
buttons[1].butncolor  = _BarColor;
buttons[1].pushcolor  = _BarColor;
buttons[1].fonttimes  = 257;
buttons[1].name       = "放弃";
buttons[1].key        = KEY_ESC;
buttons[1].pressed    = RIGHT_BUTTON;
buttons[1].high       = 1;
buttons[1].lock       = NO;
buttons[1].fun        = NULL;

/* ----- 存储调色板背景并显示调色板 ----- */
delight_mouse();
block      = getblock(col,line,39,143);
_Background = WHITE;
_Box(col*8,line,310,143);
_Bar(col*8+4,line+4,302,20,BLACK,0xffff0000);

```

```

outxys(col+17,line+5,WHITE,"调色板");
_DrawButton(buttons[0],col,line,NO,0);
_DrawButton(buttons[1],col,line,NO,0);

/* ----- 保存当前调色板数据,画出调色块 ----- */
for(i=0;i<17;i++)
{
    get_color(i,&r,&g,&b);
    buffer[i*3] = r;
    buffer[i*3+1] = g;
    buffer[i*3+2] = b;
    draw_rectangle(col*8+21+i*16,line+96,13,20,BLACK,0xff);
    _Bar(col*8+22+i*16,line+97,11,18,i,0xffff0000);
}
_Bar(col*8+280,line+100,7,12,WHITE,0xffff0000);

/* ----- 显示三原色调整滚动条 ----- */
i = 0;
draw_rectangle(col*8+19+i*18,line+94,17,24,BLACK,0xff);
get_color(i,&r,&g,&b);
outxys(col+1,line+30,BLACK,"红()");
scroll_h_box(col*8+60,line+30,176,8,0,63,&r,&oldposr);
outxys(col+1,line+50,BLACK,"绿()");
scroll_h_box(col*8+60,line+50,176,8,0,63,&g,&oldposg);
outxys(col+1,line+70,BLACK,"蓝()");
scroll_h_box(col*8+60,line+70,176,8,0,63,&b,&oldposb);
itoa(r,value,10);
outxys(col+4,line+30,BLACK,value);
itoa(g,value,10);
outxys(col+4,line+50,BLACK,value);
itoa(b,value,10);
outxys(col+4,line+70,BLACK,value);
scroll_h_box(col*8+3,line+124,304,1,0,16,&i,&oldposc);

/* ----- 主循环:根据滚动条调整调色板数据 ----- */
do
{
    light_mouse();
    key = geth();
    delight_mouse();
    if(key == LEFT_BUTTON)
    {
        /* ----- 如果选择结束键则跳出循环 ----- */

```

```

if(mouse_enter(col * 8 + 242, line + 30, col * 8 + 286, line + 54))
{
    _DrawButton(buttons[0], col, line, NO, 1);
    light_mouse();
    till_mouse_pop(LEFT_BUTTON);
    delight_mouse();
    _DrawButton(buttons[0], col, line, NO, 0);
    key = KEY_ENTER;
}
/* ----- 如果选择放弃键则跳出循环 ----- */
else if(mouse_enter(col * 8 + 242, line + 62, col * 8 + 286, line + 86))
{
    _DrawButton(buttons[1], col, line, NO, 1);
    light_mouse();
    till_mouse_pop(LEFT_BUTTON);
    delight_mouse();
    _DrawButton(buttons[1], col, line, NO, 0);
    key = KEY_ESC;
}
}

/* ----- 鼠标右键相当于 ESC 键 ----- */
else if(key == RIGHT_BUTTON)
{
    till_mouse_pop(RIGHT_BUTTON);
    key = KEY_ESC;
}

/* ----- 调整颜色选择滚动条 ----- */
old_i = i;
scroll_h_box(col * 8 + 3, line + 124, 304, 1, key, 16, &i, &oldposc);
if(old_i != i)
{
    draw_rectangle(col * 8 + 19 + old_i * 16, line + 94, 17, 24, WHITE, 0xff);
    draw_rectangle(col * 8 + 19 + i * 16, line + 94, 17, 24, BLACK, 0xff);
    get_color(i, &r, &g, &b);
    itoa(r, value, 10);
    _Block(col + 4, line + 30, 2, 18, WHITE);
    outxys(col + 4, line + 30, BLACK, value);
    itoa(g, value, 10);
    _Block(col + 4, line + 50, 2, 18, WHITE);
    outxys(col + 4, line + 50, BLACK, value);
    itoa(b, value, 10);
    _Block(col + 4, line + 70, 2, 18, WHITE);

```

```

        outxys(col+4,line+70,BLACK,value);
    }
    /* ----- 调整当前颜色的红分量 ----- */
    old_r = r;
    scroll_h_box(col*8+60,line+30,176,8,key,63,&r,&oldposr);
    if(old_r != r)
    {
        itoa(r,value,10);
        _Block(col+4,line+30,2,18,WHITE);
        outxys(col+4,line+30,BLACK,value);
        set_color(i,r,g,b);
    }
    /* ----- 调整当前颜色的绿分量 ----- */
    old_g = g;
    scroll_h_box(col*8+60,line+50,176,8,key,63,&g,&oldposg);
    if(old_g != g)
    {
        itoa(g,value,10);
        _Block(col+4,line+50,2,18,WHITE);
        outxys(col+4,line+50,BLACK,value);
        set_color(i,r,g,b);
    }
    /* ----- 调整当前颜色的蓝分量 ----- */
    old_b = b;
    scroll_h_box(col*8+60,line+70,176,8,key,63,&b,&oldposb);
    if(old_b != b)
    {
        itoa(b,value,10);
        _Block(col+4,line+70,2,18,WHITE);
        outxys(col+4,line+70,BLACK,value);
        set_color(i,r,g,b);
    }
}while(key!=KEY_ESC && key!=KEY_ENTER);
/* -- 如果是结束键则将当前调色板信息存入缓冲区 ----- */
if(key==KEY_ENTER)
    for(i=0;i<17;i++)
    {
        get_color(i,&r,&g,&b);
        buffer[i*3] = r;
        buffer[i*3+1] = g;
        buffer[i*3+2] = b;
    }
}

```

```

/* ----- 如果是放弃键则恢复原来的调色板信息 ----- */
else
    for(i=0;i<17;i++)
    {
        r = buffer[i * 3];
        g = buffer[i * 3 + 1];
        b = buffer[i * 3 + 2];
        set_color(i,r,g,b);
    }
/* ----- 恢复调色板下原来的背景 ----- */
if(block)
{
    delight_mouse();
    putblock(col,line,39,143,block);
}
_Background = old_bk;
light_mouse();
return key;
}

```

在调色板工具函数中我们设计了一个调色板数据缓冲区参数。该缓冲区用于从调色板函数中将调整后的调色板数据带出来。17 种颜色(16 种屏幕显示色和 1 种屏幕边缘颜色), 每种颜色的红、绿、蓝三原色分量值各用一个字节保存, 一共需要 51 个字节。在应用程序中可以将这些数据存储在文件中, 以备下次开机时设置屏幕色调之用。这方面的例子可以参看 HANENV 系统的工具软件 hantool 的源程序(见 HANENV 系统 1# 软盘)。

为了使应用程序具有拷贝屏幕图形的能力, 我们设计了一个屏幕图形区域拷贝函数 prnsr。该函数可以拷贝屏幕上指定矩形区域的内容, 并可指定放大倍数和左页边的宽度。值得指出的是我们为该函数设计了一个参数 palette, 用来指定屏幕拷贝时哪些颜色的像素可以被打印出来。该参数是一个 16 位的整数, 其 0—15 位分别对应于 VGA 图形模式 12H 的 16 种颜色, 如果要打印哪种颜色的像素, 就将参数 palette 的对应位置为 1, 否则置为 0。例如, 如果除了黑色, 其他颜色的像素均需打印的话, 可以将参数 palette 设置为 0xfffe。

#### 程序 11-10 屏幕图形拷贝。

我们是针对 Epson 系列打印机的图形命令来设计屏幕图形拷贝函数的, 同时也可以用于 NEC 系列和 AR3240 打印机。

```

/* -----
    函数 print_screen, 屏幕图形拷贝
    ----- */
#include <hanenv.h>
#include <string.h>

```



```

#define print(ch) biosprint(0,ch,0)
void _Cdecl print_screen(x1,y1,x2,y2,margin,palette)
int x1,y1;                /* 打印区域左上角坐标 */
int x2,y2;                /* 打印区域右下角坐标 */
int margin;               /* 左页边 */
unsigned palette;         /* 调色板: 1=打印 */
{
    int i,j,x,y,px,cols,color;
    unsigned char sum,mark,sum1[1024][3];
    /* -- 设置单项打印方式 -- */
    print(27);print(85);print(1);

    /* -- 设置 n/60 英寸换行量 -- */
    print(27);print(65);print(8);
    /* -- 设置行宽及像素尺寸 -- */
    cols = (x2-x1+1) * _Xtimes+margin;
    mark = 0xff >> (8-_Ytimes);
    /* -- 按行处理打印区域 -- */
    for(y=y1;y<=y2;y+=24/_Ytimes)
    {
        /* -- 按列处理打印区域 -- */
        for(x=x1;x<=x2;x++)
        {
            /* -- 生成 24 列打印数据 -- */
            for(j=0;j<3;j++)
            {
                sum = 0;
                for(i=0;i<8/_Ytimes;i++)
                {
                    if(y+j*8/_Ytimes+i<=y2)
                    {
                        color=_GetPixel(x,y+j*8/_Ytimes+i);
                        if((0x01<<color)&palette)
                            sum += mark<<((8-(i+1))*_Ytimes);
                    }
                }
                sum1[x][j] = sum;
            }
        }
        /* -- 设置 3 倍密度打印及每行宽度 -- */
        print(27);print(42);print(39);print(cols%256);print(cols/256);
        /* -- 空出左页边 -- */
        for(px=0;px<margin;px++)
        {
            for(j=0;j<3;j++)
            {
                print(0);
            }
        }
        /* -- 打印图形内容 -- */
        for(px=x1;px<=x2;px++)

```

```
    for(i=0;i<_Xtimes;i++)
        for(j=0;j<3;j++)
            print(sum1[px][j]);
        /* -- 换行 ----- */
        print('\n');
    }
}
```

从以上源程序中可以看出,放大倍数的设置是通过全局变量 `_Xtimes` 和 `_Ytimes` 进行的。

# 全屏幕数据编辑

在设计数据库应用软件时少不了要编写数据录入编辑界面程序。而编写数据录入编辑界面程序又少不了能够方便地对各种类型的数据字段进行屏幕编辑的函数。在本章中我们首先介绍一个基本数据字段编辑函数，然后在此基础上发展出一系列适合于对诸如整型、浮点型、日期型、字符型等各种类型的数据字段进行编辑的函数。最后，通过一个示例介绍如何使用这些函数编写出漂亮美观的全屏幕数据录入编辑界面程序来。

### 12.1 基本数据编辑函数

全屏幕数据编辑的基本单位是数据域。数据域的属性有数据类型、编辑位置和区域、数据值以及数据值的范围等，其中最重要的属性是数据类型。不同类型的数据必需使用不同类型的变量和参数表示，使用不同的符号集合和数据范围，很难用一个编辑函数统一处理。因此，我们对不同类型的数据分别编写了各自的数据编辑函数。这些函数中的大多数通过调用一个基本数据编辑函数完成对数据域的各种编辑功能，自身只需处理参数类型转换、符号集合和数据范围的检验等工作。

#### 程序 12-1 基本数据编辑。

基本数据编辑函数的编辑对象是一个字符串。在编辑时可以使用参数指定编辑区域的大小及其在屏幕上的位置(颜色由全局变量确定)和编辑格式。所谓编辑格式是一个与编辑区域等长的字符串，其每位对应一个编辑位置，说明该编辑位置上允许使用的符号类型。我们定义的格式符有：

- A —— 该编辑位置上只允许出现字母；
- 9 —— 该编辑位置上只允许出现数字和正、负号；
- . —— 该编辑位置上只允许出现小数点；
- , —— 该编辑位置上只允许出现逗号；

其他字符(例如空格)表示对应的编辑位置上允许出现任何可见字符。

基本数据编辑函数的程序结构如下:

将待编辑字符串拷贝到编辑缓冲区中;

将编辑格式串拷贝到格式缓冲区中并进行修正;

显示待编辑的字符串;

循环: 根据键盘输入键编辑缓冲区内容

{

    接收一个键盘码至变量 h;

    如果 h 是扩展键盘码(编辑命令), 则

        调整编辑指针和编辑区内容;

    如果 h 是 ASCII 码中的可见字符且与格式串相应位置的格式符相容, 则

        h 进入编辑缓冲区, 调整缓冲区指针;

    如果 h 是结束编辑命令, 则

        跳出循环;

}

如果 h 为 ESC 键, 则

    放弃被编辑的内容;

以原正文和背景颜色显示被编辑的字符串;

```
/* -----
   函数 getxya : 基本数据编辑
   ----- */
#include <hanenv.h>
#include <string.h>
#include <ctype.h>

unsigned _Cdecl getxya(col,line,width,s,picture)
int col;           /* 编辑窗口左上角列坐标(以字节为单位) */
int line;          /* 编辑窗口左上角行坐标(以象素为单位) */
int width;         /* 编辑窗口宽度 */
char *s;           /* 被编辑的字符串 */
char *picture;     /* 编辑格式 */
{
    unsigned char buff[81];           /* 编辑缓冲区 */
    unsigned char pict[81];          /* 格式缓冲区 */
    unsigned char *bptr = buff;      /* 编辑指针 */
    unsigned char *brear = buff+width-1; /* 编辑区尾部指针 */
    int _EndEdit = NO;               /* 结束编辑标记 */
    int modified = NO;               /* 是否修改过 */
    int old_c = _TextColor;          /* 原正文颜色 */

```

```

int old_b          = _Background;    /* 原背景颜色    */
int len            = strlen(s);       /* 长度临时变量 */
int have_cursor    = iscursorlight(); /* 光标已打开    */
int mouselight     = ismouselight();  /* 鼠标已打开    */
unsigned h;        /* 键盘输入键    */

/* -- 将待编辑字符串拷贝到编辑缓冲区中 ----- */
len = len > width ? width : len;
memset(buff, ' ', width);
memcpy(buff, s, len);
buff[width] = 0;

/* -- 将编辑格式串拷贝到格式缓冲区中 ----- */
if (picture)
{
    len = strlen(picture);
    len = len > width ? width : len;
    memset(pict, ' ', width);
    memcpy(pict, picture, len);
    pict[width] = 0;
}

/* -- 设置工作环境参数,显示待编辑的字符串 ----- */
set_text_color(_EditColor);
set_background(_EditBk);
_TextCol = col;
_TextLine = line;
if (mouselight)
    delight_mouse();
putnstr(buff, width);
if (mouselight)
    light_mouse();
if (! have_cursor)
    lightcursor();

/* -- 按下任意键后清编辑缓冲区 ----- */
if ((h = getch()) == LEFT_BUTTON || h == RIGHT_BUTTON)
    h = 400;
else if (isprint(h))
    memset(buff, ' ', width);
    ungeth(h);

/* -- 根据键盘输入键编辑缓冲区内容 ----- */
do

```

```

{

/* -- 接收一个键盘码 ----- */
if((h==geth())<=3)
    h += 400;

/* -- 根据键盘码编辑 ----- */
if(mouselight)
    delight_mouse();
switch(h)
{
    case KEY_Left:                /* 左箭头 */
        if(bptr==buff)
            _EndEdit = YES;
        else
        {
            bptr--;
            _TextCol -= _Xtimes;
        }
        break;
    case KEY_Right:               /* 右箭头 */
        if(bptr >= brear)
            _EndEdit = YES;
        else
        {
            bptr++;
            _TextCol += _Xtimes;
        }
        break;
    case KEY_Home:                /* Home 键 */
        if(bptr>buff)
        {
            bptr = buff;
            _TextCol = col;
        }
        else
            _EndEdit = YES;
        break;
    case KEY_End:                /* End 键 */
        if(bptr<brear)
        {
            bptr = brear;
            _TextCol = col+(width-1)*_Xtimes;

```

```

    }
    else
        _EndEdit = YES;
    break;
case KEY_Del: /* 删除键 */
    memmove(bptr, bptr+1, brear-bptr+1);
    *brear = ' ';
    putnstr(bptr, brear-bptr+1);
    modified = YES;
    break;
case Backspace: /* 退格键 */
    if(bptr>buff)
    {
        bptr--;
        _TextCol -= _Xtimes;
        memmove(bptr, bptr+1, width-(bptr-buff));
        *(buff+width-1) = ' ';
        putnstr(bptr, width-(bptr-buff));
    }
    modified = YES;
    break;
default: /* 其他键 */

/* -- 如果是功能键,退出编辑 ----- */
if(isedit(h) || h==KEY_ENTER || h==KEY_ESC
    || h>=LEFT_BUTTON)
{
    _EndEdit = YES;
    break;
}

/* -- 如果有格式串,按其进行编辑 ----- */
if(picture)
{
    switch( *(pict+(bptr-buff)) )
    {
        case 'A':
        case 'a':
            if(! isalpha(h) && h!= ' ')
                goto _EndLoop;
            break;
        case '9':
            if(! isdigit(h) && h!= '-' && h!= '+' && h!= '.')
                goto _EndLoop;
    }
}

```

```

        break;
    case '.':
        h = '.';
        break;
    case ',':
        h = ',';
        break;
    }
}
/* -- 如果是可编辑字符,进入编辑缓冲区 ----- */
if(isprint(h) && bptr-buff<=width)
{
    if(bptr<=brear)
    {
        if(isins())
            movmem(bptr,bptr+1,brear-bptr);
        *bptr++ = h;
        putnstr(bptr-1,brear-bptr+2);
        _TextCol += _Xtimes;
    }
    if(bptr-buff==width)
    {
        _EndEdit = YES;
        h = KEY_Right;
    }
    modified = YES;
}
_EndLoop;;
}
/* -- switch 语句结束 ----- */
if(mouselight)
    light_mouse();
}while(! _EndEdit);

/* -- 如果是 ESC 键或鼠标右键,放弃被编辑的内容 ----- */
if(h!=KEY_ESC && h!=RIGHT_BUTTON && modified)
{
    strepy(s,buff);
    h += 1024;
}
/* -- 以原正文和背景颜色显示被编辑的字符串 ----- */
delightcursor();
if(mouselight)

```



```

    delight_mouse();
    _TextCol = col;
    set_text_color(old_c);
    set_background(old_b);
    putnstr(buff,width);
    if(mouselight)
        light_mouse();
    if(have_cursor)
        lightcursor();

    /* --- 返回退出编辑状态的键盘码 --- */
    return h;
}

```

请注意该函数的返回值为退出编辑时所按下的按键或鼠标按钮。从程序中可以看出,使用以下键盘按键和鼠标按钮值可以退出编辑状态:

KEY\_ESC 键和鼠标右键 RIGHT\_BUTTON 用于作废本次编辑工作,恢复被编辑数据原来的值;

KEY\_ENTER 键、鼠标左键 LEFT\_BUTTON 和除了 KEY\_Left 键、KEY\_Right 键、KEY\_Home 键以及 KEY\_End 键以外的所有编辑用键(包括 KEY\_Home 至 KEY\_PgDn 及其与 Ctrl 键、左右 Shift 键的组合以及 Ctrl 键和字母键的组合,以上各键码的定义可以参看第 14 章“HANENV 系统的头文件”中的有关部分)用于退出本次编辑,并将本次编辑的结果输出;

当光标位于编辑区域左端时使用 KEY\_Left 键和 KEY\_Home 键,或者光标位于编辑区域右端时使用 KEY\_Right 键和 KEY\_End 键的效果和上述编辑键相同,亦可退出编辑状态。

另外,在退出编辑时上述函数对所用编辑键值做如下处理:如果本次编辑对被编辑数据做了修改,则在编辑键值上加上 1024,否则按原样返回。返回的编辑键值主要用于实现全屏数据编辑之用。

## 12.2 各种类型数据字段的编辑函数

在上节介绍的基本数据编辑函数 getxya 的基础上,可以构造一组用于各种不同类型的数据字段的编辑函数。在 HANENV 系统中除了基本数据编辑函数 getxya 之外,还有以下 9 个函数用于编辑各种类型的数据字段:

```

getxyb——用于编辑逻辑型数据;
getxyc——用于编辑字符型数据;
getxyd——用于编辑日期型数据;
getxyf——用于编辑双精度型数据;
getxyi——用于编辑整型数据;

```

getxyl——用于编辑长整型数据；  
 getxyp——用于编辑口令型数据；  
 getxys——用于编辑字符串型数据；  
 getxyt——用于编辑字符串型数据，编辑区域为矩形；

其中前六个函数都是建立在基本数据编辑函数 getxya 的基础之上的，下面我们以型数据编辑函数 getxyi 为例介绍这类函数的设计方法。

### 程序 12-2 编辑整型数据。

整形数据编辑函数 getxyi 的程序结构比较简单，首先将所要编辑的整型数据值转换为字符串，并为其配备好编辑格式，然后调用基本数据编辑函数 getxya。结束编辑后，结果再转换回整数类型。最后还要作数据值范围检验，如果编辑后的数据没有在指定的范围内，则回过头去重新编辑。

```

/* -----
   函数 getxyi : 编辑整型数据
   ----- */
#include <hanenv.h>
#include <string.h>
#include <limits.h>

unsigned _Cdecl getxyi(col,line,width,value,limit1,limit2)
int col;           /* 编辑窗口左上角列坐标(以字节为单位) */
int line;          /* 编辑窗口左上角行坐标(以像素为单位) */
int width;         /* 编辑窗口宽度 */
int *value;        /* 指向待编辑的整型数据的指针 */
int limit1;        /* 待编辑数据的上限 */
int limit2;        /* 待编辑数据的下限 */
{
    char s[21];     /* 编辑缓冲区 */
    char p[21];     /* 格式缓冲区 */
    unsigned h;     /* 编辑键 */

    /* -- 检验数据值上、下限的合法性 ----- */
    if(limit1 >= limit2)
        return KEY_ESC;
    if(width>20)
        width = 20;

    /* -- 构造格式缓冲区 ----- */
    memset(p,'9',width);
    *(p+width) = 0;

```

```

/* -- 对非法输入值循环 ----- */
while(1)
{
    /* ----- 如果数值溢出用星号表示 ----- */
    if(*value == INT_MAX)
    {
        memset(s, '*', width);
        s[width] = 0;
    }
    /* ----- 数据转换为字符串 ----- */
    else
        sprintf(s, "%d", *value);

    /* ----- 调用通用数据编辑函数 ----- */
    h = getxya(col, line, width, s, p);

    /* ----- 如果作废编辑或编辑内容不变 ----- */
    if(h == KEY_ESC || h == RIGHT_BUTTON)
        break;

    /* ----- 字符串转换回整型数据 ----- */
    *value = atoi(s);
    if(*value < limit1 || *value > limit2)
    {
        sound(900);
        delay(100);
        nosound();
    }
    /* -- 构造输出结果 ----- */
    else
    {
        int havemouse = ismouselight();
        sprintf(s, "%d", *value);
        _TextCol = col;
        if(havemouse)
            delight_mouse();
        putnstr(s, width);
        if(havemouse)
            light_mouse();
        break;
    }
}
/* ----- 返回退出编辑状态的键盘码 ----- */

```

```

    return h;
}

```

逻辑型数据字段、字符型数据字段、日期型数据字段、双精度型数据字段和长整型数据字段的编辑函数 `getxyb`、`getxyc`、`getxyd`、`getxyf` 和 `getxyl` 的工作原理与此类似，有兴趣的读者可以参看本书所附软盘中对应函数的源程序。

口令型数据字段编辑函数 `getxyp` 是专门为输入口令或密码而设计的，其特点是在屏幕上的编辑区域中显示的并不是实际的口令或者密码，而是一串用来表示编辑位置的星号。我们在设计字符串型数据字段编辑函数 `getxys` 和 `getxyt` 时重点考虑了如何解决由两字节的汉字输入带来的调整编辑指针和移动光标等问题。这两个函数的区别在于如何解决较长字符串型数据字段的编辑。函数 `getxys` 允许编辑长度超过编辑区域的字符串型字段，此时编辑区域就象一个窗口，被编辑的字符串可在其中横向移动。而函数 `getxyt` 可以设置一个矩形编辑区域，编辑工作在此矩形区域之中进行，因而容量较大。注意这时编辑键 `KEY_Up` 和 `KEY_Down` 的定义与编辑键 `KEY_Left`、`KEY_Right` 的用法相似，只有在光标位于编辑区域的第一行时使用 `KEY_Up` 键或者当光标位于编辑区域的最下一行时使用了 `KEY_Down` 键才会退出编辑状态，否则使用这两个编辑键可以控制光标在编辑区域中上下移动。

口令型数据字段编辑函数和两种字符串型数据字段的编辑函数的程序设计方法与基本数据编辑函数类似。为了输入汉字字符串，在字符串型数据字段的编辑函数 `getxys` 和 `getxyt` 中使用汉字键盘输入函数 `gethan` 替换了函数 `geth`。

### 12.3 设计一个全屏幕数据录入、编辑界面程序

下面我们介绍如何使用本章前面介绍的各种数据字段编辑函数编写实用的数据录入编辑版面。

首先进行功能设计。我们设想数据录入编辑版面应该具有以下功能：

1. 整个录入编辑版面应该采用弹出式窗口结构，编辑结束后自动恢复原来的屏幕内容；
2. 编辑版面上的标题最好使用较大字体；
3. 允许“后悔”，即在使用作废键 `ESC` 或鼠标右键退出编辑后被编辑的记录应恢复原状；
4. 便于操作，对于有限离散数据最好采用代码表输入。每个数据域的编辑结束后自动进入下一个数据域的编辑，可以使用编辑键切换当前编辑数据域；
5. 版面安排合理，颜色醒目和谐。

由于具体应用项目中要进行录入编辑的数据的类型和格式千变万化，所以很难编写出完全通用的全屏幕数据录入编辑版面程序来。实际上，设计全屏幕数据录入版面程序正是编写数据库应用类实用程序的主要工作之一（另一项主要工作是设计报表打印程序）。下面我们通过设计一个人事管理数据的全屏幕录入编辑版面程序来说明这类程序的一般结构。

#### 程序 12-3 全屏幕人事档案数据编辑界面。

根据前面所作的功能设计，我们确定本程序（函数）的基本结构如下：

```

unsigned edit_employee(int col,int line,EMPLOYEE_TYPE *emp)

```

```
{
}
```

其中参数 col 和 line 用于确定弹出式编辑窗口在屏幕上的位置。编辑窗口的大小和被编辑的数据字段的数量有直接联系,一般在设计版面时就已经确定。本例中的编辑窗口的大小为 480×360,其版面设计如图 12-1 所示。为了方便应用程序的总装和联调,我们没有在程序中固定该编辑版面中所用各项的颜色,而是使用了 HANENV 系统的一组全局变量。这样,如果在总装联调时发现该录入编辑版面窗口的颜色配置与整个系统的色调不协调时,只需修改这些全局变量的值就可以了,不必回过头来再修改程序。我们使用的全局变量有:

```
_TextColor    —— 编辑窗口中正文颜色;
_Background   —— 窗口背景;
_EditColor    —— 当前编辑字段中文字的颜色;
_EditBk       —— 当前编辑字段背景色;
_TitleColor   —— 编辑窗口的标题文字显示颜色;
_TitleBk      —— 编辑窗口的标题背景色。
```

以及控制显示窗口颜色的 \_BarColor 和 \_BoxLineColor 等。

这个程序较长,下面我们分段进行介绍:

```
/* -----
   函数 edit_employee : 全屏幕人事档案数据编辑界面
   ----- */
#include <hanenv.h>
#include <string.h>
```

由于被编辑的数据字段比较多,所以我们设计了一个结构类型 EMPLOYEE\_TYPE,其分量为有关职工的各种属性:

```
/* -- 定义一个职工类型 ----- */
typedef struct
{
    int dept_no;           /* 部门编号 */
    int no;                /* 职工编号 */
    char name[9];          /* 姓 名 */
    int sex;               /* 性 别 */
    struct date birth_date; /* 出生日期 */
    int education;         /* 文化程度 */
    int position;          /* 职 务 */
    double salary;         /* 工 资 */
    char address[41];      /* 家庭住址 */
    char phoneno[15];      /* 电话号码 */
    char resume[201];      /* 个人简历 */
    char note[21];         /* 备 注 */
}EMPLOYEE_TYPE;
```

从前面给出的函数基本结构可以看出,我们采用一个指向 EMPLOYEE \_TYPE 类型的指针变量作为该函数的参数,这样比将每个需要编辑的数据字段都列在参数表中要简洁多了。

通过对数据字段的分析研究,我们发现字段“部门编号”、“性别”、“文化程度”和“职务”等4个字段的取值比较特殊,应该对其进行编码。事实上,在上面的 EMPLOYEE \_

TYPE 类型中,我们已经将这几个字段设计为整型,就是准备使用编码表示这几个字段的值。使用编码表示的字段最适合使用编码表函数编辑(参看 11.4),因此下面我们分别为这4个字段定义了相应的代码表:

图 12-1 职工档案录入编辑版面

```

/* —— 定义本公司的部门 ————— */
char * _Department[] =
{
    " 经理室",
    " 办公室",
    " 销售部",
    " 技术部",
    " 生产部",
    " 财务部",
    " 一分厂",
    " 二分厂"
};

/* —— 定义性别 ————— */
char * _Sex[] =
{
    " 男",
    " 女"
};

/* —— 定义文化程度 ————— */
char * _Education[] =
{
    " 博 士",

```

```

    " 硕 士",
    " 本 科",
    " 专 科",
    " 其 他"
};

/* ----- 定义本公司的职务种类 ----- */
char * _Position[] =
{
    " 总 经 理",
    " 办公室主任",
    " 部 门 经 理",
    " 总 工 程 师",
    " 工 程 师",
    " 办 事 员",
    " 秘 书",
    " 工 人",
    " 其 他"
};

```

函数 edit\_employee 的处理流程为

```

保存屏幕背景;
保存职工档案原值;
显示职工档案编辑版面;
显示职工档案的原值;
设置当前字段号;
循环：编辑职工档案
{
    根据当前字段号分别调用各种类型的数据编辑函数进行编辑;
    根据数据编辑函数的返回值决定下一个要编辑的字段或跳出循环;
}
如果按作废键则恢复职工档案原来的内容;
恢复编辑版面的背景;
返回退出编辑键。

```

```

/* ----- 函数 edit_employee：编辑职工档案 ----- */
unsigned edit_employee(col,line,emp)
int col;                /* 编辑版面在屏幕上的列坐标(单位为字节) */
int line;               /* 编辑版面在屏幕上的行坐标(单位为象素) */
EMPLOYEE_TYPE *emp;    /* 指向职工档案的指针 */
{
    EMPLOYEE_TYPE old_emp;

```

```

unsigned bak_handle;
unsigned key;
int field;
int modified = NO;
char tmpline[81];
int len;
int old_wl,old_wt,old_wr,old_wb;
/* ----- 保存背景 ----- */
delight_mouse();
delightcursor();
bak_handle = getblockXMS(col,line,col+59,line+359);

/* ----- 保存职工档案原值 ----- */
memcpy(&old_emp,emp,sizeof(EMPLOYEE_TYPE));

/* ----- 显示职工档案编辑版面 ----- */
_Box(col*8,line,480,360);
_Bar(col*8+4,line+4,472,38,_TitleBk,0xffff0000);
_Xtimes = 2;
_Ytimes = 1;
outxystr(col+14,line+16,_TitleColor,"输入编辑人事档案");
_Xtimes = 1;
outxys(col+4,line+70,_TextColor,"部 门:");
outxys(col+4,line+100,_TextColor,"职工编号:");
outxys(col+4,line+130,_TextColor,"姓 名:");
outxys(col+4,line+160,_TextColor,"性 别:");
outxys(col+4,line+190,_TextColor,"出生日期:");
outxys(col+30,line+70,_TextColor,"文化程度:");
outxys(col+30,line+100,_TextColor,"职 务:");
outxys(col+30,line+130,_TextColor,"工 资:");
outxys(col+30,line+160,_TextColor,"家庭住址:");
outxys(col+30,line+190,_TextColor,"电话号码:");
outxys(col+4,line+220,_TextColor,"个人简历:");
outxys(col+4,line+318,_TextColor,"备 注:");

/* ----- 显示职工档案的原值 ----- */
outxys(col-15,line+70,_TextColor,_Department[emp->dept_no]);
sprintf(tmpline,"%04d",emp->no);
outxys(col+15,line+100,_TextColor,tmpline);
outxys(col+15,line+130,_TextColor,emp->name);
outxys(col+15,line+160,_TextColor,_Sex[emp->sex]);
sprintf(tmpline,"%04d.%02d.%02d",emp->birth_date.da_year,
    emp->birth_date.da_mon,emp->birth_date.da_day);

```



```

outxys(col+15, line + 190, _TextColor, tmpline);
outxys(col+41, line + 70, _TextColor, _Education[emp->education]);
outxys(col+41, line + 100, _TextColor, _Position[emp->position]);
sprintf(tmpline, "%10.2f", emp->salary);
outxys(col+41, line + 130, _TextColor, tmpline);
memcpy(tmpline, emp->address, 18);
tmpline[18] = 0;
outxys(col+41, line + 160, _TextColor, tmpline);
outxys(col+41, line + 190, _TextColor, emp->phoneno);
draw_rectangle((col+14)*8, line+216, 42*8, 5*18+8, _TextColor, 0xff);
get_window(old_wl, old_wt, old_wr, old_wb);
set_window(col+15, line+220, col+54, line+220+5*18);
movecursor(col+15, line+220);
putnstr(emp->resume, 200);
set_window(old_wl, old_wt, old_wr, old_wb);
outxys(col+15, line + 318, _TextColor, emp->note);
light_mouse();

/* ----- 编辑职工档案 ----- */
field = 0;
do
{
    switch(field)
    {
        case 0: /* 编辑部门字段 */
            key = getcode(col+15, line+70, 10, 10, "部 门", 8, _Department,
                          &(emp->dept_no), NO);
            putxys(col+15, line + 70, _TextColor, _Background,
                  _Department[emp->dept_no]);
            break;
        case 1: /* 编辑职工编号字段 */
            key = getxyi(col+15, line+100, 4, &(emp->no), 0, 9999);
            break;
        case 2: /* 编辑姓名字段 */
            len = 8;
            key = getxys(col+15, line+130, 8, emp->name, &len);
            break;
        case 3: /* 编辑性别字段 */
            key = getcode(col+15, line+160, 10, 4, "性别", 2, _Sex, &(emp->sex),
                          NO);
            putxys(col+15, line + 160, _TextColor, _Background, _Sex[emp->sex]);
            break;
        case 4: /* 编辑出生日期字段 */

```

```

    key = getxyd(col+15,line+190,&(emp->birth_date));
    break;
case 5:                                /* 编辑文化程度字段 */
    key = getcode(col+41,line+70,10,7,"文化程度",5,_Education,
                  &(emp->education),NO);
    putxys(col+41,line + 70,_TextColor,_Background,
            _Education[emp->education]);
    break;
case 6:                                /* 编辑职务字段      */
    key = getcode(col+41,line+100,14,11,"职    务",9,_Position,
                  &(emp->position),NO);
    putxys(col+41,line + 100,_TextColor,_Background,
            _Position[emp->position]);
    break;
case 7:                                /* 编辑工资字段      */
    key = getxyf(col+41,line+130,10,&(emp->salary),2,00.0,10000.0);
    break;
case 8:                                /* 编辑家庭住址字段 */
    len = 40;
    key = getxys(col+41,line+160,18,emp->address,&len);
    break;
case 9:                                /* 编辑电话号码字段 */
    len = 14;
    key = getxys(col+41,line+190,14,emp->phoneno,&len);
    break;
case 10:                               /* 编辑个人简历字段 */
    key = getxyt(col+15,line+220,40,5,emp->resume);
    break;
case 11:                               /* 编辑备注字段      */
    len = 20;
    key = getxys(col+15,line+318,20,emp->note,&len);
    break;
}
/* ----- 在编辑过程中是否做了修改 ----- */
if(key>1024)
{
    modified = YES;
    key -= 1024;
}
/* ----- 处理各编辑函数的返回值 ----- */
switch(key)
{
    /* ----- 编辑下一个字段 ----- */

```

```

    case KEY_ENTER:
    case KEY_Right:
    case KEY_Down:
        if(field! = 11)
            field++;
        break;
    /* ----- 编辑上一个字段 ----- */
    case KEY_Left:
    case KEY_Up:
        if(field == 0)
            field = 11;
        else
            field--;
        break;
    /* ----- 编辑第一个字段 ----- */
    case KEY_Home:
        field = 0;
        break;
    /* ----- 编辑最后一个字段 ----- */
    case KEY_End:
        field = 11;
        break;
}
}while(key! = KEY_Ctr_W
    && key! = KEY_ESC && key! = RIGHT_BUTTON);

/* --- 如果按作废键则恢复职工档案原来的内容 ----- */
if((key == KEY_ESC || key == RIGHT_BUTTON) && modified)
    memcpy(emp, &old_emp, sizeof(EMPLOYEE_TYPE));

/* --- 恢复编辑版面的背景 ----- */
if(bak_handle)
{
    delight_mouse();
    putblockXMS(col, line, col + 59, line + 359, bak_handle);
    light_mouse();
}
/* ----- 返回退出编辑键 ----- */
if(key == KEY_Ctr_W && modified)
    key += 1024;
return key;
}

```

在阅读和使用函数 `edit_employee` 时请注意以下几个细节:

1. 为了节约使用对于数据库应用程序来说相对比较紧张的常规内存, 我们采用了函数 `getblockXMS` 和 `putblockXMS` 保存和恢复编辑窗口的背景, 此时编辑窗口的背景存储在扩充内存中。这是因为数据记录的全屏幕编辑版面的尺寸一般比较大, 如果仍然采用函数 `getblock` 和 `putblock` 保存和恢复屏幕背景的话会占用大量的常规内存, 例如存储占屏幕四分之一大小的编辑窗口的背景就需要 64K 的存储容量。但应注意的是使用函数 `getblockXMS` 和 `putblockXMS` 需要 `HIMEM.SYS` 的支持, 在主程序中还要加入对 XMS 的初始化语句。
2. 在保存和恢复屏幕以及向屏幕上输入内容时应保证鼠标光标和文本光标均处于关闭状态, 否则影响屏幕效果。
3. 该函数的返回值共有 4 种: `KEY_ESC` 和 `RIGHT_BUTTON` (鼠标右键) 表示作废此次编辑工作, 数据记录仍然保持原来的内容; 而组合键 `KEY_Ctr_W` 表示此次编辑并没有改变数据记录的内容; 只有 `KEY_Ctr_W+1024` 表示被编辑的数据记录的内容已经被修改。
4. 为了简化程序设计, 程序中没有考虑鼠标左键的应用。实际上, 鼠标左键可以设计为快速字段切换键: 用鼠标左键点击某个字段的编辑区域则该字段立刻变为当前编辑字段; 如果用鼠标左键点击编辑窗口以外的区域则可退出编辑模块, 并返回鼠标左键值 `LEFT_BUTTON` 等。
5. 在调用函数 `edit_employee` 之前应仔细设计编辑界面中所使用的各种颜色, 并通过改变相应的全局变量进行设置。

# 菜单程序设计

菜单是应用程序最基本的功能部件之一,主要用于选择切换应用程序的功能模块。菜单程序设计的好坏直接影响着应用软件的人机界面质量。大体上说,目前应用程序中主要使用以下两种类型的菜单:一种是以字符显示和键盘输入为基础的弹出式菜单,一种是以图形显示和鼠标操作为基础的按键式菜单。弹出式菜单出现较早,发展出了许多变种,例如著名的光条式和下拉式菜单。而按键式菜单是和图形用户界面一起出现的,目前在 WINDOWS 风格的应用程序中非常流行。

无论菜单结构如何变化,其发展的基本思路无非两条:简化操作和美化界面。从操作方面来看,只使用方向小键盘和回车键的光条式菜单就比使用数字或字母代码的选择式菜单好用,而使用鼠标操作按键式菜单又比使用键盘操作光条式菜单来得简便。

在实际应用中,除了上面提到的两类菜单结构被广为使用以外,还有一些为比较特殊的应用场合设计的菜单结构,例如图标式菜单,适当选用可以提高应用程序界面的表现能力。

漂亮实用的菜单程序还是显示程序员编程实力的传统习题。在本章中我们准备通过两个菜单函数介绍这类程序的一般设计方法。

### 13.1 多功能按键式菜单及其应用

在本节中我们介绍一个多功能按键式菜单函数的设计。该函数用途广泛,既可以实现按键式菜单,也可以模拟光条式菜单或者组合成下拉式菜单。该函数既可以使用鼠标控制,也可以使用键盘控制。通过适当的设置,还可以控制整个菜单在屏幕上随意移动,以达到特殊的应用效果。

设计通用的菜单函数首先要解决的问题就是如何表示和存储菜单的各种参数。菜单的参数可以分为两类:一是关于菜单本身的参数,例如菜单的位置、大小等,二是关于菜单中条目(或按键)的参数,例如每个条目的名称、位置、对应的处理函数等。为了使我们设计的菜单函数能够适应尽可能多的应用场合,通过仔细研究流行的按键式菜单、光条式菜单和下拉式菜单的结构,我们确定了以下设计原则:

1. 参数设置要灵活,例如按键式菜单中的每个按键的位置、大小和颜色均可独立设置;
2. 功能要全面,例如单键锁定、菜单移动和鼠标、键盘操作任选;

3. 使用灵活方便,例如重复键定义、多种方式调用所选模块等。

因此我们设计了两个数据类型:一个是菜单按键结构类型,一个是按键式菜单结构类型,分别用于存放与按键式菜单条目和按键式菜单本身设置有关的各种数据。

菜单按键(菜单条目)的结构类型定义如下:

```
/* ----- 有关按键式菜单的按键类型的定义 ----- */
typedef struct          /* 按键类型定义 */
{
    int x;               /* 按键左上角相对于菜单的列坐标 */
    int y;               /* 按键左上角相对于菜单的行坐标 */
    int w;               /* 按键宽 */
    int h;               /* 按键高 */
                        /* 以上4个参数的单位均为像素 */
    unsigned butncolor;  /* 正常按键颜色 */
    unsigned pushcolor; /* 被选中的按键颜色 */
                        /* 以上两个参数均可用按钮颜色宏 */
    unsigned fonttimes; /* 按键名的放大倍数(第0字节为横
                        向放大倍数,第1字节为纵向放大
                        倍数,如不放大则此参数填257) */
    char * name;         /* 按键名 */
    unsigned key;        /* 按下该按键时菜单函数的返回值 */
    unsigned pressed;    /* 与该菜单按键同功能的键盘键码 */
    int high;            /* 按键厚度(单位为像素,可为0) */
    int lock;            /* 按键是否被锁住 */
    unsigned (*fun)();   /* 按下按键时执行的功能模块指针 */
}BUTTON_TYPE;
```

下面我们简单说明其中各参数的功能和填写要求:

1. 按键位置(x和y)以及按键尺寸(w、h和high): 其中按键位置是相对于菜单位置的数据。在填写时应注意这两者的单位不同,按键的列坐标的单位为像素,菜单列坐标的单位为字节。另外,菜单按键尺寸中的按键厚度high决定了菜单的类型。如果菜单按键的厚度不等于零,则菜单条目是一个由按键的宽、高和厚度决定的立体型按键,菜单类型就是按键式;如果按键厚度等于零,则直接显示菜单条目,菜单类型即为光条式。

2. 菜单按键颜色配置(butncolor和pushcolor): 这两个分量分别用于表示“正常”的菜单按键和“当前”菜单按键的颜色搭配,后者同时还用于显示被选中的菜单按键在按下状态下的颜色搭配。以立体型菜单按键为例,每个菜单按键需要使用4种颜色:键名(菜单条目)、按键键体、阳边(上边缘和左边缘)以及阴边(下边缘和右边缘)。在使用键盘控制菜单时,还需要区别当前菜单按键(在光条式菜单中使用光条表示)和其他菜单按键,因此每定义一个菜单按键就需要两套共8种颜色。如果每个菜单按键都使用8个整型变量存放这8种颜色就未免过于浪费了。因此我们将确定一个菜单按键的4种颜色压缩存放在一个unsigned类型的变量中:第0—3位存放键名颜色,第4—7位存放键身颜色,第8—11位存放阳边颜色,第

12—15 位存放阴边颜色,如图 13-1 所示。参数 butncolor 和 pushcolor 都是这种类型的变量。

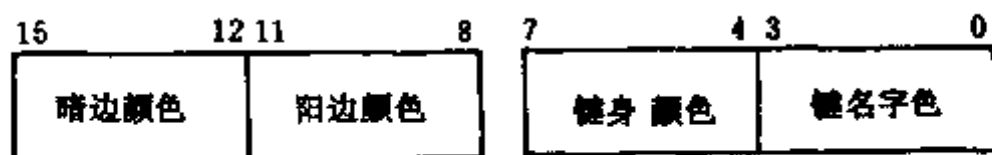


图 13-1 按键颜色存储方式

当然,这会给设置菜单按键的参数带来不便。因此我们在头文件 hanenv.h 中定义了一组宏,用于说明常用的菜单按键颜色搭配:

BLUE_BAR	—— 蓝底白字键,亮边为亮蓝色,暗边为深灰色;
GREEN_BAR	—— 绿底白字键,亮边为亮绿色,暗边为深灰色;
CYAN_BAR	—— 青底白字键,亮边为亮青色,暗边为深灰色;
RED_BAR	—— 红底白字键,亮边为亮红色,暗边为深灰色;
MAGENTA_BAR	—— 洋红底白字键,亮边为亮洋红色,暗边为深灰色;
BROWN_BAR	—— 棕底白字键,亮边为黄色,暗边为深灰色;
GRAY_BAR	—— 灰底黑字键,亮边为白色,暗边为深灰色;

在应用程序中通常选用灰色菜单按键或青色菜单按键,如果使用键盘控制菜单,则可以选择红色键作为当前菜单按键的标志。

3. 键名(name)和键名字符放大倍数(fonttimes):键名即菜单条目。键名字符放大倍数用于设置键名中字符或汉字的放大倍数,也是一个 unsigned 类型的变量,其中第 0 个字节存放键名字符的横向放大倍数,第 1 个字节存放纵向放大倍数,如图 13-2 所示。对于一般键名无需放大显示的菜单按键来说,此值应该填写为 257(即横向放大倍数和纵向放大倍数均为 1)。

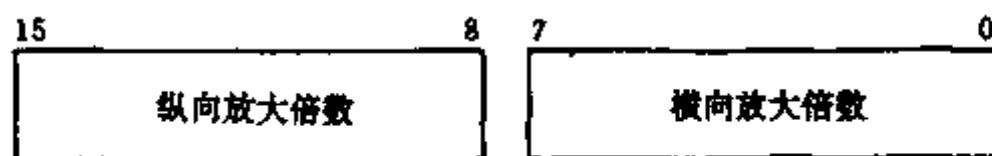


图 13-2 键名放大倍数存储方式

4. 同名键盘码(press)、返回值(key)和函数指针(fun):同名键盘码即与该菜单按键对应的键盘键码。如果允许使用键盘操纵菜单,则按下键盘上的同名键就相当于使用鼠标左键点选该菜单按键。返回值和函数指针都用于执行该菜单按键所对应的功能模块。如果函

数指针不等于空,而是指向某个处理函数,则选择该菜单按键之后就会执行该函数,并将该函数的返回值作为菜单的返回值返回;如果函数指针等于空,则菜单函数返回上述返回值(key)。该函数的编写方法为

```
unsigned process(void)
{
    .....
}
```

即该函数执行完后可以返回一个统一键盘码,该键码会作为菜单函数的返回值继续返回。特别要注意的是如果选用返回值(key)来表示菜单功能的话,一定要置函数指针为空,否则程序无法正常运行。

5. 禁用标志(lock):用于“锁住”该按键。被锁住的按键不响应鼠标或键盘,且键名使用虚体字显示。

按键式菜单的结构类型定义如下:

```
/* ----- 有关按键式菜单的结构类型定义 ----- */
typedef struct          /* 按键式菜单类型 */
{
    int col;             /* 菜单左上角列坐标(单位为字节) */
    int line;            /* 菜单左上角行坐标(单位为像素) */
    int width;           /* 菜单宽度(单位为字节) */
    int high;            /* 菜单高度(单位为像素) */
    int arrow_ctrl;      /* 可否用键盘控制菜单 */
    int fixed;           /* 菜单是否固定 */
    int left;            /* 菜单移动范围:左边界(字节) */
    int top;             /* 菜单移动范围:上边界(像素) */
    int right;           /* 菜单移动范围:右边界(字节) */
    int bottom;          /* 菜单移动范围:下边界(像素) */
    unsigned quit_key1;  /* 退出菜单所用键 */
    unsigned quit_key2;  /* 退出菜单所用键 */
    unsigned press_key1; /* 选择菜单项目所用键 */
    unsigned press_key2; /* 选择菜单项目所用键 */
    unsigned forward1;   /* 向前移动当前菜单条目所用键 */
    unsigned forward2;   /* 向前移动当前菜单条目所用键 */
    unsigned backward1;  /* 向后移动当前菜单条目所用键 */
    unsigned backward2;  /* 向后移动当前菜单条目所用键 */
    BUTTON_TYPE * buttons; /* 指向按键表的指针 */
    int button_count;    /* 按键个数 */
    int pop_up;          /* 菜单是否已经弹出 */
    int current;         /* 当前菜单条目的序号 */
    int saveimage;       /* 是否存储菜单背景 */
}
```



```
char **block;          /* 菜单背景存储指针          */  
void (*disp_fun)();     /* 显示菜单轮廓图样函数(可为空) */  
}BUTTON_MENU;
```

下面我们简单说明其中各个参数的用途和填写要求:

1. 菜单的位置(col 和 line)和大小(width 和 high): 菜单的位置同时也决定了菜单中各个菜单按键的实际位置(参看前述菜单按键结构类型定义中关于按键位置参数的说明)。另外,如果菜单是可移动的(由分量 fixed 决定),或者菜单是弹出式的,一旦退出后要恢复背景(由分量 saveimage 决定),那么菜单的大小就很重要,必须按照实际情况填写。

2. 可否使用键盘控制菜单(arrow\_ctrl): 通常操作菜单都使用鼠标。使用鼠标操作菜单的方法非常简单: 只需用鼠标器控制鼠标光标移到所需要的菜单按键上并按下鼠标左键即可完成选择菜单条目的工作。如果需要使用键盘操作菜单,则应置本参数的值为 YES。使用键盘操作菜单有两种方法: 一种是直接按下与本菜单条目所对应的键盘按键(即前述菜单按键结构类型定义中的参数 pressed 之值)。另一种是使用移动当前菜单条目键(包括向前移动当前菜单条目键 forward1、forward2 和向后移动当前菜单条目键 backward1、backward2, 见后)重新选择当前菜单按键(当前菜单条目对应的按键的颜色为 BUTTON\_TYPE 中的颜色参数 pushcolor, 而其他菜单按键的颜色为 butncolor), 并使用选择菜单条目键(press\_key1 或 press\_key2, 见后)选择与当前菜单条目对应的功能。应该说明的是即使允许使用键盘控制菜单,仍然可以直接使用鼠标操纵菜单。

3. 菜单是否可移动(fixed)及其移动范围(left、top、right 和 bottom): 如果置分量 fixed 的值为 NO,则可制作一个类似“遥控器”的按键式应用程序,此时可以利用鼠标“拖动”整个菜单在屏幕上由分量 left、top、right 和 bottom 所确定的矩形区域内随意移动。关于这方面的程序设计例子可以参看计算器函数 calculcor 的源程序(见随书所配 1# 软盘),其屏幕效果可以通过运行工具软件 hantool 中的计算器选项来观察。

4. 菜单操作键: 如果允许使用键盘控制菜单(参数 arrow\_ctrl 被设置为 YES),则需要设置控制菜单所用的各种键盘码。这类键盘码一共 4 组,每组两个: quit\_key1 和 quit\_key2 用于退出菜单函数; press\_key1 和 press\_key2 用于选择当前菜单按键所对应的程序模块; 而 forward1、forward2、backward1 和 backward2 用于重新选择当前菜单条目。通常,退出菜单可以使用 KEY\_ESC 键、KEY\_Ctr\_Q 键和鼠标右键,如果菜单是下拉式菜单的二级菜单则可以选择 KEY\_Right 和 KEY\_Left; 选择当前菜单条目可以使用回车键和空格键,而重新选择当前菜单条目可以使用 KEY\_Up 键、KEY\_Down 键、KEY\_Left 键和 KEY\_Right 键等方向控制键。

5. 菜单按键表指针(buttons)和菜单按键个数(button\_count): 这两个参数确定了菜单中所用到的菜单按键结构数组及其元素个数。

6. 菜单是否弹出(pop\_up): 这个参数实际上是一个内部变量,记录了菜单是否已在屏幕上显示。在初始化菜单参数时该项应设置为 NO。

7. 当前菜单条目序号(current): 如果允许使用键盘控制菜单,该参数指出首次调用菜单函数时的当前菜单条目的序号。

8. 是否存储菜单背景(saveimage)和菜单背景存储指针(block): 如果设置是否存储菜

单背景项为 YES, 则菜单函数将菜单显示处原来的背景存放在参数 block 处, 当使用退出菜单键退出菜单函数时就会恢复菜单显示处原来的背景。

9. 显示菜单轮廓图样函数指针 (disp\_fun): 该指针用于指向一个应用程序开发人员自己编写的菜单背景显示函数。如果设置菜单为可移动式 (参数 fix 的值为 NO), 则该菜单显示函数所显示的菜单图象也可以随之移动。在编写该函数时应该注意其显示范围不能超过菜单的位置和大小。该函数的编写方法为

```
void display_menu(BUTTON_MENU * menu)
{
    .....
}
```

在详细定义了菜单及其条目的有关参数之后, 就可以开始设计我们的菜单处理函数了。我们设计的菜单处理函数的使用方法为

```
/* -----
   按键菜单函数的使用方法
   ----- */
BUTTON_MENU menu;          /* 定义一个菜单参数变量 */
unsigned key;               /* 接收菜单函数的返回值 */
/* -- 用于设置菜单及其条目的各项参数的程序段 ----- */
.....

/* -- 第一次调用菜单函数, 用于显示菜单及其按键 ----- */
button_menu(&m, 0);

/* -- 循环处理对于菜单函数的调用 ----- */
do
{
    /* -- 读取控制菜单的键盘或鼠标按键信息 ----- */
    key = geth();

    /* -- 调用菜单函数对按键信息进行处理 ----- */
    key = button_menu(&menu, key);

    /* -- 如果使用返回值控制个功能模块的调用 ----- */
    switch(key)
    {
        case (键值 1)
            调用功能模块 1 的处理函数;
            break;
        case (键值 2)
```

```

        调用功能模块 2 的处理函数;
        break;
        .....
    }
    /* --- 如果使用条目中定义的处理函数,以上一段可不用 --- */
} while (key != menu.quit_key1 && key != menu.quit_key2);

```

由以上程序结构中可以看出按键式菜单函数的一般调用方法。值得强调的是如果在菜单条目的参数表中已经定义了函数指针 fun 指向某个功能模块的处理函数,那么对于该函数模块的处理就由菜单函数直接调用,因此就不需要上面程序段中的开关语句了。如果已经将菜单条目的参数表中的函数指针置为空,那么在上述程序段中的菜单函数就会返回对应菜单条目中定义的返回值 key,此时就应该使用开关语句对此返回值分情形进行处理。

多功能按键式菜单函数的调用格式为

```
unsigned button_menu(BUTTON_MENU * menu, unsigned key);
```

其参数 key 是驱动该菜单处理的统一键盘码,可以由函数 geth、gethan 以及全屏幕数据编辑函数族(getxyi 等)提供。在上述程序结构中菜单函数的调用共有两处:第一次调用是为了在屏幕上显示菜单,以后各次调用应由一个循环控制。

### 程序 13-1 多功能按钮式菜单。

该函数是 HANENV 系统中结构最复杂的函数之一。大体上说来,该函数的处理可以分为对鼠标按钮的处理和对键盘按键的处理两大部分。该函数的工作流程为

```

如果是第一次调用本函数(菜单尚未弹出),则
    设置弹出标记,并在屏幕上画出菜单;
否则(即菜单已经弹出,则处理键盘码)
{
    如果 key 是鼠标左键且鼠标光标在菜单内,则
    {
        如果鼠标光标指向菜单中的某个按键,则
        {
            按下该菜单按键;
            如果该菜单条目的定义中有可执行函数,则
                执行该函数,取出该函数的返回码;
            否则取出该菜单条目对应的返回码;
            弹起该菜单按键;
            退出菜单处理,返回上述返回码;
        }
        如果菜单可移动,且鼠标未指向任何菜单按键,则
        {
            保存菜单的屏幕图象;
            用鼠标控制一个虚线框选定菜单的新位置;
        }
    }
}

```

```

        抹去原来的菜单;
        保存新菜单位置上的屏幕背景;
        在新位置上恢复菜单的图象;
    }
}
如果菜单可用键盘控制,且 key 是键盘按键,则
{
    如果 key 是选择键,则
    {
        按下当前菜单按键;
        如果该菜单条目的定义中有可执行函数,则
            执行该函数,取出该函数的返回码;
        否则取出该菜单条目对应的返回码;
        弹起该菜单按键;
        退出菜单处理,返回上述返回码;
    }
    如果是重新选择当前菜单条目键,则
        移动当前菜单按键;
    如果 key 和某菜单条目的对应键盘码相同,则、
        将与 key 对应的菜单条目变为当前菜单条目;
}
}

/* -----
   程序模块 button_menu.c : 多功能按钮式菜单
   ----- */

#include <hanenv.h>
#include <fcntl.h>
#include <alloc.h>

/* --- 内部函数 _DrawButtonMenu : 在指定坐标处显示菜单 --- */
static void _Cdecl _DrawButtonMenu(b)
BUTTON_MENU * b;          /* 菜单参数          */
{
    int i;

    /* ----- 显示菜单 ----- */
    delight_mouse();
    if(b->saveimage)
        b->block = getblock(b->col,b->line,b->width,b->height);
    if(b->disp_fun)
        (* (b->disp_fun))(b);
    for(i=0;i<b->button_count;i++)

```



```

    if(b->buttons[i].key==b->quit_key1
        || b->buttons[i].key==b->quit_key2)
    {
        b->pop_up = NO;
        delight_mouse();
        if(b->saveimage && b->block)
            putblock(b->col,b->line,b->width,b->high,b->block);
        light_mouse();
        return b->quit_key1;
    }
    /* -- 如果选中项有可执行函数,执行之 ----- */
    if(b->buttons[i].fun)
        key = (* (b->buttons[i].fun))();

    /* -- 否则取出该选择项对应的键盘码 ----- */
    else
        key = b->buttons[i].key;

    /* ----- 弹起该按键 ----- */
    delight_mouse();
    _DrawButton(b->buttons[i],b->col,b->line,b->arrow_ctrl,0);
    light_mouse();

    /* ----- 退出菜单处理 ----- */
    return key;
}

/* -- 如果菜单可移动,且鼠标未指向任何按键 ----- */
if(! b->fixed && coreleft())(long)(b->width)*b->high*4+16)
{
    int x;                /* 菜单的列坐标(单位为像素) */
    int y;                /* 菜单的行坐标(单位为像素) */
    int dx;               /* 移动菜单的列增量 */
    int dy;               /* 移动菜单的行增量 */
    int oldml,oldmt,oldmr,oldmb; /* 原鼠标范围 */
    unsigned mouse_button;

    /* ----- 计算菜单的位置 ----- */
    x = b->col*8;
    y = b->line;

    /* ----- 保存菜单的屏幕图象 ----- */
    delight_mouse();
    block = getblock(b->col,b->line,b->width,b->high);

```

```

light_mouse();
/* ----- 设置菜单的移动范围 ----- */
get_mouse_range(&oldml,&oldmt,&oldmr,&oldmb);
set_mouse_range(b->left,b->top,b->right-b->width*8+1,
                b->bottom-b->high+1);

/* -- 用鼠标控制一个虚线框选定菜单的新位置 -- */
do
{
    /* ----- 读取鼠标光标的位置 ----- */
    mouse_button = get_mouse_status(&dx,&dy);

    /* -- 计算矩形虚线框的移动位置 ----- */
    dy -= y;
    dx -= x;

    /* ----- 如果需要移动虚线框 ----- */
    if(dx || dy)
    {
        delight_mouse();

        /* ----- 消除旧框 ----- */
        _SetWriteMode(XOR_MODE);
        draw_rectangle(x,y,b->width*8,b->high,LIGHTGRAY,0xf0);

        /* ----- 计算新框位置 ----- */
        x += dx;
        y += dy;

        /* ----- 重画新框 ----- */
        draw_rectangle(x,y,b->width*8,b->high,LIGHTGRAY,0xf0);
        _SetWriteMode(PUT_MODE);
        light_mouse();
    }
}while(mouse_button == LEFT_BUTTON);

/* ----- 恢复原来的鼠标移动范围 ----- */
set_mouse_range(oldml,oldmt,oldmr,oldmb);

/* ----- 取消最后一次所画的框 ----- */
delight_mouse();
_SetWriteMode(XOR_MODE);
draw_rectangle(x,y,b->width*8,b->high,LIGHTGRAY,0xf0);

```

```

    _SetWriteMode(PUT _MODE);

    /* ---- 恢复菜单下的背景(抹去原来的菜单) ---- */
    putblock(b->col,b->line,b->width,b->high,b->block);

    /* ---- 修改菜单参数表中的菜单位置 ---- */
    b->col = x/8;
    b->line = y;

    /* ---- 保存新菜单位置上的屏幕背景 ---- */
    b->block = getblock(b->col,b->line,b->width,b->high);

    /* ---- 在新位置上恢复菜单的图象 ---- */
    putblock(b->col,b->line,b->width,b->high,b->block);
    light _mouse();
}

/* ---- 菜单移动处理结束 ---- */
}

/* -- 如果输入键盘码为菜单退出键,取消菜单显示,退出 -- */
else if(key==b->quit _key1 || key==b->quit _key2 || key==KEY _ESC)
{
    b->pop _up = NO;
    delight _mouse();
    if(b->saveimage && b->block)
        putblock(b->col,b->line,b->width,b->high,b->block);
    light _mouse();
    return key;
}

/* ---- 如果菜单可用键盘控制,处理键盘键 ---- */
else if(b->arrow _ctrl)
{
    /* -- 如果输入键盘码是选择键,执行当前选项 -- */
    if(key==b->press _key1 || key==b->press _key2)
    {
        /* ---- 绘出按下状态的按键 ---- */
        delight _mouse();
        _DrawButton(b->buttons[b->current],h->col,b->line,YES,1);
        light _mouse();

        /* ---- 如果选中项有可执行函数,执行之 ---- */
        if(b->buttons[b->current].fun)

```



```

        key = (* (b->buttons[b->current].fun))();
/* ----- 否则取出该选择项对应的键盘码 ----- */
else
{
    key = b->buttons[b->current].key;
    delay(_MouseSpeed);
}
/* ----- 绘出弹起状态的按键 ----- */
delight_mouse();
_DrawButton(b->buttons[b->current],b->col,b->line,YES,0);
light_mouse();
/* ----- 退出菜单处理 ----- */
return key;
}
/* -- 如果是移动选择项键,移动当前选择项 ----- */
else if(key==b->forward1 || key==b->forward2
        || key==b->backward1 || key==b->backward2)
{
/* -- 用正常色绘制当前键 ----- */
    delight_mouse();
    _DrawButton(b->buttons[b->current],b->col,b->line,NO,0);
    light_mouse();
/* ----- 处理向前移动键 ----- */
    if(key==b->forward1 || key==b->forward2)
    {
        b->current = (b->current==b->button_count-1? 0;b->current+1);
        if(b->buttons[b->current].lock)
            ungeth(b->forward1);
    }
/* ----- 处理向后移动键 ----- */
    else if(key==b->backward1 || key==b->backward2)
    {
        b->current = (b->current==0? b->button_count-1;b->current-1);
        if(b->buttons[b->current].lock)
            ungeth(b->backward1);
    }

/* ----- 以选中键色绘制当前键 ----- */
    delight_mouse();
    _DrawButton(b->buttons[b->current],b->col,b->line,YES,0);
    light_mouse();

```

```

        /* ----- 退出菜单处理 ----- */
        return key;
    }
    /* -- 检查输入键盘码是否和某选择项对应码相同 -- */
    for(i=0;i<b->button_count;i++)
        if(key==b->buttons[i].pressed)
        {
            delight_mouse();

            /* -- 将与输入码对应的键变为当前键 ----- */
            _DrawButton(b->buttons[b->current],b->col,b->line,NO,0);
            _DrawButton(b->buttons[i],b->col,b->line,YES,1);
            light_mouse();
            delay(_MouseSpeed);
            delight_mouse();
            _DrawButton(b->buttons[i],b->col,b->line,b->arrow_ctrl,0);
            light_mouse();
            b->current = i;

            /* ----- 退出菜单处理 ----- */
            return b->buttons[i].key;
        }
    }
    /* ----- 使用键盘控制菜单的处理结束 ----- */
}
/* -- 如果菜单尚未弹出,设置标记,在屏幕上画菜单 -- */
else
{
    b->pop_up = YES;
    _DrawButtonMenu(b);
}
/* ----- 返回键盘码 ----- */
return key;
}

```

一般说来,在设计应用程序的菜单部分时首先应该确定自己所需要的菜单类型:是使用一级的按键式菜单,还是使用多级的下拉式菜单;如果使用按键式菜单,是否允许使用键盘控制菜单,是否为可移动式菜单;其次才是菜单的条目内容和颜色等参数。在确定了菜单及其条目(菜单按键)的各项参数之后,应该专门编写一段程序设置这些参数。下面我们通过几个例子说明如何设置菜单的参数。

### 程序 13-2 HANENV 系统的字库工具软件 hantool 的主菜单。

该菜单是一个按键式菜单,允许使用键盘控制,不可移动,非弹出式(在 hantool 工作的整个期间均出现在屏幕上),共有 6 个按键(参看图 5-2)。菜单参数设置部分的源程序为

```
/* ----- 设置字库工具软件 hantool 的菜单参数 ----- */
menu.col          = 5;          /* 菜单的位置与大小      */
menu.line         = 370;
menu.width        = 560;
menu.high         = 100;
menu.fixed        = YES;        /* 菜单固定              */
menu.arrow_ctrl   = YES;        /* 允许使用键盘操作菜单  */
menu.pop_up       = NO;
menu.forward1     = KEY_Down;   /* 控制菜单的键盘按键    */
menu.forward2     = KEY_Right;
menu.backward1    = KEY_Up;
menu.backward2    = KEY_Left;
menu.quit_key1    = KEY_ESC;
menu.quit_key2    = KEY_Ctr_Q;
menu.press_key1   = KEY_ENTER;
menu.press_key2   = KEY_SPACE;
menu.disp_fun     = NULL;       /* 没有菜单背景显示函数  */
menu.buttons      = (BUTTON_TYPE *)malloc(6 * sizeof(BUTTON_TYPE));
menu.button_count = 6;
menu.current      = 4;          /* 当前菜单条目序号      */
menu.saveimage    = NO;        /* 不存储菜单背景        */
```

该菜单的 6 个按键大小一样,正常颜色为灰色,当前菜单按键为红色。各菜单按键的等价键盘键码和返回码均为数字键'1'至'6'。菜单条目设置部分的源程序为

```
/* ----- 设置字库工具软件 hantool 的菜单条目参数 ----- */
for(i=0;i<6;i++)
{
    menu.buttons[i].lock      = NO;
    menu.buttons[i].butncolor = GRAY_BAR;
    menu.buttons[i].pushcolor = RED_BAR;
    menu.buttons[i].fun       = NULL;
    menu.buttons[i].w         = 160;
    menu.buttons[i].h         = 34;
    menu.buttons[i].high      = 2;
    menu.buttons[i].fonttimes = 257;
}
```

```
menu.buttons[0].x      = 0;
menu.buttons[0].y      = 0;
menu.buttons[0].name    = "造      字";
menu.buttons[0].key     = '1';
menu.buttons[0].pressed = '1';

menu.buttons[1].x      = 200;
menu.buttons[1].y      = 0;
menu.buttons[1].name    = "编辑小字库";
menu.buttons[1].key     = '2';
menu.buttons[1].pressed = '2';

menu.buttons[2].x      = 400;
menu.buttons[2].y      = 0;
menu.buttons[2].name    = "小字库转换为源程序";
menu.buttons[2].key     = '3';
menu.buttons[2].pressed = '3';

menu.buttons[3].x      = 0;
menu.buttons[3].y      = 40;
menu.buttons[3].name    = "重建小字库索引";
menu.buttons[3].key     = '4';
menu.buttons[3].pressed = '4';

menu.buttons[4].x      = 200;
menu.buttons[4].y      = 40;
menu.buttons[4].name    = "修改参数设置";
menu.buttons[4].key     = '5';
menu.buttons[4].pressed = '5';

menu.buttons[5].x      = 400;
menu.buttons[5].y      = 40;
menu.buttons[5].name    = "退      出";
menu.buttons[5].key     = KEY_ESC;
menu.buttons[5].pressed = KEY_ESC;
```

由以上菜单条目参数可知:该菜单共有6个大小相同的立体型菜单按键,平常为灰色,被选中的当前菜单按键为红色键。在设置菜单和菜单条目参数时最重要的一点的是每个参数都要设置,不能遗漏。例如键名放大倍数fonttimes,即使是横向放大倍数和纵向放大倍数均为1的正常状态,也不能置之不理,而应该赋值257(字节0和字节1的值均为1)。

上述菜单条目参数设置还说明该菜单采用根据返回值处理功能模块的方法。其程序结构(已经过简化)为

```

/* ----- 第一次调用菜单函数:画出菜单及其中的按键 ----- */
button_menu(&menu,0);

/* -- 循环:根据菜单函数的返回值调用相应的功能处理模块 -- */
do
{
    light_mouse();
    key = button_menu(&menu,geth());
    delight_mouse();
    switch(key)
    {
        case '1':                /* 进入造字          */
            makefont(col,line,fontwidth,fonhigh,hzk);
            break;
        case '2':                /* 编辑小字库      */
            edit_xzk(col,line,fontwidth,fonhigh,hzk);
            break;
        case '3':                /* 小字库转换为源程序 */
            xzk_to_sour(10,100);
            break;
        case '4':                /* 重建小字库索引    */
            reindex_xzk(col,line,fontwidth,fonhigh);
            break;
        case '5':                /* 修改参数表        */
            modify_parameters(&fontwidth,&fonhigh,hzkname,curridx,
                              xzkname,idxname,sorname);
            break;
    }
}while(key!=menu.quit_key1 && key!=menu.quit_key2);

```

### 程序 13-3 计算器。

HANENV 系统配备了一个小巧玲珑的计算器函数,调用该函数可以在屏幕上显示一个非常精致的弹出式计算器,如图 13-3 所示。该计算器的功能很强,不但可以用来做四则运算,还可以求出许多常用函数的值。其使用方法也比较新颖:计算器顶部共有两个窗口,左边一个是输入窗口,操作人员输入的计算式就在该窗口显示;右边一个是结果显示窗口,用于显示计算结果和出错信息。这种



图 13-3 计算器

操作方法允许操作人员计算整个运算式的值,当然比真实计算器的分步计算方便得多。

该函数也是使用按键式菜单函数构造的。菜单设置的的主要特点为可用鼠标拖动整个计算器在屏幕上移动,弹出式(保存屏幕背景),不允许使用键盘操作菜单。我们要说明的是不允许使用键盘操作菜单只是不设当前按键,不能使用光标移动键和回车键选择功能项目,但是仍然可以直接使用按键的等价键盘码选择计算器(菜单)按键。下面是计算器模块中用于设置菜单参数的程序片段:

```
/* ----- 初始化计算器:设置按键式菜单的有关参数 ----- */
/* ----- 设置计算器的大小和位置 ----- */
c->col      = col;
c->line     = line;
c->width    = 40;
c->high     = 130;

/* ----- 可用鼠标控制计算器在屏幕上移动 ----- */
c->fixed     = NO;
c->left      = 0;
c->top       = 0;
c->right     = 639;
c->bottom    = 419;
c->saveimage = YES;

/* ----- 不能使用键盘控制菜单 ----- */
c->arrow_ctrl = NO;
c->quit_key1  = KEY_ESC;
c->quit_key2  = RIGHT_BUTTON;

/* ----- 计算器共有 40 个按键 ----- */
c->buttons    = (BUTTON_TYPE *)malloc(40 * sizeof(BUTTON_TYPE));
c->button_count = 40;
/* ----- 使用函数_DisplayCal 显示计算器体 ----- */
c->disp_fun   = _DisplayCal;

/* ----- 为内部变量赋初值 ----- */
c->pop_up     = 0;
```

以上程序段中的显示计算器函数\_DisplayCal 用于显示计算器的壳体和显示窗口。因为计算器被设计为可以在屏幕上移动,所以该函数中的坐标均为相对坐标:

```
/* ----- 内部函数_DisplayCal: 显示计算器 ----- */
static void _Cdecl_DisplayCal(BUTTON_MENU *c)
{
```

```

    _Bar (c->col * 8, c->line, c->width * 8, c->high, LIGHTGRAY, 0xffff0000);
    draw_rectangle(c->col * 8, c->line, c->width * 8, c->high, BLACK, 0xff);
    _Hole(c->col * 8 + 1, c->line + 1, c->width * 8 - 2, c->high - 2, WHITE,
        DARKGRAY, 3);
    _Bar ((c->col + 2) * 8 - 2, c->line + 8, 195, CHAR _HIGH, LIGHTBLUE,
        0xffff0000);
    _Bar ((c->col + 27) * 8 - 2, c->line + 8, 91, CHAR _HIGH, LIGHTBLUE,
        0xffff0000);
    _Hole((c->col + 2) * 8 - 2, c->line + 7, 195, CHAR _HIGH + 2, DARKGRAY,
        WHITE, 1);
    _Hole((c->col + 27) * 8 - 2, c->line + 7, 91, CHAR _HIGH + 2, DARKGRAY,
        WHITE, 1);
}

```

计算器函数中的菜单调用部分的源程序代码段(已经过简化)为

```

/* ----- 显示计算器 ----- */
button_menu(&c, 0);

/* ----- 主循环：处理菜单按键信息 ----- */
do
{
    /* ----- 从计算器菜单接收一个键码 ----- */
    key = button_menu(&c, geth());

    /* ----- 如果是等号键则计算表达式 ----- */
    else if(key == '=')
    {
        result = expression(exp);
        putxys(c.col + 27, c.line + 8, WHITE, LIGHTBLUE, string);
    }

    /* ----- 如果是清除键则清除表达式及其显示 ----- */
    else if(key == KEY_ENTER)
    {
        _Block(c.col + 2, c.line + 8, 23, CHAR _HIGH,
            LIGHTBLUE);
        _Block(c.col + 27, c.line + 8, 11, CHAR _HIGH, LIGHTBLUE);
    }

    /* ----- 如果是退格键则删除光标前的字符 ----- */
    else if(key == Backspace && disp_pos)
    {

```

```

    _Block(c.col+2+(--disp_pos),c.line+8,1,CHAR_HIGH,
          LIGHTBLUE);
    --exp_pos;
}

/* ----- 如果输入键为 ASCII 码则加入表达式 ----- */
else if(isprint(key))
{
    outxyc(c.col+2+disp_pos++,c.line+8,WHITE,key);
    exp[exp_pos++] = key;
}
}while(key!=c.quit_key1 && key!=c.quit_key2);

```

上述程序段已经经过简化,删去了诸如错误处理和编辑运算式等内容。如果读者对计算器函数的设计有兴趣,可以参看随书1#软盘中计算器函数的源程序。

#### 程序13-4 下拉式菜单举例。

下面我们以一个下拉式菜单为例说明下拉式菜单的一般构造方法。该下拉式菜单共有两级,主菜单的5个项目一字横排,显示在屏幕的顶端。二级菜单是弹出式光条菜单,由主菜单驱动,如图13-4所示。值得注意的是在本程序中我们将二级菜单的驱动函数设计为通过主菜单内部各菜单条目的函数指针fun直接驱动,因而简化了程序设计。同时二级菜单对各个功能模块的调用也是通过这种方法进行的。因此,在设计该程序的各个功能模块时只需要直接填写process00等函数的函数体即可。这个程序虽然比较长,但结构很简单。为了节省篇幅,我们只列出了主菜单和第一个二级菜单的驱动函数的源程序。

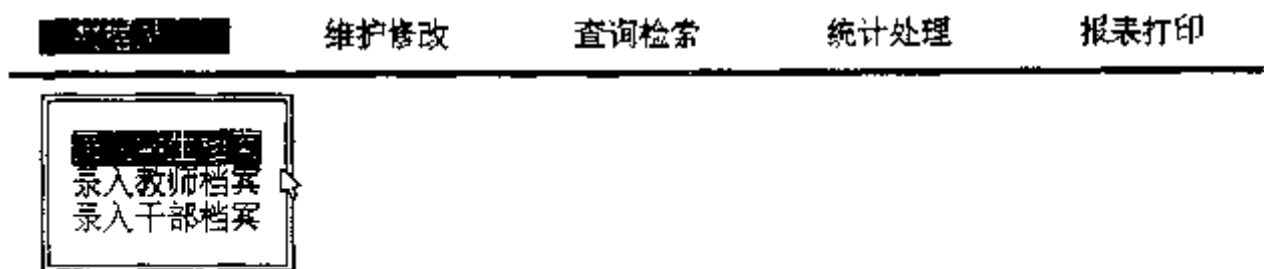


图13-4 下拉式菜单

```

/* -----
下拉式菜单设计举例
----- */

#include (hanenv.h)
#include (alloc.h)
#include (fcntl.h)

/* ----- 下拉式菜单驱动函数 ----- */

```



```

void pull_down(void)
{
    BUTTON_MENU main_menu;
    int i;
    unsigned key;

    /* ----- 设置主菜单的参数 ----- */
    main_menu.col          = 0;
    main_menu.line         = 18;
    main_menu.width        = 80;
    main_menu.high         = CHAR_HIGH;
    main_menu.arrow_ctrl   = YES;
    main_menu.fixed        = YES;
    main_menu.quit_key1    = KEY_ESC;
    main_menu.quit_key2    = KEY_ESC;
    main_menu.press_key1   = KEY_ENTER;
    main_menu.press_key2   = KEY_ENTER;
    main_menu.forward1     = KEY_Right;
    main_menu.forward2     = KEY_Right;
    main_menu.backward1    = KEY_Left;
    main_menu.backward2    = KEY_Left;
    main_menu.pop_up       = NO;
    main_menu.current      = 0;
    main_menu.saveimage     = NO;
    main_menu.disp_fun      = NULL;
    main_menu.button_count = 5;
    main_menu.buttons       = (BUTTON_TYPE *)malloc(5
                                                * sizeof(BUTTON_TYPE));

    /* ----- 设置主菜单中各条目的参数 ----- */
    for(i=0;i<5;i++)
    {
        main_menu.buttons[i].x      = (i*16+2)*8;
        main_menu.buttons[i].y      = 0;
        main_menu.buttons[i].w      = 96;
        main_menu.buttons[i].h      = CHAR_HIGH;
        main_menu.buttons[i].high   = 0;
        main_menu.buttons[i].butncolor = LIGHTCYAN<<4|BLACK;
        main_menu.buttons[i].pushcolor = RED_BAR;
        main_menu.buttons[i].fonttimes = 257;
        main_menu.buttons[i].key    = i;
        main_menu.buttons[i].pressed = i;
        main_menu.buttons[i].lock   = NO;
    }
}

```

```

    }

    main_menu.buttons[0].fun = submenu0;
    main_menu.buttons[1].fun = submenu1;
    main_menu.buttons[2].fun = submenu2;
    main_menu.buttons[3].fun = submenu3;
    main_menu.buttons[4].fun = submenu4;

    main_menu.buttons[0].name= "数据录入";
    main_menu.buttons[1].name= "维护修改";
    main_menu.buttons[2].name= "查询检索";
    main_menu.buttons[3].name= "统计处理";
    main_menu.buttons[4].name= "报表打印";

    /* ----- 显示主菜单的条目 ----- */
    button_menu(&main_menu,0);
    /* ----- 主循环:处理各子菜单 ----- */
    do
    {
        key = getch();
        key = button_menu(&main_menu,key);
    }while(key!=main_menu.quit_key1 && key!=main_menu.quit_key2);
}

/* ----- 子菜单“数据录入”的处理函数 ----- */
unsigned submenu0(void)
{
    BUTTON_MENU menu;
    int i;
    unsigned key;

    /* ----- 设置子菜单的参数 ----- */
    menu.col          = 0;
    menu.line         = 56;
    menu.width        = 16;
    menu.high         = 5 * CHAR_HIGH;
    menu.arrow_ctrl   = YES;
    menu.fixed        = YES;
    menu.quit_key1    = KEY_Right;
    menu.quit_key2    = KEY_Left;
    menu.press_key1   = KEY_ENTER;
    menu.press_key2   = KEY_ENTER;
    menu.forward1     = KEY_Down;
    menu.forward2     = KEY_Down;

```

```

menu.backward1 = KEY_Up;
menu.backward2 = KEY_Up;
menu.pop_up    = NO;
menu.current   = 0;
menu.saveimage = YES;
menu.disp_fun  = disp_submenu;
menu.button_count = 3;
menu.buttons = (BUTTON_TYPE *)malloc(3 * sizeof(BUTTON_TYPE));

```

/\* —— 设置子菜单项目的参数 —— \*/

```

for(i=0;i<3;i++)
{
    menu.buttons[i].x      = 16;
    menu.buttons[i].y      = (i+1)*CHAR_HIGH;
    menu.buttons[i].w      = 96;
    menu.buttons[i].h      = CHAR_HIGH;
    menu.buttons[i].high   = 0;
    menu.buttons[i].butncolor = LIGHTCYAN<<4|BLACK;
    menu.buttons[i].pushcolor = RED_BAR;
    menu.buttons[i].fonttimes = 257;
    menu.buttons[i].key     = i;
    menu.buttons[i].pressed = i;
    menu.buttons[i].lock    = NO;
}

```

```

menu.buttons[0].fun = process00;
menu.buttons[1].fun = process01;
menu.buttons[2].fun = process02;
menu.buttons[0].name = "录入学生档案";
menu.buttons[1].name = "录入教师档案";
menu.buttons[2].name = "录入干部档案";

```

/\* —— 显示子菜单 —— \*/

```
button_menu(&menu,0);
```

/\* —— 处理子菜单的项目 —— \*/

```

do
{
    key = geth();
    key = button_menu(&menu,key);
}while(key!=menu.quit_key1 && key!=menu.quit_key2
    && key!=KEY_ESC);
if(key==KEY_Left || key==KEY_Right)
{

```



```

int y;                /* 图标左上角行坐标(单位为像素) */
int width;            /* 图标宽(单位为像素) */
int high;            /* 图标高(单位为像素) */
char * title;         /* 条目标题 */
unsigned key;         /* 返回值 */
void (* draw_icon)(); /* 图标绘制函数 */
unsigned (* fun)();    /* 选中本条目时执行的功能 */
}ICON_TYPE;

```

下面我们对其中各参数的功能和设置方法作一简单说明:

1. 图标的位置(x 和 y)与大小(width 和 high):图标的位置和大小决定了响应鼠标的范围。与按键式菜单中按键的位置为相对于菜单位置的坐标不同,图标式菜单的图标位置是相对于屏幕左上角(0,0)处的绝对位置。

2. 菜单条目的标题(title):标题是一个字符串,用于说明该菜单条目的功能。当前菜单条目的标题被显示在屏幕上指定位置的窗口中。

3. 返回值(key)和函数指针(fun):返回值和函数指针都用于执行该菜单条目所对应的功能模块。如果函数指针不等于空,而是指向某个处理函数,则选择该菜单条目之后就会执行该函数;如果函数指针等于空,则菜单函数返回上述返回值(key)。该处理函数的编写方法为

```

unsigned process(void)
{
    .....
}

```

即该函数执行完毕后可以返回一个统一键盘码,该键码会作为菜单函数的返回值继续返回。特别要注意的是如果选用返回值来表示菜单功能的话,一定要置函数指针为空,否则程序无法正常运行。

4. 图标绘制函数(draw\_icon):由应用程序员提供,用于修改当前图标。该函数的格式应该为

```

void draw_icon(ICON_TYPE * icon,int iscurrent)
{
    .....
}

```

其中参数 iscurrent 表示所绘图标的类型。如果该参数的值为 YES,表示绘制当前图标(与其他图标应有明显的区别,例如色彩风格不同或者在图标上加有标记),否则表示绘制普通图标。如果不允许使用键盘控制图标式菜单,也可以将该函数指针置为空,表示在菜单操作过程中不改变图标。当然,无论是否允许使用键盘控制图标式菜单,在运行图标式菜单函数之前,都应该在屏幕上先画好所有的图标。

图标式菜单结构类型的定义为

```
/* ----- 有关图标式菜单的结构类型定义 ----- */
typedef struct          /* 图标式菜单类型          */
{
    int x;               /* 条目标题的列坐标(单位为字节) */
    int y;               /* 条目标题的行坐标(单位为象素) */
    int width;           /* 条目标题的宽度(单位为字节)   */
    int arrow_ctrl;      /* 可否用键盘控制菜单          */
    unsigned forward1;   /* 向前移动当前菜单条目所用键  */
    unsigned forward2;   /* 向前移动当前菜单条目所用键  */
    unsigned backward1;  /* 向后移动当前菜单条目所用键  */
    unsigned backward2;  /* 向后移动当前菜单条目所用键  */
    unsigned press_key1; /* 选择当前菜单条目所用键      */
    unsigned press_key2; /* 选择当前菜单条目所用键      */
    int icon_count;      /* 图标数量                    */
    ICON_TYPE * icons;   /* 指向图标表的指针            */
    int current;         /* 当前菜单条目的序号          */
    int mouse_x;         /* 当前鼠标列坐标              */
    int mouse_y;         /* 当前鼠标行坐标              */
} ICON_MENU;
```

下面我们简单说明其中各参数的含义及其设置方法:

1. 条目标题的位置(x 和 y)与宽度(width):每当鼠标光标进入某个菜单条目的图标范围之内,该菜单条目的标题就会显示在由这几个参数确定的屏幕窗口之中。

2. 可否使用键盘控制菜单(arrow\_ctrl)以及各控制键(forward1、forward2、backward1、backward2、press\_key1 和 press\_key2):一般情况下总是使用鼠标控制图标式菜单。如果设置参数 arrow\_ctrl 之值为 YES,则也可以使用各键盘控制键操纵菜单。这时必须为每个菜单条目编写图标显示函数,在该函数中当前图标和其他图标要有明显的区别标志。向前和向后移动键用于重新选择当前菜单条目,选择键用于选中当前菜单条目,相当于使用鼠标光标点选当前图标。

3. 图标数量(icon\_count)和指向图标表的指针(icons):这两个参数确定了菜单中使用的菜单条目表及其条目个数。

4. 当前图标(current):如果没有特殊情况的话,可以置该参数的值为 0。

5. 鼠标坐标(mouse\_x 和 mouse\_y):用于在程序中保存鼠标的最新坐标。其初始设置并不重要。

有了上述两个结构类型定义之后,就可以开始设计我们的图标式菜单处理函数了。我们设计的图标式菜单处理函数的使用方法为

```
/* -----
图标式菜单函数的使用方法
```

```

----- */
BUTTON_MENU menu;          /* 定义一个菜单变量 */
unsigned key;               /* 接收菜单函数的返回值 */
/* ----- 用于设置图标式菜单及其菜单条目的程序段 ----- */
.....

/* ----- 自编函数,用于显示菜单中的各个图标 ----- */
for(i=0;i(menu.icon_count;i++)
    draw_icons(menu.icons,NO);

/* ----- 循环处理对于菜单函数的调用 ----- */
do
{
    /* ----- 调用图标式菜单函数 ----- */
    key = icon_menu(&menu);

    /* ----- 如果使用返回值控制个功能模块的调用 ----- */
    switch(key)
    {
        case (键值1)
            调用功能模块1的处理函数;
            break;
        case (键值2)
            调用功能模块2的处理函数;
            break;
        .....
    }
    /* -- 如果使用按键中定义的处理函数,以上一段可不用 -- */

    /* -- 在此循环中还可以根据鼠标状态处理其他事务 ----- */
    .....

}while(key!=KEY_ESC && key!=RIGHT_BUTTON);

```

由以上程序结构可以看出图标式菜单函数的一般调用方法。该函数的说明为

```
unsigned icon_menu(ICON_MENU * menu);
```

对图标式菜单函数的调用应该由一个循环控制。如果在菜单的各菜单条目的定义中没有定义与图标对应的处理模块,则可以在此循环中根据菜单函数的返回值进行处理。另外,如果在处理图标的同时还要使用鼠标控制其他事务,也可以将其处理放在这个循环中。此时可以首先判断图标式菜单的返回值是否鼠标按键,如果是,再判断鼠标光标的位置,然后

再进行相应的处理。

#### 程序 13-5 图标式菜单。

该函数的工作流程为

```
查询鼠标光标的位置及鼠标按钮状态；
如果鼠标移动了，则
{
    查看每个鼠标光标是否进入了某个图标的范围；
    如果鼠标光标确实进入了某个图标的范围，则
    {
        如果当前条目有图标绘制函数，则
            将当前条目的图标改为一般图标；
        修正当前菜单条目序号；
        如果当前条目有图标绘制函数，则
            将新当前菜单条目的图标改为当前图标；
        显示新的当前菜单条目的说明；
    }
    记下鼠标光标的新位置；
}
如果允许使用键盘控制图标式菜单并且有键按下，则
{
    读键盘缓冲区中的键盘码；
    如果是移当前图标动键，则
    {
        如果当前条目有图标绘制函数，则
            将当前条目的图标改为一般图标；
        修正当前菜单条目序号；
        如果当前条目有图标绘制函数，则
            将新当前菜单条目的图标改为当前图标；
        显示新的当前菜单条目的说明；
    }
    如果是项目选择键，则
    {
        如果当前菜单条目有对应处理函数，则
            调用该处理函数；
        否则
            返回当前条目的返回键码；
    }
}
如果按下了鼠标左键且鼠标光标在当前图标上，则
{
    如果当前菜单条目有对应处理函数，则
```



```

        调用该处理函数;
    否则
        返回当前条目的返回键码;
    }

/* -----
   函数 icon_menu : 图标式菜单
   ----- */
#include <hanenv.h>

unsigned _Cdecl icon_menu(menu)
ICON_MENU * menu;          /* 图标式菜单      */
{
    int mouse_x, mouse_y;    /* 鼠标光标的位置  */
    unsigned key = 0;        /* 键盘键码及返回码 */

    /* ----- 查询鼠标光标的位置及鼠标按钮状态 ----- */
    key = get_mouse_status(&mouse_x, &mouse_y);

    /* ----- 如果鼠标移动了 ----- */
    if(mouse_x != menu->mouse_x || mouse_y != menu->mouse_y)
    {
        int i;

        /* ----- 查看每个菜单条目中的图标范围 ----- */
        for(i=0; i<menu->icon_count; i++)
        /* ----- 如果鼠标光标进入了某个图标的范围 ----- */
        if(mouse_enter(menu->icons[i].x, menu->icons[i].y,
            menu->icons[i].x+menu->icons[i].width,
            menu->icons[i].y+menu->icons[i].high)
            && i != menu->current)
        {
            /* ----- 如果当前条目有图标绘制函数, 抹去图标 ----- */
            if(menu->icons[menu->current].draw_icon)
                (* (menu->icons[menu->current].draw_icon))(menu->icons
                    + menu->current, NO);
            /* ----- 修正当前菜单条目 ----- */
            menu->current = i;
            /* ----- 如果当前条目有图标绘制函数, 重画图标 ----- */
            if(menu->icons[menu->current].draw_icon)
                (* (menu->icons[menu->current].draw_icon))(menu->icons
                    + menu->current, YES);
            /* ----- 显示新当前菜单条目的说明 ----- */

```

```

        _Bar(menu->x, menu->y, menu->width, _Ytimes * CHAR _HIGH,
            _TitleBk, 0xffff0000);
        drawxystr(menu->x, menu->y, _TitleColor,
            menu->icons[menu->current].title);
    }
    /* ----- 记下鼠标光标的新位置 ----- */
    menu->mouse_x = mouse_x;
    menu->mouse_y = mouse_y;
}
/* -- 如果允许使用键盘控制图标式菜单并且有键按下 -- */
if(menu->arrow_ctrl && bioskey(1))
{
    /* ----- 读键盘缓冲区中的键盘码 ----- */
    key = geth();

    /* ----- 如果是向前移动键 ----- */
    if(key == menu->forward1 || key == menu->forward2)
    {
        /* -- 如果当前菜单条目有图标绘制函数, 抹去图标 -- */
        if(menu->icons[menu->current].draw_icon)
            (* (menu->icons[menu->current].draw_icon))(menu->icons
                + menu->current, NO);
        /* ----- 调整当前菜单条目 ----- */
        if(menu->current == menu->icon_count - 1)
            menu->current = 0;
        else
            menu->current++;

        /* -- 如果当前菜单条目有图标绘制函数, 重画图标 -- */
        if(menu->icons[menu->current].draw_icon)
            (* (menu->icons[menu->current].draw_icon))(menu->icons + menu->current,
                YES);

        /* ----- 显示新当前菜单条目的说明 ----- */
        _Bar(menu->x, menu->y, menu->width, _Ytimes * CHAR _HIGH, _TitleBk,
            0xffff0000);
        drawxystr(menu->x, menu->y, _TitleColor,
            menu->icons[menu->current].title);
    }
    /* ----- 如果是向后移动键 ----- */
    else if(key == menu->backward1 || key == menu->backward2)
    {

```

```

/* -- 如果当前菜单条目有图标绘制函数,抹去图标 -- */
if(menu->icons[menu->current].draw_icon)
    (* (menu->icons[menu->current].draw_icon))(menu->icons
        +menu->current,NO);
/* ----- 调整当前菜单条目 ----- */
if(menu->current == 0)
    menu->current = menu->icon_count-1;
else
    menu->current--;

/* -- 如果当前菜单条目有图标绘制函数,重画图标 -- */
if(menu->icons[menu->current].draw_icon)
    (* (menu->icons[menu->current].draw_icon))(menu->icons
        +menu->current,YES);

/* ----- 显示新当前菜单条目的说明 ----- */
_bar(menu->x,menu->y,menu->width,_Ytimes*CHAR_HIGH,_TitleBk,
    0xffff0000);
drawxystr(menu->x,menu->y,_TitleColor,
    menu->icons[menu->current].title);
}
/* ----- 如果是项目选择键 ----- */
else if(key == menu->press_key1 || key == menu->press_key2)
{
    /* -- 如果当前菜单条目有对应处理函数,调用之 -- */
    if(menu->icons[menu->current].fun)
        return (* (menu->icons[menu->current].fun))
            (menu->icons[menu->current]);

    /* ----- 否则返回当前条目的返回键码 ----- */
    else
        return menu->icons[menu->current].key;
}
}

/* -- 如果按下了鼠标左键且鼠标光标在当前图标上 -- */
if(key == LEFT_BUTTON &&
    mouse_enter(menu->icons[menu->current].x,
        menu->icons[menu->current].y,
        menu->icons[menu->current].x+menu->icons[menu->current].width,
        menu->icons[menu->current].y+menu->icons[menu->current].high))
{
    /* -- 如果当前菜单条目有对应处理函数,调用之 -- */
    if(menu->icons[menu->current].fun)

```

```
        return ( * (menu->icons[menu->current].fun))
            (menu->icons[menu->current]);

/* ----- 否则返回当前条目的返回键码 ----- */
else
    return menu->icons[menu->current].key;
}
/* -- 返回 0 表示此次应继续对图标式菜单进行操作 -- */
return key;
}
```

## 第 4 篇

# HANENV 系统应用

## HANENV 系统的头文件

由于 HANENV 系统要处理的内容很多,所以我们将设计 HANENV 系统时涉及的各种定义和说明集中到一个头文件 `hanenv.h` 中去。下面我们介绍头文件 `hanenv.h` 的设计思想,因为内容较多,且所有的函数将在第 15 章中作详细的说明,因此在本章中我们重点介绍符号常数、结构类型和全局变量的定义和说明。

```

/*****
头文件 HANENV.H：应用程序中文用户界面软件包 HANENV
的定义和说明
作      者：刘路放
日      期：1995.7
*****/

```

由于在 `hanenv.h` 中说明的某些函数的参数表中使用了在头文件 `stdio.h` 中定义的文件结构类型 `FILE` 和在头文件 `dos.h` 中定义的日期结构类型 `struct date` 和时间结构类型 `struct time`, 所以我们在 `hanenv.h` 的开始处插入关于这两个头文件的文件包含编译预处理行。这样, 使用 `hanenv.h` 的应用程序首部就不再需要加入关于这两个头文件的文件包含的编译预处理行了。

```
#include <stdio.h>
#include <dos.h>
/ * -----
```

第一部分：基础与环境

- 定义逻辑常数
- EGA/VGA 图形模式 12H 的显示存储器和寄存器操作
- 鼠标操作
- 键盘操作
- 使用扩充内存

## HANENV 系统的初始化和退出

```

----- */
/* -----
1. 定义逻辑常数
----- */

#ifndef NO
#define NO 0
#define YES 1
#endif

```

在使用 HANENV 系统编写应用程序时可以使用表示逻辑值的符号常数 YES 和 NO, 以提高程序的可读性。

```

/* -----
2. EGA/VGA 图形模式 12H 的有关定义和操作
----- */
/* ----- 省缺颜色常数 ----- */
#ifndef defined(_COLORS)
#define _COLORS
enum COLORS
{
    BLACK,          /* 黑 */
    BLUE,           /* 蓝 */
    GREEN,          /* 绿 */
    CYAN,           /* 青 */
    RED,            /* 红 */
    MAGENTA,        /* 洋红 */
    BROWN,          /* 棕 */
    LIGHTGRAY,      /* 浅灰 */
    DARKGRAY,       /* 深灰 */
    LIGHTBLUE,      /* 亮蓝 */
    LIGHTGREEN,     /* 亮绿 */
    LIGHTCYAN,      /* 亮青 */
    LIGHTRED,       /* 亮红 */
    LIGHTMAGENTA,   /* 亮洋红 */
    YELLOW,         /* 黄 */
    WHITE           /* 白 */
};
#endif

#define EDGE_COLOR 16 /* 屏幕边缘色 */

```

在头文件 graphics.h 中也有关于颜色的枚举类型定义。为了在不使用图形库的情况下

也能以符号名调用图形模式 12H 的 16 种颜色,我们加入了以上关于图形模式省缺颜色的枚举类型定义。

通常屏幕边缘的颜色被设置为与屏幕背景色(0 号颜色寄存器)相同。使用函数 `set_color` 可以改变这一省缺设置,单独设置屏幕边缘的颜色,这样便可达到同屏显示 17 种颜色的效果。在使用函数 `set_color` 和函数 `get_color` 时可以使用符号常数 `EDGE_COLOR` 表示屏幕边缘颜色。

```
/* ----- 说明屏幕窗口的全局变量 ----- */
extern int _ScreenTop;          /* 当前屏幕首行位置 */
extern int _ScreenWidth;        /* 逻辑屏幕宽度(以字节为单位) */
extern int _ScreenHigh;         /* 逻辑屏幕高度(以象素为单位) */
extern int _WindowLeft;         /* 屏幕显示窗口左上角列坐标 */
extern int _WindowTop;          /* 屏幕显示窗口左上角行坐标 */
extern int _WindowHigh;         /* 屏幕显示窗口高度 */
extern int _VideoBusy;          /* 是否正在操作 VGA 寄存器组 */
```

为了便于编写能在大于实际屏幕尺寸的逻辑屏幕上工作的屏幕显示函数,我们定义了一组说明屏幕设置状态的全局变量。这些变量的逻辑含义可参看图 2-1 “逻辑屏幕与实际屏幕窗口的关系”。

全局变量 `_VideoBusy` 用于标记是否正在操作 VGA 显示卡上的显示寄存器组,以避免使用中断 1CH 的正文光标和鼠标光标与之发生冲突。关于这方面的内容可以参看 9.1 节:“BIOS 的时钟中断 1CH”。

这组全局变量的初值为

```
_ScreenTop    : 0;
_ScreenWidth  : 80;
_ScreenHigh   : 480;
_WindowLeft   : 0;
_WindowTop    : 0;
_WindowHigh   : 480;
_VideoBusy    : NO.

/* ----- 向屏幕写操作的类型(_SetWriteMode 函数的参数) ----- */
#define PUT_MODE    0          /* 新数据覆盖屏上原有数据 */
#define AND_MODE    0x08       /* 新数据和屏上原有数据作与运算 */
#define OR_MODE     0x10       /* 新数据和屏上原有数据作或运算 */
#define XOR_MODE    0x18       /* 新数据和屏上原有数据作异或运算 */
```

库函数 `_SetWriteMode` 用于改变屏幕写操作的类型。允许设置的屏幕写操作的类型有:覆盖、与、或以及异或。这组宏定义了表示这些操作的符号常数。



```

/* ----- 对 EGA/VGA 图形模式 12H 的操作函数(宏)----- */
void _Cdecl _SetWriteMode (int write_mode);
void _Cdecl close_display (void);
void _Cdecl open_display (void);
void _Cdecl get_color (int color_no,int * red,int * green,int * blue);
void _Cdecl set_color (int color_no,int red,int green,int blue);
void _Cdecl change_color (int color,int new_color);
char * *_Cdecl getblock (int col,int line,int width,int high);
void _Cdecl putblock (int col,int line,int width,int high,char * * block);
void _Cdecl _MoveImage (int col1,int y1,int width,int h,int col2,int y2);
void _Cdecl _Block (int col,int line,int width,int high,int color);
void _Cdecl _SmoothScroll (int x,int y);

#define set_screen_width(width) (_ScreenWidth = (width))
#define set_screen_high(high) (_ScreenHigh = (high))
#define split_screen(high) (_WindowHigh = (high))
#define smooth_scroll(x,y) (_VideoBusy=1,_SmoothScroll(x,
(_WindowTop=(y))),_VideoBusy=0)

```

上面这组函数和带参数的宏用于改变图形模式 12H 的工作状态或屏幕图形块的保存、恢复和移动。

为了适应虚拟屏幕工作状态,我们改写了关于鼠标光标的有关程序。

```

/* -----
3. 鼠标操作
----- */
/* ----- 说明鼠标工作状态的全局变量 ----- */
extern int _HaveMouse; /* 鼠标装载成功标记 */
extern int _MouseLight; /* 当前鼠标光标是否显示 */
extern int _MouseBusy; /* 是否正在画鼠标光标 */
extern int _MouseCol; /* 当前鼠标光标列坐标(单位为象素) */
extern int _MouseRow; /* 当前鼠标光标行坐标(单位为象素) */
extern int _MouseColor; /* 鼠标光标颜色 */
extern int _MouseBk; /* 鼠标光标边缘颜色 */
extern int _MouseLeft; /* 鼠标活动范围左上角列坐标 */
extern int _MouseTop; /* 鼠标活动范围左上角行坐标 */
extern int _MouseRight; /* 鼠标活动范围右下角列坐标 */
extern int _MouseBottom; /* 鼠标活动范围右下角行坐标 */
/* (以上 4 个变量值的单位均为象素) */
extern int _MouseSpeed; /* 鼠标响应速度 */
extern int _DoublePressTime; /* 双击鼠标按钮的测试时间 */

```

除了可以使用虚拟屏幕之外,我们编写的鼠标操作函数族还具有使用方便、功能设置全面等优点。这组函数用于控制鼠标的功能状态。注意全局变量 `_MouseSpeed` 的作用是控制连续按下鼠标按键时的延迟时间,而变量 `_DoublePressTime` 是测试鼠标双击时的最大时间间隔。这两个全局变量只用于少数几个函数中。以上全局变量的初值为

```

_MouseCol      : 320;
_MouseRow      : 240;
_MouseColor    : WHITE;
_MouseBk       : BLACK;
_MouseLeft     : 0;
_MouseTop      : 0;
_MouseRight    : 639;
_MouseBottom   : 479;
_MouseSpeed    : 200(毫秒);
_DoublePressTime : 400(毫秒);

/* ----- 对鼠标的操作函数(宏) ----- */
void _Cdecl reset _mouse      (void);
void _Cdecl light _mouse      (void);
void _Cdecl delight _mouse    (void);
int _Cdecl get _mouse _status (int *x,int *y);
void _Cdecl set _mouse _pos   (int x,int y);
int _Cdecl mouse _pressed     (int which _button,int *x,int *y,int *count);
int _Cdecl mouse _release     (int which _button,int *x,int *y,int *count);
void _Cdecl set _mouse _range (int left,int top,int right,int bottom);
void _Cdecl get _mouse _range (int *left,int *top,int *right,int *bottom);
void _Cdecl till _mouse _pop   (int which _button);
int _Cdecl double _press      (unsigned key);
void _Cdecl mouse _moved      (int *dx,int *dy);

#define havemouse()          _HaveMouse
#define ismouselight()       _MouseLight
#define mousecolor()         _MouseColor
#define mousebk()            _MouseBk
#define mousecol()           _MouseCol
#define mouserow()           _MouseRow
#define set _mouse _color(color,bk) (_MouseColor=(color),_MouseBk=(bk))
#define mouse _enter(i,t,r,b)   (_MouseCol)=(i)&&_MouseCol<=(r)&&
                                   _MouseRow=(t)&&_MouseRow<=(b))
#define set _double _press _time(time)(_DoublePressTime=(time))

```

HANENV 系统的一个主要特色是为了编程方便重新设计了键盘操作函数。新的键盘

操作库函数 `geth` 不仅可以响应键盘信息,而且还可以对鼠标操作做出反应。另外,我们在该函数中引入了触发器的概念,使得应用程序在线帮助、热键触发的工具等功能的编程非常容易实现。

```

/* -----
   4. 键盘操作
----- */
/* ----- 触发器的类型定义及全局变量的说明 ----- */
typedef struct          /* 触发器的定义 */
{
    unsigned press_key;    /* 按键触发事件 */
    unsigned mouse_affair; /* 鼠标触发事件 */
    int left;              /* 鼠标左边界(单位为像素) */
    int top;               /* 鼠标上边界(单位为像素) */
    int right;             /* 鼠标右边界(单位为像素) */
    int bottom;            /* 鼠标下边界(单位为像素) */
    unsigned (* trigger)(unsigned h); /* 事件函数 */
} TRIGGER;

```

所谓触发器就是一个功能模块,可以由指定的热键或鼠标事件(在指定区域内按下鼠标按钮)激活。上述 `TRIGGER` 类型可以用来定义一个表示触发器事件的变量,其中分量 `trigger` 是一个函数指针,指向触发器事件的功能模块函数。关于触发器的设计可以参看第15章中关于函数 `geth` 的说明。

```

extern int      _TriggerCount;    /* 触发器个数 */
extern TRIGGER  _Triggers[];      /* 触发器 */

```

在 HANENV 系统中最多只允许设置 20 个触发器事件,其中第一个(`Triggers[0]`)已经为定时器占用。

```

extern unsigned (* mouseKB)(unsigned h); /* 虚拟键盘激活条件 */

/* ----- 键盘操作函数的说明 ----- */
unsigned _Cdecl geth          (void);
void     _Cdecl ungetstr      (char * string);
void     _Cdecl ungeth        (unsigned h);
unsigned _Cdecl mouse_keyboard(unsigned h);
void     _Cdecl load_MKB      (unsigned key);
void     _Cdecl enableMKB     (void);
void     _Cdecl disableMKB    (void);
void     _Cdecl set_mouse_KB  (int x,int y,unsigned color,int left,int top,int right,
                               int bottom);

```

```
void _Cdecl free_MKB (void);

/* ----- HANENV 系统的统一键盘编码定义 ----- */
/* ----- 功能键 F1 --- F10 ----- */
#define KEY_F1 315
#define KEY_F2 316
#define KEY_F3 317
#define KEY_F4 318
#define KEY_F5 319
#define KEY_F6 320
#define KEY_F7 321
#define KEY_F8 322
#define KEY_F9 323
#define KEY_F10 324

/* ----- 复合功能键 Ctrl+F1 --- Ctrl+F10 ----- */
#define KEY_Ctr_F1 350
#define KEY_Ctr_F2 351
#define KEY_Ctr_F3 352
#define KEY_Ctr_F4 353
#define KEY_Ctr_F5 354
#define KEY_Ctr_F6 355
#define KEY_Ctr_F7 356
#define KEY_Ctr_F8 357
#define KEY_Ctr_F9 358
#define KEY_Ctr_F10 359

/* ----- 复合功能键 Alt+F1 --- Alt+F10 ----- */
#define KEY_Alt_F1 360
#define KEY_Alt_F2 361
#define KEY_Alt_F3 362
#define KEY_Alt_F4 363
#define KEY_Alt_F5 364
#define KEY_Alt_F6 365
#define KEY_Alt_F7 366
#define KEY_Alt_F8 367
#define KEY_Alt_F9 368
#define KEY_Alt_F10 369

/* ----- 复合功能键 Shift+F1 --- Shift+F10 ----- */
#define KEY_Shift_F1 340
#define KEY_Shift_F2 341
#define KEY_Shift_F3 342
```

```

#define KEY_Shift_F4      343
#define KEY_Shift_F5      344
#define KEY_Shift_F6      345
#define KEY_Shift_F7      346
#define KEY_Shift_F8      347
#define KEY_Shift_F9      348
#define KEY_Shift_F10     349

/* ----- 数字小键盘上的功能键 ----- */
#define KEY_Home          327
#define KEY_Up            328
#define KEY_PgUp          329
#define KEY_Left          331
#define KEY_Right         333
#define KEY_End           335
#define KEY_Down          336
#define KEY_PgDn          337
#define KEY_Ins           338
#define KEY_Del           339

/* ----- 左 Shift 键与数字小键盘的功能键复合 ----- */
#define KEY_LS_Left       587
#define KEY_LS_Up         584
#define KEY_LS_Right      589
#define KEY_LS_Down       592
#define KEY_LS_Home       583
#define KEY_LS_End        591
#define KEY_LS_PgUp       585
#define KEY_LS_PgDn       593

/* ----- 右 Shift 键与数字小键盘的功能键复合 ----- */
#define KEY_RS_Left       459
#define KEY_RS_Up         456
#define KEY_RS_Right      461
#define KEY_RS_Down       464
#define KEY_RS_Home       455
#define KEY_RS_End        463
#define KEY_RS_PgUp       457
#define KEY_RS_PgDn       465

/* ----- Ctrl 键与数字小键盘的功能键复合 ----- */
#define KEY_Ctr_Home      375
#define KEY_Ctr_PgUp      388

```

```
#define KEY_Ctr_Left      371
#define KEY_Ctr_Right     372
#define KEY_Ctr_End       373
#define KEY_Ctr_PgDn      374

/* ----- Alt, Ctrl 和 Shift 键 tab 键的复合 ----- */
#define KEY_Alt_Tab       321
#define KEY_Ctr_Tab       304
#define KEY_Shift_Tab     271

/* ----- Alt 键 + 字母键 ----- */
#define KEY_Alt_A         286
#define KEY_Alt_B         304
#define KEY_Alt_C         302
#define KEY_Alt_D         288
#define KEY_Alt_E         274
#define KEY_Alt_F         289
#define KEY_Alt_G         290
#define KEY_Alt_H         291
#define KEY_Alt_I         279
#define KEY_Alt_J         292
#define KEY_Alt_K         293
#define KEY_Alt_L         294
#define KEY_Alt_M         306
#define KEY_Alt_N         305
#define KEY_Alt_O         280
#define KEY_Alt_P         281
#define KEY_Alt_Q         272
#define KEY_Alt_R         275
#define KEY_Alt_S         287
#define KEY_Alt_T         276
#define KEY_Alt_U         278
#define KEY_Alt_V         303
#define KEY_Alt_W         273
#define KEY_Alt_X         301
#define KEY_Alt_Y         277
#define KEY_Alt_Z         300

/* ----- Alt 键 + 数字键 ----- */
#define KEY_Alt_1         376
#define KEY_Alt_2         377
#define KEY_Alt_3         378
#define KEY_Alt_4         379
```

```
#define KEY_Alt_5      380
#define KEY_Alt_6      381
#define KEY_Alt_7      382
#define KEY_Alt_8      383
#define KEY_Alt_9      384
#define KEY_Alt_0      385
#define KEY_Alt_MINUS  386
#define KEY_Alt_EQUAL   387
```

```
/* ----- Ctrl 键 + 字母键 ----- */
```

```
#define KEY_Ctr_A      641
#define KEY_Ctr_B      642
#define KEY_Ctr_C      643
#define KEY_Ctr_D      644
#define KEY_Ctr_E      645
#define KEY_Ctr_F      646
#define KEY_Ctr_G      647
#define KEY_Ctr_H      648
#define KEY_Ctr_I      649
#define KEY_Ctr_J      650
#define KEY_Ctr_K      651
#define KEY_Ctr_L      652
#define KEY_Ctr_M      653
#define KEY_Ctr_N      654
#define KEY_Ctr_O      655
#define KEY_Ctr_P      656
#define KEY_Ctr_Q      657
#define KEY_Ctr_R      658
#define KEY_Ctr_S      659
#define KEY_Ctr_T      660
#define KEY_Ctr_U      661
#define KEY_Ctr_V      662
#define KEY_Ctr_W      663
#define KEY_Ctr_X      664
#define KEY_Ctr_Y      665
#define KEY_Ctr_Z      666
```

```
/* ----- 常用的 ASCII 码 ----- */
```

```
#define Beep           0x07
#define Backspace      0x08
#define KEY_ENTER      0x0d
#define KEY_ESC        0x1b
#define KEY_SPACE      0x20
```

```

/* ----- 鼠标按钮按下或释放的组合情况 ----- */
#define NO_BUTTON          400
#define LEFT_BUTTON        401
#define RIGHT_BUTTON       402
#define L_R_BUTTON         403

```

函数 geth 可以对键盘输入的 ASCII 码、扩展键盘码、功能键组合、鼠标按钮甚至汉字作出反应。我们将 geth 函数的返回值进行统一编码并提供了一整套宏定义，这样可以大大简化应用程序中的键盘输入处理部分。

```

/* -----
5. 使用扩充存储器
----- */
/* - 通过 INT15H 使用扩充内存的数据类型和全局变量的定义 - - - */
typedef struct          /* 保护地址模式下段的描述符类型定义 */
{
    unsigned    size;      /* 读写缓冲区长度          */
    unsigned    baselow;   /* 基地址低位字            */
    unsigned char basehigh; /* 基地址高位字节          */
    unsigned char attr;    /* 存取权                  */
    unsigned    blank;     /* 保留字                  */
}Descriptor;             /* 描述符类型              */
typedef struct          /* 全局描述符表 GDT 类型   */
{
    Descriptor BlankDsc;   /* 空地址描述符            */
    Descriptor GDTDsc;     /* 本 GDT 表的地址描述符   */
    Descriptor Source;     /* 源数据块地址描述符      */
    Descriptor Destin;     /* 目标数据块地址描述符    */
    Descriptor BiosCS;     /* ROM BIOS 代码地址描述符 */
    Descriptor BiosSS;     /* ROM BIOS 堆栈地址描述符 */
}GDT;                    /* 全局描述符表 GDT 类型   */
extern long _CurrentEMM;   /* 当前自由扩充存储器地址  */
extern GDT _MemCtrlBlock; /* 全局描述符表 GDT 的说明 */

/* ----- 通过 INT15H 使用扩充内存的函数说明 ----- */
void _Cdecl _Init_GDT (void);
unsigned _Cdecl _GetEMMsize (void);
void _Cdecl _SetSourAddr(long addr,unsigned size);
void _Cdecl _SetDestAddr(long addr,unsigned size);
void _Cdecl _MoveDataEMM(unsigned size);
long _Cdecl _LoadEMM (FILE * file,long size);

```



```

/* 通过 XMS 规范使用扩充内存所用的数据结构和全局变量的定义 */
struct EMB                                /* 扩充内存移动结构定义 */
{
    unsigned long len;                    /* 需传输的数据字节数(32 位) */
    unsigned     sour_han;                /* 源数据块句柄 */
    unsigned long sour_off;               /* 源偏移量(32 位) */
    unsigned     dest_han;                /* 目标数据块句柄 */
    unsigned long dest_off;               /* 目标偏移量(32 位) */
};
extern struct EMB _Emb;                  /* 扩充内存移动结构变量 */
extern int _LargestXMS;                  /* 最大自由扩充内存块尺寸 */
extern int _AmountOfXMS;                 /* 剩余自由扩充内存数量总和 */
extern void far (* _FunctionXMS)();      /* XMS 服务程序入口地址 */

/* ----- 通过 XMS 规范使用扩充内存的函数说明 ----- */
void _Cdecl init_XMS    (void);
void _Cdecl _SizeofXMS  (void);
int  _Cdecl _GetXMS     (int size);
int  _Cdecl _ChangeXMS  (int handle,int size);
void _Cdecl _FreeXMS     (int handle);
void _Cdecl _MoveDataXMS(void);
int  _Cdecl _LoadXMS     (FILE * file,long size);
int  _Cdecl getblockXMS  (int left,int top,int right,int bottom);
void _Cdecl putblockXMS  (int left,int top,int right,int bottom,int handle);

#define sizeofXMS()      _AmountOfXMS
#define largestXMS()     _LargestXMS
#define haveXMS()        ((int)_FunctionXMS)

```

在应用程序中使用扩充存储器可以使用两种方法:直接使用 BIOS 功能调用 15H 或者使用 XMS 规范。这两种方法 HANENV 都支持,其原理可参看第 4 章。

```

/* -----
6. HANENV 系统的初始化和退出
----- */
/* -- HANENV 系统的错误检查变量和函数的说明 ----- */
extern int _ErrorNo; /* 某些函数将其出错码放在此变量中 */
int _Cdecl han_error(void);

/* ----- HANENV 系统的初始化等函数的说明 ----- */
extern void interrupt far (* _Oldint1CH)(); /* 原中断 1CH 地址 */
extern void interrupt far (* _Oldint09H)(); /* 原中断 09H 地址 */

```

```

extern void interrupt far (*_Oldint23H)(); /* 原中断 23H 地址 */
extern void far (*_Int1CHfun[])(); /* 时钟处理程序向量 */

void interrupt _Newint1CH(void);
void interrupt _Newint09H(void);
void interrupt _Newint23H(void);
void _Cdecl _SaveEnv (void);
void _Cdecl _LoadEnv (void);
void _Cdecl init_hanenv (void);
void _Cdecl close_hanenv (void);
int _Cdecl call_DOS (char *envname);

/* -----
    第二部分：汉字处理
        汉字库
        汉字显示
        正文方式与光标
        汉字输入模块
    ----- */
/* -----
    1. 有关汉字库的数据类型定义、全局变量和函数说明
    ----- */
typedef struct /* 汉字库类型 */
{
    int fontwidth; /* 点阵宽(以字节为单位) */
    int fonthigh; /* 点阵高(以像素为单位) */
    unsigned char *codelist; /* 指向小字库索引表的指针 */
    int wherefont; /* 字库装载位置 */
    int fontcount; /* 字库中的字模数量 */
    FILE *fontfile; /* 字库文件名 */
    unsigned long EMM_addr; /* 字库在扩充内存中的地址 */
    int fonthandle; /* 字库在 XMS 扩充内存中的句柄 */
    unsigned char *fontbuff; /* 字库区在常规内存中的指针 */
}HZK;

/* ----- 汉字库的装载位置定义 ----- */
#define DSK 0 /* 汉字库在硬盘中 */
#define EMM 1 /* 汉字库在扩充内存中 */
#define XMS 2 /* 汉字库在 XMS 扩充内存中 */
#define MEM 3 /* 小字库在内存中 */
extern HZK *_CurrentHZK; /* 当前汉字库结构变量的指针 */
HZK *_Cdecl load_HZK(int width,int high,char *hzkname,unsigned char *codelist,
                    int wherefont);

```

```

HZK *_Cdecl free_HZK(HZK *hzk);
int _Cdecl takefont(unsigned h);
/* -----
    2. 汉字和西文字符的显示
    ----- */
/* ----- HANENV 系统的判定函数(宏) ----- */
#define isSP(h)      ((unsigned)(h)>0xa0a0)
#define ishan(s)     (*((unsigned*)(s))>0xa0a0)

int _Cdecl isins      (void);
int _Cdecl isedit     (unsigned key);
int _Cdecl isscan     (unsigned h);

/* --- 有关汉字和西文字符显示的全局变量和函数的说明 --- */
#define CHAR_HIGH     18
/* ASCII 码高度(行高) */
#define HORIZONTAL     0 /* 横向显示字模点阵 */
#define VERTICAL       1 /* 纵向显示字模点阵 */

#define SOLIDLINE      0 /* 字模笔划用实线填充 */
#define DOTTEDLINE     1 /* 字模笔划用实线填充 */

extern unsigned char _ChrFont[]; /* 标准 ASCII 码字模点阵 */
extern unsigned char _HanFont[]; /* 汉字显示缓冲区 */
extern int _Xtimes; /* 横向放大倍数 */
extern int _Ytimes; /* 纵向放大倍数 */

```

这两个全局变量用于表示汉字或字符的放大倍数,其初值均为 1。

```

extern int _FontDirection; /* 点阵显示方向 */
extern int _FontBkStyle; /* 字模笔划填充方式 */

```

这两个全局变量决定使用函数 `_DrawFont` 显示字模点阵时的参数,其中全局变量 `_FontDirection` 的初值为 `HORIZONTAL`, `_FontBkStyle` 的初值为 `SOLIDLINE`。

```

#define set_xtimes(times)      (_Xtimes=(times))
#define set_ytimes(times)      (_Ytimes=(times))
#define set_font_direction(direction) (_FontDirection=(direction))
#define set_font_bk_style(style) (_FontBkStyle=(style))
void _Cdecl _DrawF (int x,int y,int width,int high,int color,unsigned char *font);
void _Cdecl _OutF (int col,int line,int width,int high,int color,unsigned char *font);
void _Cdecl _DrawFont(int x,int y,int width,int high,int color,unsigned char *font);
void _Cdecl _OutFont (int col,int line,int width,int high,int color,

```

```

        unsigned char *font);
unsigned char *_Cdecl takehan(unsigned char *string,unsigned *h);
void _Cdecl drawxys (int x,int y,int color,char *s);
void _Cdecl drawxystr(int x,int y,int color,char *string);
void _Cdecl outxys (int col,int line,int color,char *string);
void _Cdecl outxystr (int col,int line,int color,char *string);
void _Cdecl putxys (int col,int line,int color,int bk,char *string);
void _Cdecl putxystr (int col,int line,int color,int bk,char *string);

#define drawxyc(x,y,color,ch) _DrawF(x,y,1,CHAR_HIGH,color,
                                _ChrFont+(ch)*CHAR_HIGH)
#define outxyc(col,line,color,ch) _OutF(col,line,1,CHAR_HIGH,color,
                                _ChrFont+(ch)*CHAR_HIGH)
#define putxyc(col,line,color,bk,ch) _Block(col,line,1,CHAR_HIGH,bk),
                                outxyc(col,line,color,ch)
#define drawxychar(x,y,color,ch) _DrawFont(x,y,1,CHAR_HIGH,color,
                                _ChrFont+(ch)*CHAR_HIGH)
#define outxychar(col,line,color,ch) _OutFont(col,line,1,CHAR_HIGH,color,
                                _ChrFont+(ch)*CHAR_HIGH)
#define putxychar(col,line,color,bk,ch) _Block(col,line,_Xtimes,CHAR_HIGH*
                                Ytimes,bk),outxychar(col,line,color,ch)
#define drawxyh(x,y,color,han) takefont(han),_DrawF(x,y,_CurrentHZK->fontwidth,_Cur-
                                rentHZK->fonthigh,color,_HanFont)
#define outxyh(col,line,color,han)takefont(han),_OutF(col,line,_CurrentHZK->fontwidth,_Cur-
                                rentHZK->fonthigh,color,_HanFont)
#define putxyh(col,line,color,bk,han) _Block(col,line,_CurrentHZK->fontwidth,
                                _CurrentHZK->fonthigh,bk),outxyh(col,line,color,han)
#define drawxyhan(x,y,color,han) takefont(han),_DrawFont(x,y,_CurrentHZK->fontwidth,_
                                CurrentHZK->fonthigh,color,_HanFont)
#define outxyhan(col,line,color,han) takefont(han),_OutFont(col,line,_CurrentHZK->fontwidth,
                                _CurrentHZK->fonthigh,color,_HanFont)
#define putxyhan(col,line,color,bk,han) _Block(col,line,_Xtimes*_CurrentHZK->
                                fontwidth,_CurrentHZK->fonthigh*_Ytimes,bk),outxyhan(col,line,color,han)

/* ----- 扩充微型西文 ASCII 字符 ----- */
#define UP_ARROW      127    /* 向上箭头 */
#define DOWN_ARROW    128    /* 向下箭头 */
#define RIGHT_ARROW   129    /* 向右箭头 */
#define LEFT_ARROW    130    /* 向左箭头 */
#define SMALL_BAR      131    /* 矩形块 */

extern unsigned char _TinyFonts[]; /* 微型 ASCII 码字模点阵 */

```

```
#define drawxyt(x,y,c,ch) (_DrawF(x,y,1,8,c,_TinyFonts+((ch)-' ')*8))
void _Cdecl drawxytiny (int x,int y,int color,char *s);
/* -----
```

### 3. 正文方式下的汉字和字符显示函数

```
----- */
extern int _TextCol;          /* 当前光标列坐标(单位为字节) */
extern int _TextLine;        /* 当前光标行坐标(单位为像素) */
extern int _TextColor;       /* 当前正文前景色 */
extern int _Background;      /* 当前正文背景色 */
extern int _TextWinLeft;     /* 正文窗口左列坐标(单位为字节) */
extern int _TextWinTop;      /* 正文窗口顶行坐标(单位为像素) */
extern int _TextWinRight;    /* 正文窗口右列坐标(单位为字节) */
extern int _TextWinBottom;   /* 正文窗口底行坐标(单位为像素) */
```

这组全局变量用于决定正文方式下的工作参数。其初值分别为

```
_TextCol      : 0;
_TextLine     : 0;
_TextColor    : LIGHTGRAY;
_Background   : BLACK;
_TextWinLeft  : 0;
_TextWinTop   : 0;
_TextWinRight : 80;
_TextWinBottom : 480.
```

```
#define set_text_color(color) (_TextColor=(color))
#define set_background(color) (_Background=(color))
#define set_window(l,t,r,b) (_TextWinLeft=(l),
    _TextWinTop=(t), _TextWinRight=(r), _TextWinBottom=(b))
#define get_window(l,t,r,b) (l=_TextWinLeft,t=_TextWinTop,
    r=_TextWinRight,b=_TextWinBottom)

#define outc(ch)          outxyc(_TextCol,_TextLine,_TextColor,ch)
#define outchar(ch)       outxychar(_TextCol,_TextLine,_TextColor,ch)
#define outh(han)         outxyh(_TextCol,_TextLine,_TextColor,han)
#define outhan(han)       outxyhan(_TextCol,_TextLine,_TextColor,han)
void _Cdecl outh1          (unsigned h,int whichhalf);
void _Cdecl outhan1        (unsigned h,int whichhalf);
void _Cdecl outs           (char *string);
void _Cdecl outstr         (char *string);
void _Cdecl putnstr        (char *string,int width);

/* ----- 光标操作函数(宏) ----- */
#define movecursor(x,y) (_TextCol=(x),_TextLine=(y))
#define iscursorlight() ((int)_Int1CHfun[1])
```

```

void far _CursorFun (void);
void _Cdecl setcursor (int width,int high,int line,int speed,int color);
void _Cdecl lightcursor (void);
void _Cdecl delightcursor (void);
void _Cdecl scroll _up (int high);
void _Cdecl scroll _down (int high);

/* -----
   4. 汉字输入模块
   ----- */
/* ----- 定义汉字输入提示行风格 ----- */
#define WIN_STYLE 0 /* WINDOWS 风格 */
#define DOS_STYLE 1 /* DOS 风格 */
/* ----- 汉字输入法模块的类型定义 ----- */
typedef struct
{
    char modename[9]; /* 汉字输入法名称 */
    int maxcodes; /* 码表长度 */
    char *codeset; /* 输入法所用按键集合 */
    int wildchar; /* 通用替代键 */
    int haveload; /* 该汉字输入法是否已经装载 */
    unsigned (*get_han)(); /* 汉字输入法模块指针 */
    int wherecodes; /* 码表位置 */
    FILE *codefile; /* 码表文件 */
    long filefront; /* 码表文件头指针 */
    long filelen; /* 码表文件长度 */
    int handle; /* 码表在 XMS 扩充内存的句柄 */
    unsigned long EMM_addr; /* 码表在扩充内存中的地址 */
}MODE_TYPE;

```

我们为汉字输入法设计了一个相当全面的结构类型，因而为编写可以自由装卸的汉字输入法模块提供了方便。同时也有利于应用程序的设计人员为 HANENV 系统设计新的输入法模块。

```

/* ----- 全局变量定义 ----- */
extern int _HanMode; /* 当前汉字输入方法 */
extern int _PromptStyle; /* 提示行风格 */
extern int _PmtColor; /* 提示行字符颜色 */
extern int _PmtBk; /* 提示行背景颜色 */
extern int _DispPrompt; /* 提示行是否弹出 */
extern int _PromptLine; /* 提示行位置 */

```

```
extern int _SectionCode;          /* 区码          */
extern int _PositionCode;         /* 位码          */
extern MODE _TYPE _Mode[];       /* 汉字输入法参数 */
```

这些全局变量规定了汉字输入和提示行的工作状态。其初值分别为

```
_HanMode      : ALT_F10(西文);
_PromptStyle   : WIN_STYLL;
_PmtColor      : BLACK;
_PmtBk         : LIGHTGRAY;
_DispPrompt    : NO;
_PromptLine    : 450;
_Mode         : 10个汉字输入法被初始化为
{
    {"国标区位",0,NULL,0,YES,_GetSP, 0},
    {"          ",0,NULL,0,NO, NULL, 0},
    {"          ",0,NULL,0,NO, NULL, 0},
    {"          ",0,NULL,0,NO, NULL, 0},
    {"          ",0,NULL,0,NO, NULL, 0},
    {"英文数字",0,NULL,0,YES,_GetAscii, 0},
    {"          ",0,NULL,0,NO, NULL, 0},
    {"          ",0,NULL,0,NO, NULL, 0},
    {"图形符号",0,NULL,0,YES,_GetFigure, 0},
    {"          ",0,NULL,0,NO, NULL, 0}
}

/* ----- 汉字输入法切换功能键定义 ----- */
#define ALT_F1      0      /* 功能键 Alt+F1      */
#define ALT_F2      1      /* 功能键 Alt+F2      */
#define ALT_F3      2      /* 功能键 Alt+F3      */
#define ALT_F4      3      /* 功能键 Alt+F4      */
#define ALT_F5      4      /* 功能键 Alt+F5      */
#define ALT_F6      5      /* 功能键 Alt+F6      */
#define ALT_F7      6      /* 功能键 Alt+F7      */
#define ALT_F8      7      /* 功能键 Alt+F8      */
#define ALT_F9      8      /* 功能键 Alt+F9      */
#define ALT_F10     9      /* 功能键 Alt+F10     */

/* ----- 汉字输入模块函数说明 ----- */
#define set_prompt_style(color,bk,style) (_PmtColor=(color),_PmtBk=(bk),
                                         _PromptStyle=(style))
#define set_prompt_line(line)           (_PromptLine=(line))

unsigned _Cdecl _GetSP      (unsigned h);
unsigned _Cdecl _GetAscii  (unsigned h);
```

```

unsigned _Cdecl _GetFigure      (unsigned h);
unsigned _Cdecl _SetAscMode     (unsigned h);
unsigned _Cdecl _GetNewCode     (unsigned h);
unsigned _Cdecl _GetPY         (unsigned h);
void _Cdecl _ClearPrompt       (int clear_mode);
void _Cdecl set_asc_mode       (void);
int _Cdecl set_han_mode        (int altfn, char * modename,
                                unsigned (* get_han)(), char * codefile,
                                int wherecodes);

void _Cdecl free_mode          (int altfn);
unsigned _Cdecl gethan          (void);

```

```
/* ----- */
```

### 第三部分：用户界面

点、线、框、曲线

仿 WINDOWS 窗口式界面的构件函数

菜单设计

全屏幕数据录入函数

定时器

字符串处理函数

表达式与计算器

```
----- */
```

```
/* ----- */
```

#### 1. 屏幕作图函数

```
----- */
```

```

void _Cdecl _PutPixel          (int x, int y, int color);
int _Cdecl _GetPixel           (int x, int y);
void _Cdecl _H_Line            (int x, int y, int len, int color, unsigned char pattern);
void _Cdecl _V_Line            (int x, int y, int len, int color, unsigned char pattern);
void _Cdecl draw_rectangle     (int x, int y, int width, int high, int color,
                                unsigned char pattern);

void _Cdecl draw_line          (int x1, int y1, int x2, int y2, int color);
void _Cdecl _Bar                (int x, int y, int width, int hight, int color,
                                unsigned long pattern);

```

```
/* ----- 画弧(扇形的填充参数) ----- */
```

```

#define ONLY_ARC                0          /* 只画弧线 */
#define DRAW_SECTOR             1          /* 画空心扇形 */
#define FILL_SECTOR              2          /* 画实心扇形 */

```

```
void _Cdecl _DrawArc           (int col, int row, int a, int b, int t1, int t2, int color, int fill);
```

```
#define draw_circle(x, y, r, color) _DrawArc(x, y, r, r, 0, 360, color, ONLY_ARC)
```



```

#define draw_arc(x,y,r,t1,t2,color) _DrawArc(x,y,r,r,t1,t2,color,ONLY_ARC)
#define draw_pie(x,y,r,color) _DrawArc(x,y,r,r,0,360,color,
                                     FILL_SECTOR)
#define draw_fan(x,y,r,t1,t2,color) _DrawArc(x,y,r,r,t1,t2,color,
                                     FILL_SECTOR)
#define draw_ellipse(x,y,a,b,color) _DrawArc(x,y,a,b,0,360,color,ONLY_ARC)

/* 西文制表符画线、框的线型(用于填写以下函数的参数 type) — */
#define SINGLE 0 /* 单线 */
#define DOUBLE 1 /* 双线 */

void _Cdecl char _rectangle (int col,int line,int width,int high,int type);
void _Cdecl char _v_line (int col,int line,int len,int type);
void _Cdecl char _h_line (int col,int line,int len,int type);

```

尽管 HANENV 系统可以和 Turbo C 的图形库联合使用编程,但是只要需要设计的图案不是特别复杂,例如只是设计类 WINDOWS 风格的用户界面时,HANENV 系统自己的作图类库函数已经能够对付,因而降低了编程的复杂性。

```

/* -----
2. 仿 WINDOWS 窗口式界面的构件函数
----- */

/* — 按钮颜色(用于填写使用按键的函数的颜色参数及 _BarColor) — */
#define BLUE_BAR 0x891f /* 蓝色按键 */
#define GREEN_BAR 0x8a2f /* 绿色按键 */
#define CYAN_BAR 0x8b3f /* 青色按键 */
#define RED_BAR 0x8c4f /* 红色按键 */
#define MAGENTA_BAR 0x8d5f /* 洋红色按键 */
#define BROWN_BAR 0x8e6f /* 棕色按键 */
#define GRAY_BAR 0x8f70 /* 灰色按键 */

/* ----- 界面色彩全局变量 ----- */
extern unsigned _BarColor; /* 按钮颜色: 0—3 — 前景色
                           4—7 — 背景色
                           8—11 — 阳边色
                           12—15 — 暗边色 */
extern int _TitleColor; /* 标题前景色 */
extern int _TitleBk; /* 标题背景色 */
extern int _EditColor; /* 编辑框前景色 */
extern int _EditBk; /* 编辑框背景色 */
extern int _TabColor; /* 表格前景色 */
extern int _TabBk; /* 表格背景色 */

```

```
extern int      _BoxLineColor;          /* 画框颜色      */
```

这组全局变量主要供 WINDOWS 风格的界面函数使用,表示各种构件的颜色。其初值分别为

```
_BarColor      : GRAY_BAR(灰色键);
_TitleColor    : WHITE;
_TitleBk       : BLUE;
_EditColor     : BLACK;
_EditBk        : LIGHTGRAY;
_TabColor      : RED;
_TabBk         : WHITE;
_BoxLineColor  : BLACK。

/* ----- 设置界面色彩全局变量的宏 ----- */
#define set_bar_color(c)      (_BarColor=(c))
#define set_edit_color(c)    (_EditColor=(c))
#define set_edit_bk(b)       (_EditBk=(b))
#define set_title_color(c)   (_TitleColor=(c))
#define set_title_bk(b)      (_TitleBk=(b))
#define set_tab_color(c)     (_TabColor=(c))
#define set_tab_bk(b)        (_TabBk=(b))
#define set_mouse_speed(s)   (_MouseSpeed=(s))
void      _Cdecl _Hole      (int x,int y,int width,int high,int color1,int color2,
                             int thickness);
void      _Cdecl _Box      (int x,int y,int width,int high);
void      _Cdecl scroll_h_box(int x,int y,int len,int pagesize,unsigned key,
                             int range,int *n,int *oldpos);
void      _Cdecl scroll_v_box(int x,int y,int len,int pagesize,unsigned key,
                             int range,int *n,int *oldpos);
void      _Cdecl dispinfo   (int col,int line,int width,int high,char *prompt);
int        _Cdecl ask       (int col,int line,int width,int high,char
                             *title,char *prompt);
unsigned   _Cdecl getcode    (int col,int line,int width,int high,char *title,int n,
                             char *odelist[],int *no,int mark);
unsigned   _Cdecl _GetCode   (int col,int line,int width,int high,char *title,int n,
                             void (*getrec)(),int *no);
unsigned   _Cdecl dir_menu   (int col,int line,
int high,char *path,int attrib,int full_width,void (*fun)(),int multiselect);
unsigned   _Cdecl ascii_list (int col,int line,int *no);
void       _Cdecl SP_list    (int col,int line);
unsigned   _Cdecl palette    (int col,int line,char *buffer);
void       _Cdecl prnsr      (int x1,int y1,int x2,int y2,int margin,
                             unsigned palette);
```

```

/* -----
3. 菜单设计
----- */
/* ----- 有关按键式菜单的按键类型的定义 ----- */
typedef struct          /* 按键类型定义 */
{
    int x;               /* 按键左上角相对于菜单的列坐标 */
    int y;               /* 按键左上角相对于菜单的行坐标 */
    int w;               /* 按键宽 */
    int h;               /* 按键高 */
                        /* 以上4个参数的单位均为像素 */
    unsigned butncolor;  /* 正常按键颜色 */
    unsigned pushcolor;  /* 被选中的按键颜色 */
                        /* 以上两个参数均可用按钮颜色宏 */
    unsigned fonttimes;  /* 按键名的放大倍数(第0字节为横
                        向放大倍数,第1字节为纵向放大
                        倍数,如不放大则此参数填257) */
    char *name;          /* 按键名 */
    unsigned key;        /* 按下该按键时菜单函数的返回值 */
    unsigned pressed;    /* 与该菜单按键同功能的键盘键码 */
    int high;            /* 按键厚度(单位为像素,可为0) */
    int lock;            /* 按键是否被锁住 */
    unsigned (*fun)();   /* 按下按键时执行的功能模块指针 */
}BUTTON_TYPE;

/* ----- 有关按键式菜单的结构类型定义 ----- */
typedef struct          /* 按键式菜单类型 */
{
    int col;             /* 菜单左上角列坐标(单位为字节) */
    int line;            /* 菜单左上角行坐标(单位为像素) */
    int width;           /* 菜单宽度(单位为字节) */
    int high;            /* 菜单高度(单位为像素) */
    int arrow_ctrl;      /* 可否用键盘控制菜单 */
    int fixed;           /* 菜单是否固定 */
    int left;            /* 菜单移动范围:左边界(字节) */
    int top;             /* 菜单移动范围:上边界(像素) */
    int right;           /* 菜单移动范围:右边界(字节) */
    int bottom;          /* 菜单移动范围:下边界(像素) */
    unsigned quit_key1;  /* 退出菜单所用键 */
    unsigned quit_key2;  /* 退出菜单所用键 */
    unsigned press_key1; /* 选择菜单项目所用键 */
    unsigned press_key2; /* 选择菜单项目所用键 */
    unsigned forward1;   /* 向前移动当前菜单条目所用键 */
}

```

```

unsigned forward2;          /* 向前移动当前菜单条目所用键 */
unsigned backward1;        /* 向后移动当前菜单条目所用键 */
unsigned backward2;        /* 向后移动当前菜单条目所用键 */
BUTTON_TYPE * buttons;     /* 指向按键表的指针 */
int button_count;          /* 按键个数 */
int pop_up;                /* 菜单是否已经弹出 */
int current;               /* 当前菜单条目的序号 */
int saveimage;             /* 是否存储菜单背景 */
char ** block;             /* 菜单背景存储指针 */
void (* disp_fun)();       /* 显示菜单轮廓图样函数(可为空) */
}BUTTON_MENU;

void _Cdecl DrawButton(BUTTON_TYPE button,int col,int line,int chameleon,
                      int op);

unsigned _Cdecl button_menu(BUTTON_MENU * b,unsigned h);

/* ----- 有关图标式菜单的菜单条目的结构类型定义 ----- */
typedef struct             /* 图标类型 */
{
    int x;                 /* 图标左上角列坐标(单位为像素) */
    int y;                 /* 图标左上角行坐标(单位为像素) */
    int width;             /* 图标宽(单位为像素) */
    int high;              /* 图标高(单位为像素) */
    char * title;          /* 条目标题 */
    unsigned key;          /* 返回值 */
    void (* draw_icon)();  /* 图标绘制函数 */
    unsigned (* fun)();    /* 选中本条目时执行的功能 */
}ICON_TYPE;

/* ----- 有关图标式菜单的结构类型定义 ----- */
typedef struct             /* 图标式菜单类型 */
{
    int x;                 /* 条目标题的列坐标(单位为字节) */
    int y;                 /* 条目标题的行坐标(单位为像素) */
    int width;             /* 条目标题的宽度(单位为字节) */
    int arrow_ctrl;        /* 可否用键盘控制菜单 */
    unsigned forward1;      /* 向前移动当前菜单条目所用键 */
    unsigned forward2;      /* 向前移动当前菜单条目所用键 */
    unsigned backward1;     /* 向后移动当前菜单条目所用键 */
    unsigned backward2;     /* 向后移动当前菜单条目所用键 */
    unsigned press_key1;    /* 选择当前菜单条目所用键 */
    unsigned press_key2;    /* 选择当前菜单条目所用键 */
    int icon_count;        /* 图标数量 */
    ICON_TYPE * icons;     /* 指向图标表的指针 */

```

```

int current;          /* 当前菜单条目的序号      */
int mouse_x;          /* 当前鼠标列坐标      */
int mouse_y;          /* 当前鼠标行坐标      */
}ICON_MENU;

unsigned _Cdecl icon_menu(ICON_MENU * menu);

/* -----
4. 全屏幕数据录入函数

该族函数受以下全局变量的影响:
_Xtimes 和 _Ytimes      —— 字符的横向和纵向放大数;
_EditColor 和 _EditBk    —— 编辑时字符及编辑区背景颜色;
_TextColor 和 _Background —— 退出编辑后字符及编辑区颜色。
该族函数的参数 col 以字节为单位, line 以像素为单位, width 以
字符为单位。
----- */
unsigned _Cdecl getxya(int col, int line, int width, char * string, char * picture);
unsigned _Cdecl getxyb(int col, int line, int * value);
unsigned _Cdecl getxyc(int col, int line, int * value, char * set);
unsigned _Cdecl getxyd(int col, int line, struct date * d);
unsigned _Cdecl getxyf(int col, int line, int width, double * value, int dec, double limit1,
                        double limit2);
unsigned _Cdecl getxyi(int col, int line, int width, int * value, int limit1, int limit2);
unsigned _Cdecl getxyl(int col, int line, int width, long * value, long limit1, long limit2);
unsigned _Cdecl getxyp(int col, int line, int width, char * s);
unsigned _Cdecl getxys(int col, int line, int width, char * string, int * len);
unsigned _Cdecl getxyt(int col, int line, int width, int high, char * string);
/* -----
5. 时钟与定时器
----- */
typedef struct          /* 定时器类型的定义      */
{
    struct date d;      /* 日期                  */
    struct time t;      /* 时间                  */
    int tag;            /* 闹钟激活方式          */
    unsigned (* fun)(); /* 事件函数              */
}ALARM;

/* 定义闹钟激活方式 */
#define PER_SECOND 0 /* 每秒激活 */
#define PER_MINATE 1 /* 每分钟激活 */
#define PER_HOUR 2 /* 每小时激活 */
#define PER_DAY 3 /* 每日激活 */

```

```

#define PER_MONTH      4      /* 每月激活          */
#define PER_YEAR       5      /* 每年激活          */
#define ONCE_ONLY      6      /* 只在指定时间激活一次 */
extern ALARM *_Alarm[];      /* 定时器表          */
extern int _AlarmCount;      /* 定时器个数        */
void _Cdecl set_clock (int col,int line,int xtimes,int ytimes,int color,int bk);
void _Cdecl run_clock (void);
void _Cdecl light_clock(void);
void _Cdecl close_clock(void);
int _Cdecl set_alarm (struct date date,struct time time,int type,unsigned (*fun)());
void _Cdecl del_alarm (int alarm_no);
/* -----

    6. 字符串处理函数
    ----- */

char *_Cdecl space(int len);
char *_Cdecl trim (char *string);
char *_Cdecl ltrim(char *string);
char *_Cdecl skipSPACE(char *string);
/* -----

    7. 表达式与计算器
    ----- */

/* ----- 表达式函数的错误码 ----- */
#define NO_ERROR          0      /* 无语法错误        */
#define NULL_EXPRESSION    201 /* 空表达式          */
#define NONCOMPLETE_EXP    202 /* 表达式不完全      */
#define NO_RIGHT_BRACKET   203 /* 缺少右括号        */
#define HAVE_NOT_ITEM      204 /* 表达式缺项        */
#define NO_RIGHT_QUOTATION 205 /* 缺少右引号        */
#define CONTINUOUS_OPTION  207 /* 连续的运算符      */
#define ITEM_TYPE_WRONG    208 /* 运算对象类型错误  */
#define SQRT_OF_NEGATIVE    209 /* 求负数的平方根    */
#define LOG_OF_NEGATIVE    210 /* 求0和负数的对数  */
#define IS_NOT_FUNCTION    211 /* 函数名拼写错误    */
#define TOO_MUCH_OP         212 /* 运算符过多        */
#define WRONG_NUMERAL      213 /* 数值错误          */
extern int _ResultType;
extern char _ResultString[];
double _Cdecl expression(char *string);
double _Cdecl calculator (int col,int line);

```

# HANENV 系统的库函数

本章收录了 HANENV 系统中 196 个函数的使用方法,供应用程序的设计编程人员参考使用。应该说明的是 HANENV 系统的部分函数如触发器、定时器、菜单、代码表以及汉字输入模块等的用法是比较复杂的,也很难在篇幅有限的条目中作出完整的介绍。因此在阅读本章的有关条目的同时还应该参考前面各章的有关内容,特别是其源程序和应用举例。另外,有关 HANENV 系统中的全局变量的设置和初值、结构类型的定义及其填写规定等内容已经在第 14 章“HANENV 系统的头文件”中作了介绍,因此在本章中遇到这类问题时就不再重复。

### 1. `ascii_list`——查询 ASCII 码表

#### 函数格式:

```
unsigned _Cdecl ascii_list(int col,int line,int *no);
```

功能说明:在屏幕指定位置显示一个代码表式提示窗口,内容为标准和扩展 ASCII 码的码值(包括 10 进制数与 16 进制数)和图象(如图 15-1 所示),供操作人员在屏幕上直接查看。

#### 参数设置:

`col` —— ASCII 码表左上角列坐标,以字节为单位;  
`line` —— ASCII 码表左上角行坐标,以象素为单位;  
`no` —— 被选中的 ASCII 码。

#### 全局变量:

`_TextColor` —— 正文色;  
`_Background` —— 背景色;  
`_TitleColor` —— 标题文字色;  
`_TitleBk` —— 标题背景色;  
`_EditColor` —— 光条文字色;  
`_EditBk` —— 光条色;  
`_BoxLineColor` —— 框的轮廓线色;

\_BarColor      ——— 框色;  
 \_Xtimes        —— 所有字符或汉字的横向放大倍数;  
 \_Ytimes        —— 所有字符或汉字的纵向放大倍数。

**返回值:**退出 ASCII 码表时所使用的统一键盘码。

**操作说明:**

(1)移动光条、翻页、退出等操作见函数 getcode 的操作说明;

(2)如果需要通过参数 no 带出选中的 ASCII 码,可以使用鼠标左键双击表中某条目或者使用回车键来得到光条处的 ASCII 码的码值。

**注 意:**如果实际应用时只需要在屏幕上查看 ASCII 码表,不需要选择码值,则可以分别将全局变量 \_EditColor 和 \_EditBk 的值设置为与全局变量 \_TextColor 和 \_Background 的值相同,使光条消失即可。

**源 程 序:**见程序 11-5。

DEC	HEX	ASCII CODES
0	00H	
1	01H	☐
2	02H	☐
3	03H	♥
4	04H	♦
5	05H	♠
6	06H	♣
7	07H	•

图 15-1 ASCII 码表

## 2. ask——逻辑提问框

**函数格式:**

int ask(int col,int line,int width,int high,char \* title,char \* prompt);

**功能说明:**在屏幕指定位置显示提问框(如图 11-1 所示),然后等待回答。

**参数设置:**

col        —— 提问框左上角列坐标,以字节为单位;  
 line       —— 提问框左上角行坐标,以像素为单位;  
 width      —— 提问框宽度,以字节为单位;  
 high      —— 提问框高度,以像素为单位;



title —— 提问框标题,如无标题则该参数可取空值(NULL);

prompt --- 提示,如无提示则该参数可取空值(NULL)。

#### 全局变量:

\_TextColor —— 提示文字色;

\_Background —— 提示框背景色;

\_TitleColor —— 标题文字色;

\_TitleBk —— 标题背景色;

\_BoxLineColor —— 框的轮廓线色;

\_BarColor —— 框及按钮颜色;

\_Xtimes --- 所有字符或汉字(包括按钮名称)的横向放大倍数;

\_Ytimes —— 所有字符或汉字(包括按钮名称)的纵向放大倍数;

返回值:回答肯定返回 1,回答否定返回 0,按 ESC 键或鼠标右键则返回 -1。

#### 操作说明:

(1)使用鼠标左键直接点选提问框中的两个按钮;

(2)使用键盘键操作,字母 Y 和 y 表示“是”,字母 N 和 n 表示“否”。

注 意:提问框的宽度和高度不宜太小,否则可能容纳不下按钮,尤其在使用了大于 1 的放大倍数之后。具体尺寸可以通过实验确定。

源程序:见程序 11-2。

### 3. \_Bar——画实心矩形

#### 函数格式:

```
void _Bar(int x,int y,int width,int height,int color,unsigned long pattern);
```

功能说明:本函数用于在屏幕上指定位置显示一矩形块。本函数与 \_Block 函数的区别在于本函数允许使用参数 pattern 设置填充图案,以及本函数中的横坐标、矩形的宽度等参数均以像素为单位。

#### 参数设置:

x —— 矩形左上角列坐标,以像素为单位;

y —— 矩形左上角行坐标,以像素为单位;

width —— 矩形的宽度,以像素为单位;

high —— 矩形的高度,以像素为单位;

color ----- 颜色;

pattern —— 矩形的填充类型。其意义为

第四字节:横向填充模式;

第三字节:纵向填充模式;

第二字节:横向移位线;

第一字节:横向移位位数。

注 意:参数 pattern 中的横向填充模式和纵向填充模式均以 8 位,即 1 个字节表示,其中值为 1 的位对应于显示点,为 0 的位不显示。pattern 的第 0 和第 1 两个字节用于表示纵向显示时是否移位和移位位数。移位可以使显示的图案更加富于变化。

源程序:见程序 10-5。

举 例:

```
/* -----
   格纹背景的生成举例
   ----- */

#include <hanenv.h>

main()
{
    init_hanenv();
    _Block(0,0,80,480,LIGHTGRAY);
    _Bar(8,8,624,464,WHITE,0x95950000);
    geth();
    close_hanenv();
}
```

运行结果:将屏幕置成灰底白格背景。

#### 4. \_Block——画实心矩形

函数格式:

```
void _Block(int col,int line,int width,int high,int color);
```

功能说明:本函数用于在屏幕上指定位置显示一矩形块。本函数与 \_Bar 函数的区别在于本函数的横坐标和矩形的宽度均以字节为单位,且本函数的运行速度快于 \_Bar 函数。本函数主要用于清屏幕上的指定区域。

参数设置:

col —— 矩形左上角列坐标,以字节为单位;  
 line —— 矩形左上角行坐标,以像素为单位;  
 width —— 矩形的宽度,以字节为单位;  
 high —— 矩形的高度,以像素为单位;  
 color —— 矩形颜色。

源程序:见程序 10-10。

举 例:见 \_Bar。

#### 5. \_Box——画 WINDOWS 风格的矩形框

函数格式:

```
void _Box(int x,int y,int width,int high);
```

功能说明:本函数用于在屏幕上指定位置显示一个类 WINDOW 风格的矩形框。

参数设置:

x —— 矩形框左上角列坐标,以像素为单位;  
 y —— 矩形框左上角行坐标,以像素为单位;  
 width —— 矩形框的宽度,以像素为单位;

high —— 矩形框的高度,以像素为单位。

**全局变量:**

\_Background —— 框内背景色;  
\_BoxLineColor —— 框的轮廓线色;  
\_BarColor —— 其中背景色决定框的颜色。

源程序:见程序 10-10。

## 6. button\_menu——多功能按键式菜单

**函数格式:**

```
unsigned _Cdecl button_menu(BUTTON_MENU * b,unsigned key);
```

**功能说明:**button\_menu 是一个功能很强的函数,可以用来构造各种按钮式、光条式以及下拉式菜单。这些菜单既可以使用鼠标驱动,也可以使用键盘选择条目。而且如果有必要的话,整个菜单还可以在屏幕上移动。

**参数设置:**

b —— 菜单参数。在头文件 hanenv.h 中有关于结构类型 BUTTON\_MENU 的定义和详细的填写说明,请参看第 14 章;

key —— 驱动菜单函数所用的键盘码或鼠标按钮。

**返回值:**在菜单中选择某条目时该条目对应按键的键码,或退出菜单时的退出键码。

**注意:**该函数的工作原理、参数设置和使用方法举例请参看 13.1:“多功能按键式菜单及其应用”。

源程序:见程序 13-1。

## 7. calcutor——弹出式计算器

**函数格式:**

```
double calcutor(int col,int line);
```

**功能说明:**本函数实现了一个弹出式计算器(如图 13-2 所示)。

**参数设置:**

col —— 矩形左上角列坐标,以字节为单位;

line —— 矩形左上角行坐标,以像素为单位;

**返回值:**计算结果值。

**操作说明:**

(1)使用鼠标点选计算器的按钮可以实现计算功能;

(2)可以使用与计算器各按钮同名的键盘按键直接操作计算器。部分计算器按钮与键盘码的对应关系为

清除键(红色 C) —— 回车键  
sin (正弦) —— 功能键 F2  
cos (余弦) —— 功能键 F3  
tan (正切) —— 功能键 F4  
atg (反正切) —— 功能键 F5

ln (自然对数)	—— 功能键 F6
lg (常用对数)	—— 功能键 F7
exp (指数函数)	—— 功能键 F8
pow (幂函数)	—— 功能键 F9
sqr (平方根)	—— 功能键 F10
abs (绝对值)	—— 功能键 Shift+F2
max (最大值)	—— 功能键 Shift+F3
min (最小值)	—— 功能键 Shift+F4
int (取整)	—— 功能键 Shift+F5
rnd (舍入)	—— 功能键 Shift+F6
mod (求模)	—— 功能键 Shift+F7
hex (16 进制)	—— 功能键 Shift+F8
pi( $\pi$ )	—— 功能键 Shift+F9

当然也可以直接使用字母键逐个输入上述函数名;

(3)将鼠标光标移至计算器上按钮以外的地方并按下鼠标左键可以拖动整个计算器在屏幕上移动位置。

**注 意:**关于计算器函数可以计算的表达式类型以及错误信息表请参阅函数 expression 的说明。

**源 程 序:**部分源程序见程序 13-3。

**举 例:**参看 geth。

## 8. call\_DOS——调用 DOS shell

**函数格式:**

```
int call_DOS(char * envname);
```

**功能说明:**本函数用于在程序中临时返回 DOS 提示行。调用本函数后,首先保存当前屏幕,然后回到西文状态,恢复原西文状态的屏幕显示,调用 DOS shell,并显示字符串“Press EXIT to return < 环境名 > ...”。此时可以使用 DOS 中的各种命令。如果键入 EXIT< 回车 > 则恢复原屏幕,返回原程序。

**参数设置:**

envname——环境名,西文字符串

**返 回 值:**如果未申请到足够大的扩充内存空间(此时无法从 DOS 状态恢复应用程序的屏幕显示内容)返回零,否则返回非零值。

**源 程 序:**见程序 5-4。

## 9. change\_color——改变省缺的 16 种颜色设置

**函数格式:**

```
void chang_color(int color, int new_color);
```

**功能说明:**改变 VGA 图形模式 12H 的颜色选择寄存器之值与 DAC 颜色寄存器号的对应关系。例如,在 VGA 图形模式 12H 的初始设置中,屏幕背景色被设置为调色板的第 0 号。

寄存器(黑色)。如果要改为其他颜色,就可以使用 `change_color` 完成。

**参数设置:**

`color` —— 颜色号(0—5);  
`new_color`——调色板寄存器号(02—55);

**源程序:**见程序 1-3。

## 10. `_ChangeXMS`——修改扩展存储器存储块大小

**函数格式:**

```
int _ChangeXMS(int handle,int size);
```

**相关函数:**

```
void init_XMS(void);
int _GetXMS(int size);
void _FreeXMS(int handle);
void _MoveDataXMS(void);
int _LoadXMS(FILE *file,long filelen);
int sizeofXMS(void);
int largestXMS(void);
int haveXMS(void);
```

**功能说明:**该组函数用于在扩充存储器规范 XMS 支持下使用扩充存储器。各函数的具体功能为:

<code>init_XMS</code>	——用于初始化 XMS 的驱动程序,应该出现在所有对 XMS 族其他函数的引用之前;
<code>haveXMS</code>	—— XMS 驱动程序是否安装成功;
<code>_GetXMS</code>	——申请一块指定大小的扩充存储块;
<code>_FreeXMS</code>	——释放一块已经申请扩充存储块;
<code>_ChangeXMS</code>	——改变一块已经申请好的扩充存储块的大小;
<code>_MoveDataXMS</code>	——在常规内存与扩充内存之间移动数据;
<code>_LoadXMS</code>	——将数据文件装入扩充存储器;
<code>sizeofXMS</code>	——求扩充存储器剩余存储容量;
<code>largestXMS</code>	——求扩充存储器剩余最大存储块的大小。

**参数设置:**

`handle` —— 扩充内存块的把柄;  
`size` —— 扩充内存块的大小,单位为 K 字节。  
`file` —— 准备装入扩充存储器中的文件的文件名;  
`filelen` —— 文件长度。

**全局变量:**

`_ErrorNo` —— 错误类型。

**返回值:**

`haveXMS` —— 如果 XMS 驱动程序安装成功返回非零值,否则返回零;

\_GetXMS —— 扩充内存块的把柄;  
 \_ChangeXMS —— 扩充内存块的把柄;  
 \_LoadXMS —— 扩充内存块的把柄;  
 sizeofXMS —— 扩充存储器剩余存储容量(单位为 K);  
 largestXMS —— 扩充存储器剩余最大存储块的大小(单位为 K)。

**注 意:** 这组函数的应用前提是计算机中已经安装了 HIMEM.SYS。

**参 阅:** 函数 \_GetEMMsize。

**源 程 序:** 函数 sizeofXMS、largestXMS 和 haveXMS 实际上是定义于 hanenv.h 中的宏,其定义请参看第 14 章。其他函数的源程序见 4.3 节“利用扩充存储器管理规范 XMS 访问扩充存储器”。

### 11. char \_h\_line——用扩展 ASCII 码画横线

**函数格式:**

```
void char _h_line(int col,int line,int len,int type);
```

**相关函数:**

```
void char _v_line(int col,int line,int len,int type);
```

```
void char _rectangle(int col,int line,int width,int high,int type);
```

**功能说明:** 这三个函数使用扩展 ASCII 码中的制表符号画线和矩形框

**参数设置:**

col, line —— 线左(上)端坐标,col 的单位为字符;  
 width, high —— 矩形框的宽和高,单位为字符;  
 len —— 线长,以字符为单位;  
 type —— 单线还是双线。可以使用如下符号常数:  
         SINGLE: 单线;  
         DOUBLE: 双线。

**全局变量:**

\_TextColor —— 画线颜色;  
 \_Background —— 画线背景色;  
 \_Xtimes —— 画线字符的横向放大倍数;  
 \_Ytimes —— 画线字符的纵向放大倍数。

**源 程 序:** 程序 10-6、程序 10-7 和程序 10-8。

### 12. char \_rectangle——用扩展 ASCII 码画矩形框

**函数格式:**

```
void char _rectangle(int col,int line,int width,int high,int type);
```

**功能说明:** 参看 char \_h\_line。

### 13. char \_v\_line——用扩展 ASCII 码画竖线

**函数格式:**

```
void char_v_line(int col,int line,int len,int type);
```

功能说明:参看 char\_h\_line。

#### 14. close\_clock——关闭时钟的显示

函数格式:

```
void close_clock(void);
```

相关函数:

```
void light_clock(void);
```

功能说明:在使用函数 run\_clock 安装了时钟之后,可以使用这两个函数关闭和打开时钟的显示。

源程序:见程序 9-6。

参 阅:程序 run\_clock。

#### 15. close\_display——关闭屏幕显示

函数格式:

```
void close_display(void);
```

相关函数:

```
void open_display(void);
```

功能说明:这两个函数分别用于关闭和打开屏幕显示开关。

源程序:见程序 1-5。

#### 16. close\_hanenv——结束 HANENV 系统的操作

函数格式:

```
void close_hanenv(void);
```

相关函数:

```
void init_hanenv(void);
```

功能说明: init\_hanenv 函数的功能主要有:为 HANENV 系统的部分全局变量赋初值、保存原西文屏幕信息、设置 VGA 显示模式 12H 以及探测并安装鼠标驱动程序等。close\_hanenv 函数的功能为恢复原显示模式和屏幕信息等。这两个函数应该配对使用,分别用于 HANENV 系统的启动与退出。HANENV 中的所有其他函数均应在这两个本函数之间使用。如果想同时使用 HANENV 系统和 Turbo C 的图形库,则应先初始化 HANENV 系统,然后再初始化图形模块。

举 例:见 Bar。

源程序:见程序 5-2、程序 5-3。

#### 17. del\_alarm——删除定时器事件

函数格式:

```
void del_alarm(int alarm_no);
```

**功能说明:**删除定时器事件。

**参数设置:**

alarm\_no——事件号码。

**源程序:**见程序 9-6。

**参 阅:**函数 set\_alarm。

## 18. delightcursor——关闭光标显示

**函数格式:**

void delightcursor(void);

**相关函数:**

int iscursorlight(void);

void lightcursor(void);

void movecursor(int col,int line);

**功能说明:**这组函数用于控制 HANENV 系统提供的正文闪烁光标。各函数的功能分别为

iscursorlight —— 测试正文光标是否打开;

delightcursor —— 用于关闭正文光标的显示;

lightcursor —— 用于打开正文光标的显示;

movecursor —— 将正文光标移到指定坐标处。

**参数设置:**

col —— 正文光标的列坐标,以字节为单位;

line —— 正文光标的行坐标,以像素为单位。

**返回值:**如果光标打开则函数 iscursorlight 返回非零值,否则返回零。

**参 阅:**设置光标参数函数 setcursor。

**源程序:**delightcursor 和 lightcursor 的源程序见程序 9-2。其余两个函数实际上是定义于头文件 hanenv.h 中的宏,见第 14 章。

## 19. delight\_mouse——关闭鼠标光标

**函数格式:**

void delight\_mouse(void);

**相关函数:**

void light\_mouse(void);

**功能说明:**这两个函数分别用于打开和关闭鼠标光标的显示。

**参 阅:**函数 get\_mouse\_range、set\_mouse...以及 till\_mouse\_pop 的说明。

**源程序:**见程序 2-6、程序 2-7。

## 20. dir\_menu——文件目录选择菜单

**函数格式:**

unsigned dir\_menu(int col,int line,int high,char \* path,int attrib,int full\_width,



```
void (*fun)(),int multiselect);
```

**功能说明:**弹出式文件目录选择菜单。利用该函数可以在屏幕上弹出一个代码表式的文件目录选择菜单(见图 11-6),通过鼠标或键盘选择其中一个或多个文件进行处理。

**参数设置:**

col;     —— 菜单左上角列坐标(单位为字节);  
 line;    —— 菜单左上角行坐标(单位为像素);  
 high;    —— 菜单高度(单位为正文行);  
 path     —— 备选文件路径(其中可使用通配符“\*”和“?”);  
 attrib   —— 备选文件属性,可以是定义于 Turbo C 的头文件 dos.h 中的下列符号常数:

FA\_RDONLY:只读文件  
 FA\_HIDDEN:隐藏文件  
 FA\_SYSTEM:系统文件  
 FA\_LABEL:卷标  
 FA\_DIRECT:目录  
 FA\_ARCHIVE:档案

或者是它们的组合,例如

FA\_RDONLY|FA\_HIDDEN:隐藏的只读文件

如果填写 0,表示对以上属性均不选,即只列出普通的文件的目录。

full\_width—— 如果该项填写 NO,则菜单条目中仅包含文件名和后缀;否按全宽度显示菜单条目,即菜单条目中还有文件长度、更新日期和时间;  
 fun         —— 指向处理被选中的文件的函数指针。该参数必须指向一个用户自己编写的文件处理函数。其格式应该为

```
void processfiles(char * filename)
{
}
```

multiselect —— 注意由函数 dir\_menu 提供的文件名中已带有路径。如果允许多选(见下条),函数 dir\_menu 在结束文件选择之后会自动调用该函数逐个处理被选中的文件,而第一个被处理的文件就是结束选择时所选定的那个文件。如果不允许多选,则 dir\_menu 调用上述用户自定义函数直接处理被选中的文件;是否允许一次选择多个文件。在允许同时选择多个文件的情况下,可以使用鼠标左键单击相应文件条目或使用空格键在当前文件条目之前做一个星号标记(或者取消星号标记),在使用鼠标左键双击最后一个选中的文件条目后由预先指定的用户自定义函数(见上条)对所有被选中的文件进行处理。

**全局变量:**

\_TextColor   —— 正文色;

_Background	—— 背景色;
_TitleColor	—— 标题文字色;
_TitleBk	—— 标题背景色;
_EditColor	—— 光条文字色;
_EditBk	—— 光条色;
_BoxLineColor	—— 框的轮廓线色;
_BarColor	—— 框色;
_Xtimes	—— 所有字符或汉字的横向放大倍数;
_Ytimes	—— 所有字符或汉字的纵向放大倍数;

操作说明: 见函数 `getcode` 的操作说明。

源程序: 见程序 11-8。

## 21. disableMKB —— 禁止虚拟键盘

函数格式:

```
void disableMKB(void);
```

相关函数:

```
void enableMKB(void);
```

**功能说明:** 这两个函数分别用于禁止或允许使用鼠标操纵的虚拟小键盘。如果应用程序已经使用 `load_MKB` 函数安装了虚拟键盘, 则函数 `disableMKB` 禁止用热键或鼠标激活虚拟键盘, 而函数 `enableMKB` 则恢复允许使用虚拟键盘。通常这两个函数用于临时性禁止使用虚拟键盘, 以避免与其它屏幕显示信息冲突。

**参阅:** `free_mouse_KB` 和 `set_mouse_KB` 函数的说明。

## 22. dispinfo —— 提示信息显示

函数格式:

```
void dispinfo(int col, int line, int width, int high, char * prompt);
```

**功能说明:** 在屏幕上指定弹出一个提示框, 框中有预先设定的说明信息。待用户按下键盘上任一键或鼠标按钮时即退出提示, 自动恢复原来的屏幕背景。

**参数设置:**

col	—— 提示框左上角列坐标(以字节为单位);
line	—— 提示框左上角行坐标(以像素为单位);
width	—— 提示框宽度(以字节为单位);
high	—— 提示框高度(以像素为单位);
prompt	—— 提示信息。

**全局变量:**

_TextColor	—— 正文色;
_Background	—— 背景色;
_BoxLineColor	—— 提示框的轮廓线色;
_BarColor	—— 框色;

`_Xtimes`       —— 字符或汉字的横向放大倍数;  
`_Ytimes`       —— 字符或汉字的纵向放大倍数;

**操作说明:**按下任一键盘按键或鼠标按钮即可退出本函数并恢复原来的屏幕背景。

**源程序:**见程序 11-1。

### 23. `double _press` —— 测试鼠标按钮是否双击

**函数格式:**

```
int double _press(unsigned key);
```

**功能说明:**该函数用于测试是否双击某鼠标按钮。

**参数设置:**

`key` —— 欲测试的鼠标按钮。可以使用如下符号常数:

`LEFT_BUTTON` : 测试是否双击鼠标左键;

`RIGHT_BUTTON` : 测试是否双击鼠标右键。

**返回值:**如果是双击鼠标某按钮则返回非零值,否则返回零。

**应用举例:**

```
/* -----
   测试鼠标双击的程序片断
   ----- */
.....
do
{
    key = gethan();
    if(key == LEFT_BUTTON && double_press(LEFT_BUTTON))
        dosomething();
}while(key != KEY_ESC);
```

**参 阅:**函数 `set_double_press_time`。

### 24. `_DrawArc` —— 多功能画弧函数

**函数格式:**

```
void _DrawArc(int col,int row,int a,int b,int t1,int t2,int color,int fill);
```

**功能说明:**该函数可以用来画圆、椭圆以及圆弧和椭圆弧,以及空心或实心的扇形,非常适合绘制用于数据统计的饼图和立体饼图。

**参数设置:**

`col`     —— 圆心横坐标(以像素为单位);

`row`     —— 圆心纵坐标(以像素为单位);

`a`       —— 椭圆横半径(以像素为单位);

`b`       —— 椭圆纵半径(以像素为单位);

`t1`      —— 弧起始角度(以度为单位);

`t2`      —— 弧终止角度(以度为单位);

color —— 颜色;  
 fill —— 填充模式: 可以使用下列定义于头文件 hanenv.h 中的符号常数:  
     ONLY\_ARC —— 只画弧线;  
     DRAW\_SECTOR —— 画空心扇形;  
     FILL\_SECTOR —— 画实心扇形。

源程序: 见程序 10-12。

## 25. draw\_arc —— 画圆弧

函数格式:

```
void draw_arc(int x,int y,int r,int t1,int t2,int color);
```

相关函数:

```
void draw_circle(int x,int y,int r,int color);  

void draw_pie(int x,int y,int r,int color);  

void draw_fan(int x,int y,int r,double t1,double t2,int color);  

void draw_ellipse(int x,int y,int a,int b,int color);
```

功能说明: 这组函数用于绘制曲线。函数 draw\_circle 用于画圆; 函数 draw\_pie 用于画实心圆; 函数 draw\_fan 用于画扇型, 而函数 draw\_ellipse 用于画椭圆。

参数设置:

x —— 横坐标;  
 y —— 纵坐标;  
 r —— 半径;  
 color —— 颜色;  
 t1 —— 弧的起始角度;  
 t2 —— 弧的终止角度;  
 a —— 椭圆横轴半径;  
 b —— 椭圆纵轴半径。

源程序: 实际上, 这是一组用于简化对函数 DrawArc 的调用的宏, 其定义可参看头文件 hanenv.h。

## 26. \_DrawButton —— 在指定坐标画按钮

函数格式:

```
void _DrawButton(BUTTON_TYPE * button,int col,int line,int chameleon,  

                 int op);
```

功能说明: 该函数用于在屏幕上的指定位置上显示一个立体的按钮。

参数设置:

button —— 按钮参数。在头文件 hanenv.h 中有结构类型 BUTTON\_TYPE 的定义和详细说明, 请参看第 14 章的有关部分;

- col —— 菜单左上角列坐标(以字节为单位)。该参数和下一个参数都用于为所画按钮提供一个相对坐标。如果不是在菜单中绘制按钮,也可以使用绝对坐标(即本参数和下一个参数均取值0);
- line —— 菜单左上角行坐标(以像素为单位)。
- chameleon —— 按下按钮时是否改变颜色(使用结构 `BUTTON_TYPE` 中的分量 `pushbutton` 所指定的颜色);
- op —— 按钮状态(0=弹起,1=按下)。

#### 全局变量:

`_BoxLineColor` —— 按钮轮廓线色;

源程序:见程序10-11。

### 27. `draw_circle` —— 画圆弧

#### 函数格式:

```
void draw_circle(int x,int y,int r,int color);
```

功能说明:参看函数 `draw_arc`。

### 28. `draw_ellipse` —— 画椭圆

#### 函数格式:

```
void draw_ellipse(int x,int y,int a,int b,int color);
```

功能说明:参看函数 `draw_arc`。

### 29. `_DrawF` —— 显示字模点阵

#### 函数格式:

```
void _DrawF(int x,int y,int width,int high,int color,unsigned char *font);
```

#### 相关函数:

```
void _DrawFont(int x,int y,int width,int high,int color,unsigned char *font);
```

```
void _OutF(int col,int line,int width,int high,int color,unsigned char *font);
```

```
void _OutFont(int col,int line,int width,int high,int color,unsigned char *font);
```

功能说明:这四个函数均用于在屏幕上指定位置处显示一个字模点阵。函数 `_DrawF` 和 `_DrawFont` 使用逐点输出法,速度稍慢,但对显示位置没有限制。而函数 `_OutF` 和 `_OutFont` 使用字节输出法,速度快,但显示位置中的横坐标必须以字节为单位,即取8的倍数。函数 `_DrawFont` 和 `_OutFont` 还可以设定显示字模时的放大倍数。这四数是 HANENV 系统的基本显示函数,但通常显示汉字、字符或字符串总是使用其他显示函数。只有需要直接显示存放在某数组中的点阵时,才使用这些函数。

#### 参数设置:

x,y —— 使用逐点输出法显示字模点阵时的位置参数。x 为横坐标, y 为纵坐标,单位均为像素;

col,line —— 使用字节输出法显示字模点阵时的位置参数。col 为显示列,单位为字节,line 为显示行,单位为线;

width —— 字模点阵的宽度(以字节为单位);  
high —— 字模点阵的高度(以象素为单位);  
color —— 颜色;  
font; —— 字模点阵的存放地址。

**全局变量:**使用函数 DrawFont 和 OutFont 时,可以使用全局变量:

\_Xtimes —— 字模点阵的横向放大倍数;  
\_Ytimes —— 字模点阵的纵向放大倍数;

**源程序:**见程序 6-4、程序 6-8、程序 6-9、程序 6-10。

### 30. draw\_fan —— 画扇形

**函数格式:**

```
void draw_fan(int x,int y,int r,double t1,double t2,int color);
```

**功能说明:**参看函数 draw\_arc。

### 31. \_DrawFont —— 显示字模点阵

**函数格式:**

```
void _DrawFont(int x,int y,int width,int high,int color,unsigned char * font);
```

**功能说明:**见函数 DrawF。

### 32. draw\_line —— 画直线

**函数格式:**

```
void draw_line(int x1,int y1,int x2,int y2,int color);
```

**功能说明:**画任意角度的直线。

**参数设置:**

x1 —— 直线始端横坐标;  
y1 —— 直线始端纵坐标;  
x2 —— 直线末端横坐标;  
y2 —— 直线末端纵坐标;  
color —— 颜色。

**源程序:**见程序 10-11。

### 33. draw\_pie —— 画实心圆

**函数格式:**

```
void draw_pie(int x,int y,int r,int color);
```

**功能说明:**参看函数 draw\_arc。

### 34. draw\_rectangle —— 画矩形

**函数格式:**

```
void draw _ rectangle(int x,int y,int width,int high, int color,unsigned char
pattern);
```

**功能说明:**在屏幕上指定位置处画一个矩形。

**参数设置:**

x       —— 矩形左上角横坐标;  
y       —— 矩形左上角纵坐标;  
width   —— 矩形宽;  
high    —— 矩形高;  
colorb   —— 颜色;  
pattern —— 线型,参看函数\_H\_Line 的参数说明。

**源程序:**见程序 10-4。

## 35—42. drawxy ... —— 汉字显示函数族

**函数格式:**

```
void drawxyc(int x,int y,int color,int ch);
void drawxychar(int x,int y,int color,int ch);
void drawxyh(int x,int y,int color,unsigned h);
void drawxyhan(int x,int y,int color,unsigned h);
void drawxys(int x,int y,int color,char *s);
void drawxystr(int x,int y,int color,char *s);
void drawxyt(int x,int y,int color,int ch);
void drawxytiny(int x,int y,int color,char *s);
```

**功能说明:** draw ... 函数族调用逐点显示字模函数\_DrawF 和\_DrawFont,用于显示字符、汉字或字符串。draw ... 函数族的特点是显示位置任意,但显示速度稍慢。各函数的具体功能说明如下:

drawxyc     —— 显示 ASCII 码字符(包括扩展 ASCII 字符);  
drawxychar —— 显示 ASCII 码字符,并可设置放大倍数;  
drawxyh     —— 显示汉字或全角字符;  
drawxyhan   —— 显示汉字或全角字符,并可设置放大倍数;  
drawxys     —— 显示字符串;  
drawxystr   —— 显示字符串,并可设置放大倍数;  
drawxyt     —— 显示小型 ASCII 码字符(点阵尺寸为 6×8);  
drawxytiny   —— 显示小型 ASCII 码组成的字符串。

**参数设置:**

x, y       —— 汉字或字符点阵左上角坐标;  
color       —— 汉字或字符的颜色;  
ch          —— 待显示字符;  
h          —— 待显示汉字的机内码;

s —— 待显示的字符串。

#### 全局变量:

\_Xtimes —— 汉字或字符的横向放大倍数;

\_Ytimes —— 汉字或字符的纵向放大倍数;

参 阅: DrawF 函数、outxy...函数族和 putxy...函数族。

源 程 序: drawxyc、drawxychar、drawxyh、drawxyhan 以及 drawxyt 是定义于头文件 hanenv.h 中的宏,可参看第14章。函数 drawxys 和 drawxytiny 的源程序见程序 6-7 和程序 6-6。

### 43. enableMKB —— 允许虚拟键盘

#### 函数格式:

```
void enableMKB(void);
```

功能说明:参阅 disableMKB。

### 44. expression —— 计算表达式

#### 函数格式:

```
double expression(char *s);
```

功能说明:本函数用于计算表达式的值。在表达式中可以使用以下项目:

- 1) 数据,包括十进制数据和十六进制数据;
- 2) 运算符和括号,包括加(+)、减(-)、乘(\*)、除(/)以及左右括号;
- 3) 函数,可以使用的函数有:

ABS —— 绝对值,格式为 abs(x),求  $|x|$ ;

SQR —— 平方根,格式为 sqrt(x),求  $\sqrt{x}$ ;

EXP —— 指数,格式为 exp(x),求  $e^x$ ;

POW —— 乘方,格式为 pow(x,n),求  $x^n$ ;

LG —— 常用对数,格式为 lg(x),求  $\lg x$ ;

LN —— 自然对数,格式为 ln(x),求  $\ln x$ ;

SIN —— 正弦,格式为 sin(x),求  $\sin x$ ;

COS —— 余弦,格式为 cos(x),求  $\cos x$ ;

TAN —— 正切,格式为 tan(x),求  $\tan x$ ;

ATAN —— 反正切,格式为 atan(x1,x2),求  $\arctan(x1/x2)$ ;

MAX —— 求大,格式为 max(x1,x2),求  $x1, x2$  中较大者;

MIN —— 求小,格式为 min(x1,x2),求  $x1, x2$  中较小者;

INT —— 取整,格式为 int(x),求  $x$  的整数部分;

ROUND —— 舍入,格式为 round(x,n),对  $x$  保留  $n$  位小数做舍入;

MOD —— 求余,格式为 mod(x,y),求  $x/y$  的余数;

HEX —— 将十进制数据转换为十六进制数据。

#### 参数设置:



s —— 存放表达式的字符串。

**返回值:** 计算结果。

**全局变量:**

_ResultString	—— 如果计算结果是十六进制, 则以字符串的形式表示;
_ResultType	—— 如果计算结果是十六进制, 则该全局变量的值为 'C' ;
_ErrorNo	—— 错误码。计算表达式函数的错误码为
NO_ERROR	(无语法错误)
NULL_EXPRESSION	(空表达式)
NONCOMPLETE_EXP	(表达式不完全)
NO_RIGHT_BRACKET	(缺少右括号)
HAVE_NOT_ITEM	(表达式缺项)
NO_RIGHT_QUOTATION	(缺少右引号)
CONTINUOUS_OPTION	(连续的运算符)
ITEM_TYPE_WRONG	(运算对象类型错误)
SQRT_OF_NEGATIVE	(求负数的平方根)
LOG_OF_NEGATIVE	(求 0 和负数的对数)
IS_NOT_FUNCTION	(函数名拼写错误)
TOO_MUCH_OP	(运算符过多)
WRONG_NUMERAL	(数值错误)。

#### 45. free\_han\_mode —— 释放汉字输入法

**函数格式:**

```
void free_han_mode(int altfn);
```

**相关函数:**

```
int set_han_mode(int altfn, char * modename, unsigned (* get_han)(),
char * codefile, int wherecodes);
void set_asc_mode(void);
```

**功能说明:** 函数 set\_han\_mode 用于安装某个汉字输入法模块; free\_han\_mode 用于卸去某汉字输入法模块, 并释放其占用的系统资源; set\_asc\_mode 函数用于安装全角 ASCII 码输入模块。

**参数设置:**

altfn	—— 该输入法使用的切换键。可以使用定义于头文件 hanenv.h 中的符号常数 ALT_F1 ~ ALT_F10, 代表功能键 Alt+Fn;
modename	—— 该输入法的名称, 最好由 4 个汉字组成;
get_han	—— 指向该输入法操作函数的指针, 通常可以填写输入法操作函数的函数名。HANENV 系统现有的输入法操作函数有:
_GetSP;	国标区位输入法;
_GetAscii;	英文数字输入法;

<code>_GetFigure:</code>	图形符号输入法(以上为基本输入法,由函数 <code>init_mode</code> 直接安装);
<code>_GetPY:</code>	拼音输入法,包括全拼和双拼拼音(以上各输入法均无需码表文件);
<code>_GetNewCode:</code>	通用输入法。通过选配不同内容的码表文件可以实现各种汉字和词组输入法。HANENV 系统目前配有五笔字型输入法的码表文件 <code>wbx.cod</code> 。
<code>codefile</code>	——该输入法使用的码表文件的文件名。如果该输入法不使用码表文件,该参数应置为空;
<code>wherecodes</code>	——该输入法的码表装载何处。可以使用定义于 <code>hanenv.h</code> 中的符号常数: XMS: 扩充存储器(使用 XMS 规范); EMM: 扩充存储器(使用中断 INT15H); DSK: 硬盘上; MEM: 内存中(只限于不用码表的输入法)。

**返回值:** 如果输入法安装成功,返回非零值,否则返回零。

**源程序:** 见程序 8-7、程序 8-8 和程序 8-9。

**参 阅:** 函数 `gethan` 的说明。

#### 46. `free_HZK` —— 汉字库卸载

**函数格式:**

```
HZK *free_HZK(HZK *hzk);
```

**相关函数:**

```
HZK *load_HZK(int width,int high,char *hzkname,unsigned char *codelist,
int wherefont);
```

**功能说明:** 这两个函数分别用于装载和卸载当前汉字库。函数 `load_HZK` 用于安装指定参数的汉字库作为当前汉字库,而函数 `free_HZK` 用于释放某汉字库所占用的常规和扩充内存资源。

**参数设置:**

<code>hzk</code>	—— 被卸载的汉字库结构变量(函数 <code>free_HZK</code> 使用);
<code>width</code>	—— 字模点阵宽,以字节为单位;
<code>high</code>	—— 字模点阵高;
<code>hzkname</code>	—— 汉字库文件名;
<code>codelist</code>	—— 如果是小字库,则指向小字库索引,否则应置为空。
<code>whilefont</code>	—— 汉字库安装在何处。可以使用下列定义于头文件 <code>hanenv.h</code> 中的符号常数: DSK: 汉字库在硬盘中; EMM: 汉字库在扩充内存中;

XMS: 汉字库在 XMS 扩充内存中;

MEM: 小字库在内存中。

**返回值:**函数 free\_HZK 返回一空指针,而函数 load\_HZK 返回一指向汉字库结构变量的指针。

**源程序:**见程序 7-1 和程序 7-2。

#### 47. free\_MKB —— 释放虚拟键盘

**函数格式:**

```
void free_MKB(void);
```

**相关函数:**

```
void load_MKB(unsigned key);
```

**功能说明:**这两个函数分别用于安装和释放虚拟键盘。函数 load\_MKB 用于安装一个屏幕小键盘(见图 15-2),该虚拟键盘可以使用鼠标操作,并可以在屏幕上随意移动位置。free\_MKB 函数用于释放虚拟键盘占用的系统资源。

**参数设置:**

key —— 弹出虚拟键盘所用的热键键码。通常可使用鼠标点选提示行上的键盘标记来激活虚拟键盘,但也可以使用本参数另外再设置一个热键(通常使用功能键 KEY\_F1—KEY\_F10、KEY\_Ctr\_F1—KEY\_Ctr\_F10)。

**操作说明:**

- 1) 按下热键或使用鼠标左键点选提示行上的“键盘”标记可以调出或撤消虚拟键盘;
- 2) 使用鼠标左键点选小键盘中的某个按键的结果相当于直接按下键盘上的对应按键;
- 3) 将鼠标光标移至小键盘上除按键以外的地方并按下鼠标左键可以拖动虚拟键盘在屏幕上移动。

**参 阅:**函数 disableMKB 和函数 set\_mouse\_KB 的说明。

#### 48. \_FreeXMS —— 释放 XMS 块

**函数格式:**

```
int _FreeXMS(int handle);
```

**功能说明:**参看函数\_ChangeXMS 的说明。

#### 49. \_GetAscii —— 西文字符输入法模块

**功能说明:**参看函数 free\_han\_mode 的说明。

#### 50. getblock —— 存储屏幕矩形区域

**函数格式:**

```
char * * getblock(int col,int line,int width,int high);
```

**相关函数:**

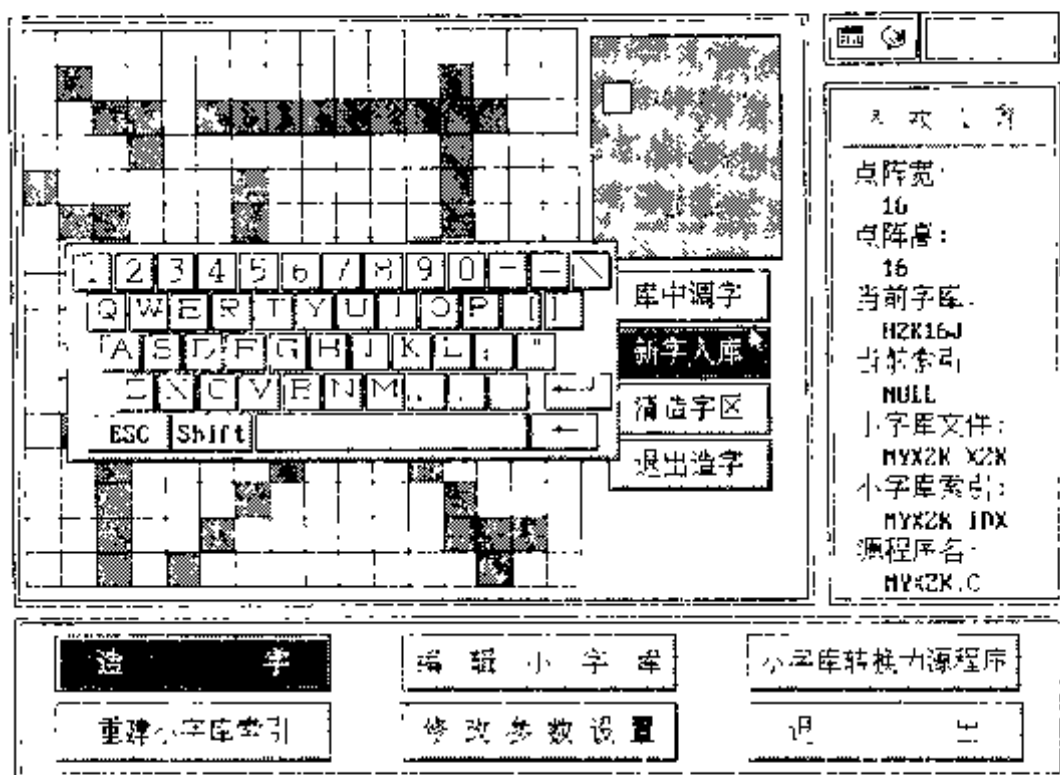


图 15-2 虚拟键盘

```
void putblock(int col,int line,int width,int high,char * * block);
```

**功能说明:**这两个函数用于保存和恢复屏幕上的矩形区域。函数 getblock 申请一块常规内存,并拷贝屏幕上的指定矩形区域的内容;函数 putblock 恢复保存在内存块中的屏幕图象,并释放所申请的常规内存块。保存和恢复屏幕图形块的坐标可以不同,因此利用这两个函数还可以实现屏幕图象块的移动。申请的最大图象块的尺寸不能超过 256K(640×480 大小的屏幕块即需占用 256K 的存储器)。

#### 参数设置:

col,line —— 屏幕上矩形区域左上角的坐标,其中 col 的单位为字节;  
width —— 屏幕上矩形区域的宽度,单位为字节;  
high —— 屏幕上矩形区域的高度;  
block —— 指向内存中存储块的指针。

**返回值:**函数 getblock 的返回值为内存中存储块的地址。

**参 阅:**函数 getblockXMS 的说明。

**源 程 序:**见程序 1-5、程序 1-8。

### 51. getblockXMS —— 存储屏幕矩形区域至扩展存储器

#### 函数格式:

```
int getblockXMS(int left,int top,int right,int bottom);
```

#### 相关函数:

```
void putblockXMS(int left,int top,int right,int bottom,int handle);
```

**功能说明:**这两个函数用于保存和恢复屏幕上的矩形区域。函数 getblockXMS 申请一块扩充内存,并拷贝屏幕上的指定矩形区域的内容;函数 putblockXMS 恢复保存在内存块

中的屏幕图象,并释放所申请的扩充内存块。由于使用了扩充内存,所以这两个函数的速度慢于函数 `getblock` 和 `putblock`。但这两个函数不占用常规内存,屏幕图象块的大小也不受限制。

**参数设置:**

`left` —— 屏幕矩形区域左边界(单位为字节);  
`top` —— 屏幕矩形区域上边界;  
`right` —— 屏幕矩形区域右边界(单位为字节);  
`bottom` —— 屏幕矩形区域下边界;  
`handle` —— 扩充内存块把柄。

**返回值:**函数 `getblockXMS` 的返回值为扩充内存块把柄。

**参 阅:**函数 `getblock` 的说明。

**注 意:**在设置参数时必须使矩形区域的宽度为偶数字节。

**源 程 序:**见程序 4-12、程序 4-13。

## 52. `getcode` —— 代码表

**函数格式:**

```
unsigned getcode(int col,int line,int width,int high,char * title,int n,
                char * codelist[],int * no,int mark);
```

**相关函数:**

```
unsigned _GetCode(int col,int line,int width,int high,char * title,int n,
                 void (* getrec)(),int * no);
```

**功能说明:**这两个函数均用于实现一个弹出式代码表。代码表采用 WINDOWS 风格窗口,窗口右侧有一个滚动条,通过使用鼠标操作该滚动条可以实现移动光条、翻页等功能。使用鼠标双击某条目即表示选中该条目,随即会退出代码表并恢复原来的屏幕背景,通过参数 `no` 带出被选中的条目的序号。这两个函数的不同之处是它们对代码表条目的来源的要求不同。函数 `getcode` 要求以字符串数组的方式提供代码条目,如果允许重复选择(由参数 `mark` 决定),则使用鼠标左键点击某代码条目时可以在该代码条目前面作一星号标记,以备退出代码表后对其进行处理。而函数 `_GetCode` 要求直接提供一个产生代码条目的函数作为参数。事实上,函数 `getcode` 正是用调用函数 `_GetCode` 的方式实现的。

**参数设置:**

`col` —— 代码表左上角列坐标(以字节为单位);  
`line` —— 代码表左上角行坐标(以像素为单位);  
`width` —— 代码表宽度(以字节为单位);  
`high` —— 代码表高度(以正文行为单位);  
`title` —— 代码表标题;  
`n` —— 代码表长度(代码条目数量);  
`no` —— 希望显示在光条中的菜单条目的序号。在离开代码表函数时为被选中的代码条目的序号;

codelist —— 字符串数组格式的代码条目。如果允许重复选择, 则在每个代码条目的最前面最少有一个空格(供存放星号标记用);

mark —— 是否允许重复选择多个代码条目。以上两个参数由 getcode 函数使用;

getrec —— 由用户提供的构造代码条目的函数。该函数应该具有下列格式:

```
void fun(int i, char * code, unsigned key)
{
    .....
    strcpy(code, ...);
}
```

其中各参数的填写规定为

i —— 代码条目的序号;

code —— 生成的代码条目, 长度不超过 80 个字符;

key —— 调用该函数之前接收的键盘或鼠标按钮键码值(127=单击鼠标左键)。即在函数中要将生成的第 i 个代码条目的内容拷贝到 code 中去。

#### 全局变量:

\_TextColor —— 正文色;

\_Background —— 背景色;

\_TitleColor —— 标题文字色;

\_TitleBk —— 标题背景色;

\_EditColor —— 光条文字色;

\_EditBk —— 光条色;

\_BoxLineColor —— 框的轮廓线色;

\_BarColor —— 框色;

\_Xtimes —— 所有字符或汉字的横向放大倍数;

\_Ytimes —— 所有字符或汉字的纵向放大倍数。

**返回值:** 离开代码表函数时所按下的键盘按键或鼠标按钮。可能是如下统一键盘码: KEY\_ESC、RIGHT\_BUTTON、KEY\_ENTER、LEFT\_BUTTON、KEY\_Left 和 KEY\_Right。通常前两个键码用于表示作废当前的选择。应该指出, 如果使用鼠标左键点击代码表以外的地方也会退出代码表, 此时可以通过测定鼠标光标的位置进行其他控制。

#### 操作说明:

(1) 使用鼠标操作代码表右侧的滚动条可以实现移动光条、翻页等功能(参看滚动条函数 scroll\_h\_box 的说明);

(2) 使用鼠标左键双击某代码条目表示选中该代码条目并退出代码表函数;

(3) 如果允许重复选择(仅 getcode 函数可以), 则使用鼠标左键点击某代码条目表可以给该代码条目前面加上或取消星号标记;

(4) 键盘方向键:

KEY\_Up —— 光条上移一行;  
 KEY\_Down —— 光条下移一行;  
 KEY\_PgUp —— 上翻一页;  
 KEY\_PgDn —— 下翻一页;  
 KEY\_Home —— 光条移至表首;  
 KEY\_End —— 光条移至表尾;

(5)回车键、左右方向键(KEY\_Left 和 KEY\_Right)退出代码表,变量 no 中为光条处代码条目的序号(0 至 n-1);

(6)ESC 键或鼠标右键(RIGHT\_BUTTON)退出代码表,变量 no 中仍为进入代码表时的值。

**举 例:**见程序 11-7 和图 11-2。

**源 程 序:**见程序 11-4、程序 11-6。

### 53. \_GetCode —— 代码表

**函数格式:**

```
unsigned _GetCode(int col,int line,int width,int high,char * title,int n,
                  void(*getrec)(),int * no);
```

**功能说明:**参看函数 getcode 的说明。

### 54. get\_color —— 分析某颜色的红、绿、蓝分量比例

**函数格式:**

```
void get_color(int color_no,int * red,int * green,int * blue);
```

**相关函数:**

```
void set_color(int color_no,int red,int green,int blue);
```

**功能说明:**这两个函数用于读取或改变 VGA 图形模式 12H 中某颜色的红、绿、蓝三原色的比例。函数 get\_color 将指定颜色所含的红、绿、蓝三原色的数值(0~63)读出来,而 set\_color 可以将某颜色设置为程序设计人员指定的其他颜色,其选择范围可达 262114 (256K)种,为制作漂亮新颖的屏幕界面提供了重要的手段。

**参数设置:**

color\_no —— 颜色号。如果取值 0—15,则分别对应 VGA 图形模式 12H 所允许的 16 种颜色,如果取值 16,则可以读写屏幕边缘颜色。可以使用颜色名称常数:

BLACK	—— 黑
BLUE	—— 蓝
GREEN	—— 绿
CYAN	—— 青
RED	—— 红
MAGENTA	—— 洋红
BROWN	—— 棕

	LIGHTGRAY	—— 浅灰
	DARKGRAY	—— 深灰
	LIGHTBLUE	—— 亮蓝
	LIGHTGREEN	—— 亮绿
	LIGHTCYAN	—— 亮青
	LIGHTRED	—— 亮红
	LIGHTMAGENTA	—— 亮洋红
	YELLOW	—— 黄
	WHITE	—— 白
	EDGE_COLOR	—— 屏幕边缘色
red	—— 红分量(0—63);	
green	—— 绿分量(0—63);	
blue	—— 蓝分量(0—63)。	

参 阅:palette 函数的说明。

源 程 序:见程序 1-2、程序 1-4。

## 55. \_GetEMMsize —— 取扩充存储器大小

函数格式:

```
unsigned _GetEMMsize(void);
```

相关函数:

```
void _Init_GDT(void);
```

```
void _SetSourAddr(long addr, unsigned size);
```

```
void _SetDestAddr(long addr, unsigned size);
```

```
void _MoveDataEMM(unsigned size);
```

```
long _LoadEMM(FILE * file, long size);
```

**功能说明:**该组函数使用 BIOS 的功能调用 15H 实现对扩充存储器的直接访问。函数 \_Init\_GDT 用于初始化全局描述符表 GDT,在首次使用 GDT 传输扩充存储器中的数据时应先调用本函数。函数 \_GetEMMsize 用于查询扩充内存容量。为了实现保护模式下常规内存和扩充存储器之间的数据传输,我们设计了 3 个库函数, \_SetSourAddr 用于设置全局描述符表中的源数据地址描述符, \_SetDestAddr 用于设置目标数据地址描述符,而函数 \_MoveDataEMM 调用 INT15H 的 87H 号子功能进行数据传输。\_LoadEMM 函数利用以上函数将文件直接装入扩充存储器。

参数设置:

addr —— 20 位绝对地址(0—16M)。对常规内存中的地址来说,其绝对地址为段地址左移 4 位加偏移量;

size —— 存储块长度(小于 64K,必须为偶数);

file —— 欲装入扩充存储器中的文件的文件名。

返回值:

\_GetEMMsize —— 扩充存储器大小(以 K 为单位);



\_LoadEMM —— 扩充存储器自由空间的首地址。

参 阅: 函数\_ChangeXMS。

源 程 序: 见程序 4-2 至程序 4-5。

## 56. \_GetFigure —— 图形符号输入法模块

功能说明: 参看函数 free\_mode 的说明。

## 57. geth —— 读键盘(不带汉字输入功能)

函数格式:

```
unsigned geth(void);
```

功能说明: 这是一个多功能键盘、鼠标输入函数, 其功能为

(1) 接收 BIOS 键盘中断 16H 来的按键信息, 并将其转换为 HANENV 系统的统一键盘编码。我们将键盘键入的 ASCII 码、组合键和功能键的键盘扫描码、鼠标按钮状态以及汉字机内码综合起来, 编成一套统一键盘码, 用两个字节(unsigned 类型的变量)存储。这样可以大大方便应用程序的编程工作。除了西文 ASCII 码和汉字机内码保持原样不变以外, 其他编码的码值分布在 256—1024 之间, 由头文件 hanenv.h 中的一组宏定义(参看第 14 章);

(2) 接收鼠标按钮信息, 并将其转换为 HANENV 系统的统一键盘编码;

(3) 接收由鼠标控制的屏幕虚拟小键盘来的的按键信息, 并将其转换为 HANENV 系统的统一键盘编码(参看有关函数 free\_MKB 的说明);

(4) 根据由以上几种渠道接收的键盘编码驱动事件触发器。

返 回 值: 由键盘和鼠标按钮信息转换而来的统一键盘码。

源 程 序: 见程序 3-2。

有关事件触发器的注解: 函数 geth 允许应用程序的设计人员将某些功能模块设计为触发器控制方式, 其触发条件可以是热键, 也可以是鼠标事件(在屏幕上某区域按下或释放指定的鼠标按钮)。在头文件 hanenv.h 中有触发器变量类型的定义和变量说明:

```
/* -----
   触发器的数据类型和全局变量的定义
   ----- */
typedef struct          /* 触发器类型的定义 */
{
    unsigned press_key; /* 按键触发事件 */
    unsigned mouse_affair; /* 鼠标触发事件(按钮) */
    int left; /* 鼠标左边界 */
    int top; /* 鼠标上边界 */
    int right; /* 鼠标右边界 */
    int bottom; /* 鼠标下边界 */
    unsigned (* trigger)(unsigned h); /* 事件函数 */
} TRIGGER;
```

```
extern int      _TriggerCount;      /* 触发器个数          */
extern TRIGGER _Triggers[];         /* 触发器数组变量      */
```

在应用程序中定义触发器只需要填写相应的触发器变量并将触发器计数变量的值加 1 即可完成对触发器的设置。在 HANENV 中最多可以定义 20 个触发器事件,其中第 0 个触发器系为定时器(闹钟)准备,不能随意占用。例如,可以为应用程序设计一个弹出式计算器:

```
/* -----
   安装弹出式计算器
   ----- */
if(_TriggerCount<20)
{
    _Triggers[_TriggerCount].press_key = KEY_F2;
    _Triggers[_TriggerCount].trigger   = pop_cal;
    _TriggerCount++;
}
```

在应用程序中将上面这段程序放在 init\_hanenv 函数之后的某处即可设置一个弹出式计算器,其触发事件为热键 F2 键。以上程序段中的触发器函数 pop\_cal 可以编写为

```
unsigned pop_cal(unsigned h)
{
    h = 0;
    calcutor(20,100);
    return h;
}
```

## 58. gethan —— 读键盘(带汉字输入功能)

函数格式:

```
unsigned gethan(void);
```

**功能说明:**该函数除了具有 geth 函数的所有功能以外,还可以输入汉字。预定义的基本汉字输入法有区位码输入法和图形符号输入法两种,如果还要使用其他汉字输入法或西文全角字符输入法,可以使用函数 set\_han\_mode 和函数 set\_asc\_mode。

**操作说明:**使用复合功能键 Alt+Fn 可以切换各种汉字输入法。几个基本输入法使用的切换键为

Alt+F1 —— 国标区位码输入法;

Alt+F6 —— 英文数字输入;

Alt+F9 —— 图形符号输入法。

**返回值:**统一键盘码或汉字机内码。

**源程序:**见程序 8-1。

**参 阅:** 参看函数 `free_han_mode` 的说明。

### 59. `get_mouse_range` —— 取当前鼠标活动范围

**函数格式:**

```
void get_mouse_range(int *left, int *top, int *right, int *bottom);
```

**相关函数:**

```
int havemouse(void);
int get_mouse_status(int *x, int *y);
int mouse_enter(int left, int top, int right, int bottom);
int ismouselight(void);
void mouse_moved(int *dx, int *dy);
int mouse_pressed(int button, int *x, int *y, int *count);
int mouse_release(int button, int *x, int *y, int *count);
int mousecolor(void);
int mousebk(void);
int mousecol(void);
int mouserow(void);
```

**功能说明:** 该组函数测试鼠标的当前状态。各函数的具体功能如下:

<code>havemouse</code>	—— 鼠标驱动程序是否安装成功;
<code>get_mouse_range</code>	—— 取当前鼠标活动范围;
<code>get_mouse_status</code>	—— 取鼠标当前的按钮状态及鼠标光标位置;
<code>mouse_enter</code>	—— 测试鼠标光标是否进入指定矩形区域;
<code>ismouselight</code>	—— 测试鼠标光标是否打开;
<code>mouse_moved</code>	—— 测试鼠标光标相对于上次测试时位置的位移;
<code>mouse_pressed</code>	—— 测试鼠标指定键的击键次数及鼠标光标坐标;
<code>mouse_release</code>	—— 测试鼠标指定键的释放次数及鼠标光标坐标;
<code>mousecolor</code>	—— 测试鼠标光标的颜色;
<code>mousebk</code>	—— 测试鼠标光标的背景色;
<code>mousecol</code>	—— 测试鼠标光标的横坐标;
<code>mouserow</code>	—— 测试鼠标光标的纵坐标;

**参数设置:**

`x, y` —— 鼠标光标的当前坐标;

`button` —— 鼠标按钮的统一键盘码, 应使用宏定义:  
           `LEFT_BUTTON`: 左按钮  
           `RIGHT_BUTTON`: 右按钮

`count` —— 鼠标击键或释放的次数;

`dx, dy` —— 鼠标移动的距离;

`left, top, right, bottom` —— 鼠标活动范围的左上角和右下角的坐标值。

**返回值:**

havemouse	—— 鼠标驱动程序安装成功返回非零值, 否则返回零;
get_mouse_status	—— 鼠标当前的按键值: NO_BUTTON: 无按钮按下 LEFT_BUTTON: 左按钮按下 RIGHT_BUTTON: 右按钮按下
mouse_enter	—— 鼠标光标进入指定矩形区域返回非零值, 否则返回零;
ismouselight	—— 鼠标光标打开时返回非零值, 否则返回零;
mouse_pressed	—— 鼠标指定键按下时返回非零值, 否则返回零;
mouse_release	—— 鼠标指定键释放时返回非零值, 否则返回零;
mousecolor	—— 鼠标光标的颜色;
mousebk	—— 鼠标光标的背景色;
mousecol	—— 鼠标光标的横坐标;
mouserow	—— 鼠标光标的纵坐标;

**参 阅:** 函数 `delight_mouse`、`set_mouse...` 以及 `till_mouse_pop` 的说明。

**源 程 序:** 函数 `havemouse`、`ismouselight`、`mousecolor`、`mousebk`、`mousecol`、`mouserow` 和 `mouse_enter` 是定义于头文件 `hanenv.h` 中的宏, 参看第 14 章。其他函数的源程序见程序 2-8 至程序 2-11。

### 60. `get_mouse_status` —— 取鼠标当前的按键及光标位置

**函数格式:**

```
int get_mouse_status(int *x, int *y);
```

**功能说明:** 参阅 `get_mouse_range`。

### 61. `_GetNewCode` —— 通用码表输入法模块

**功能说明:** 参看函数 `free_mode` 的说明。

### 62. `_GetPixel` —— 取屏幕象素颜色

**函数格式:**

```
int _GetPixel(int x, int y);
```

**功能说明:** 取屏幕上指定坐标处的象素颜色值。

**参数设置:**

`x, y` —— 屏幕象素坐标;

**返回值:** 屏幕象素颜色值(0—15)。

### 63. `_GetPY` —— 拼音输入法模块

功能说明:参看函数 `free_mode` 的说明。

#### 64. `_GetSP` —— 区位码输入法模块

功能说明:参看函数 `free_mode` 的说明。

#### 65. `get_window` —— 取正文窗口参数

函数格式:

```
void get_window(int left,int top,int right,int bottom);
```

相关函数:

```
void set_window(int left,int top,int right,int bottom);
```

功能说明:函数 `set_window` 用于设置正文窗口的位置和大小;`get_window` 用于读取当前正文窗口的设置参数。

参数设置:

`left` —— 正文窗口左上角列坐标(单位为字节);

`top` —— 正文窗口左上角行坐标(单位为像素);

`right` —— 正文窗口右下角列坐标(单位为字节);

`bottom` —— 正文窗口右下角行坐标(单位为像素);

源程序:这两个函数都是定义于头文件 `hanenv.h` 中的宏,参看第14章。

#### 66. `_GetXMS` —— 申请 XMS 块

函数格式:

```
int _GetXMS(int size);
```

功能说明:参阅 `_ChangeXMS`。

#### 67—76. `getxy ...` —— 全屏幕数据编辑函数族

函数格式:

```
unsigned getxya(int col,int line,int width,char * string,char * picture);
```

```
unsigned getxyb(int col,int line,int * value);
```

```
unsigned getxyc(int col,int line,int * value,char * set);
```

```
unsigned getxyd(int col,int line,struct date * value);
```

```
unsigned getxyf(int col,int line,int width,double * value,int dec,double limit1,  
               double limit2);
```

```
unsigned getxyi(int col,int line,int width,int * value,int limit1,int limit2);
```

```
unsigned getxyl(int col,int line,int width,long * value,long limit1,  
               long limit2);
```

```
unsigned getxyp(int col,int line,int width,char * string);
```

```
unsigned getxys(int col,int line,int width,char * string,int * len);
```

```
unsigned getxyt(int col,int line,int width,int high,char * string);
```

**功能说明:**该组函数用于数据字段的屏幕编辑。具体说明如下:

- getxya —— 基本字段编辑函数,由其他字段编辑函数调用;
- getxyb —— 逻辑数据编辑函数;
- getxyc —— 字符数据编辑函数;
- getxyd —— 日期数据编辑函数;
- getxyf —— 双精度数据编辑函数;
- getxyi —— 整型数据编辑函数;
- getxyl —— 长整型数据字段编辑函数;
- getxyp —— 口令编辑函数;
- getxys —— 字符串数据编辑函数;
- getxyt —— 字符串数据编辑函数,编辑区域为矩形。

**参数设置:**

- col,line —— 编辑窗口坐标(注意横坐标以字节为单位);
- width —— 编辑窗口宽度;
- high —— 编辑窗口高度(以字符行为单位,仅 getxyt 函数使用);
- value —— 被编辑的数据;
- dec —— 编辑双精度数据时的小数位;
- limit1,limit2 —— 数据的上限和下限;
- string —— 被编辑的字符串;
- len —— 指向被编辑字符串长度变量的指针。在结束编辑后,该变量将带回退出编辑时被编辑字符串的实际长度;
- picture —— 编辑格式字符串;
- set —— 在编辑字符类型数据时允许使用的字符集合。

**全局变量:**

- \_EditColor —— 编辑区的字符颜色;
- \_EditBk —— 编辑区背景颜色;
- \_TextColor —— 退出编辑函数后的正文色;
- \_Background —— 退出编辑函数后的背景色;
- \_Xtimes —— 所有字符或汉字的横向放大倍数;
- \_Ytimes —— 所有字符或汉字的纵向放大倍数。

**返回值:**退出编辑函数时所使用的编辑键码。如果在编辑时对原来的数据内容做过修改,则该键码值被加上 1024。

**操作说明:**进入编辑函数后,首先在屏幕上显示编辑窗口,内容为被编辑数据原来的值。此时即可使用键盘对数据进行编辑。编辑用键:

- KEY\_Left —— 光标左移,如果光标位于编辑窗口首部则退出编辑;
- KEY\_Right —— 光标右移,如果光标位于编辑窗口尾部则退出编辑;
- KEY\_Home —— 光标移至编辑窗口之首,如果光标已经位于编辑窗口首部则退出编辑;
- KEY\_End —— 光标移至编辑窗口之尾,如果光标已经位于编辑窗口尾部则

退出编辑；

KEY\_Up —— 光标上移,如果光标位于编辑窗口顶行则退出编辑；

KEY\_Down —— 光标下移,如果光标位于编辑窗口底行则退出编辑；

(以上两个键的解释仅适用于 getxyt。对其他函数来说,使用这两个键均退出编辑)

KEY\_Del —— 删除光标处的字符或汉字；

Backspace —— 删除光标前的字符或汉字；

KEY\_Ins —— 切换插入/改写状态；

KEY\_ESC(包括鼠标右键) —— 退出编辑,恢复被编辑数据原来的值；

KEY\_ENTER —— 结束编辑；

此外,其他编辑键如 PgUp 键、PgDn 键、Ctrl+ 光标移动键、Ctrl+ 字母键以及鼠标左键等均为结束编辑(和回车键的功能相同)。在使用 getxys 和 getxyt 编辑字符串时允许使用汉字。

**源程序:**部分源程序见程序 12-1 和程序 12-2。

**举例:**使用以上函数编写全屏幕数据编辑界面的例子见程序 12-3。

## 77. han\_error —— 错误检测

**函数格式:**

```
int han_error(void);
```

**功能说明:**HANENV 系统中某些函数将其出错信息存入全局变量,本函数用于读出这些错误信息。以这种方式提供错误信息的函数有: \_SizeofXMS、\_GetXMS、\_ChangeXMS、\_FreeXMS、\_MoveDataXMS 以及 calculte。

**返回值:**错误码。函数 calculte 的错误码可以参看函数 calculte 的说明,其他函数的错误码为:

- 080H —— 功能未实现；
- 081H —— 检测到了 VDISK 设备；
- 082H —— A20 出现错误；
- 0A0H —— 所有扩充内存都已分配；
- 0A1H —— 所有扩充内存句柄均被占用；
- 0A2H —— 句柄无效；
- 0A3H —— 无效源句柄；
- 0A4H —— 无效源偏移量；
- 0A5H —— 无效目的句柄；
- 0A6H —— 无效目的偏移量；
- 0A7H —— 长度无效；
- 0A8H —— 移动有重叠；
- 0A9H —— 奇偶校验错；
- 0ABH —— 句柄加锁。

## 78. havemouse —— 测试鼠标驱动程序是否安装成功

函数说明:

```
int havemouse(void);
```

功能说明:参阅 get\_mouse\_range。

## 79. \_H\_Line —— 画横线

函数格式:

```
void _H_Line(int x,int y,int len,int c,unsigned char pattern);
```

相关函数:

```
void _V_Line(int x,int y,int len,int c,unsigned char pattern);
```

功能说明:画水平或垂直线。

参数设置:

x, y —— 线左端点或上端点坐标,均以点为单位;  
len —— 线长度,以点为单位;  
c —— 线颜色;  
pattern —— 线型,pattern 中为 1 的位对应点,为 0 的位对应空;

源程序:见程序 10-2 和程序 10-3。

## 80. \_Hole —— 画一具有凹凸感的矩形框

函数格式:

```
void _Hole(int x,int y,int w,int h,int c1,int c2,int thickness);
```

功能说明:画一具有凹凸感的矩形框。通过恰当地选用 c1 和 c2,即可画出凸出的按键或是凹进的窗口。

参数设置:

x, y —— 框左上角坐标;  
w, h —— 框的宽和高,均以点为单位;  
c1, c2 —— 框左、上边的颜色和框右、下边的颜色;  
thickness —— 框边的厚度。

举 例:

```
#include <stdio.h>
#include <dos.h>
#include <hanenv.h>

main()
{
    init_hanenv();
    _Block(0,0,80,480,LIGHTGRAY);
    _Hole(160,100,80,40,WHITE,DARKGRAY,2);
    _Hole(400,100,80,40,DARKGRAY,WHITE,2);
```



```

    geth();
    close_hanenv();
}

```

**运行结果:**在屏幕上分别显示一个凸矩形和一个凹矩形。

**源程序:**见程序 10-9。

### 81. icon\_menu —— 图标式菜单

**函数格式:**

```
unsigned icon_menu(ICON_MENU * menu);
```

**功能说明:**该函数用于实现图标式菜单。关于该函数的参数填写、使用方法举例等请参阅 13.2 节。

### 82. \_Init\_GDT —— 初始化扩充存储器参数

**函数格式:**

```
void _Init_GDT(void);
```

**功能说明:**参阅 \_GetEMMsize。

### 83. init\_hanenv —— 初始化 HANENV 系统

**功能说明:**参阅 close\_hanenv 函数。

### 84. init\_XMS —— 初始化扩充存储器

**函数格式:**

```
int init_XMS(void);
```

**功能说明:**参阅 \_ChangeXMS。

### 85—89. is... —— 汉字判断函数族

**函数格式:**

```
int isedit(unsigned h);
```

```
int ishan(char * s);
```

```
int isins(void);
```

```
int isscan(unsigned h);
```

```
int isSP(unsigned h);
```

**功能说明:**该组函数用于判断汉字或统一键盘码。各函数的具体功能如下:

isedit —— 用于判定键码是否编辑用键。所谓编辑用键是指:

```
KEY_Home;
```

```
KEY_Up;
```

```
KEY_PgUp;
```

```
KEY_Left;
```

KEY\_Right;

KEY\_End;

KEY\_Down;

KEY\_PgDn;

以及这组键与 Ctrl 键和左、右 Shift 键的组合、Ctrl 键与字母 键的组合;

ishan —— 用于判断指针 s 是否指向汉字;

isins —— 用于判断是否处于插入状态;

isscan —— 用于判断 h 是否扫描码(256 — 1024);

isSP —— 用于判断 h 是否汉字。

**返回值:**若判断成功则返回非零值,否则返回零。

**源程序:**函数 isSP 和 ishan 为头文件 hanenv.h 中定义的宏,参看第14章。

90. iscursorlight —— 检测光标是否打开

**函数格式:**

int iscursorlight(void);

**功能说明:**参阅 delightcursor。

91. ismouselight —— 测试鼠标光标是否打开

**函数格式:**

int ismouselight(void);

**功能说明:**参阅 get\_mouse\_range。

92. largestXMS —— 求扩充存储器剩余最大存储块的大小

**函数格式:**

int largestXMS(void);

**功能说明:**参阅\_ChangeXMS。

93. light\_clock —— 显示时钟

**函数格式:**

void light\_clock(void);

**功能说明:**参阅 close\_clock。

94. lightcursor —— 打开光标

**函数格式:**

void lightcursor(void);

**功能说明:**参阅 delightcursor。

## 95. light\_mouse —— 打开鼠标光标

函数格式:

```
void light_mouse(void);
```

功能说明:参阅 delight\_mouse。

## 96. \_LoadEMM —— 将文件装入扩充存储器

函数格式:

```
long _LoadEMM(FILE *file, long size);
```

功能说明:参阅 \_GetEMMsize。

## 97. \_LoadEnv —— 恢复西文字符模式的屏幕信息

函数格式:

```
void _LoadEnv(void);
```

相关函数:

```
void _SaveEnv(void);
```

功能说明:这两个函数用于保存和恢复原西文字符模式的屏幕信息。

源程序:见程序 5-1。

## 98. load\_MKB —— 安装虚拟键盘

函数格式:

```
void _Cdecl load_MKB(key);
```

功能说明:参阅 free\_MKB。

## 99. load\_HZK —— 装入汉字库

函数格式:

```
HZK * load_HZK(int width, int high, char * hzkname, unsigned char * codelist,  
               int wherefont);
```

功能说明:参阅 free\_HZK。

## 100. \_LoadXMS —— 将文件装入扩展存储器

函数格式:

```
int _LoadXMS(FILE *file, long size);
```

功能说明:参阅 \_ChangeXMS。

## 101. ltrim —— 删除字符串左端的空格

函数格式:

```
char * ltrim(char * string);
```

相关函数:

```
char * trim(char * string);
```

功能说明:这两个函数分别用于去除字符串前端和尾部的空格。

参数设置:

string —— 被处理的字符串。

返回值:指向被处理的字符串的指针。

102. mousebk —— 取鼠标光标背景色

函数格式:

```
int mousebk(void);
```

功能说明:参阅 get\_mouse\_range。

103. mousecol —— 取鼠标光标横坐标

函数格式:

```
int mousecol(void);
```

功能说明:参阅 get\_mouse\_range。

104. mousecolor —— 取鼠标光标颜色

函数格式:

```
int mousecolor(void);
```

功能说明:参阅 get\_mouse\_range。

105. mouse\_enter —— 测试鼠标光标是否进入指定区域

函数格式:

```
int mouse_enter(int left,int top,int right,int bottom);
```

功能说明:参阅 get\_mouse\_range。

106. mouse\_moved —— 测试鼠标移动

函数格式:

```
void mouse_moved(int * dx,int * dy);
```

功能说明:参阅 get\_mouse\_range。

107. mouse\_pressed —— 测试鼠标按键是否按下

函数格式:

```
int mouse_pressed(int which_button,int * x,int * y,int * count);
```

功能说明:参阅 get\_mouse\_range。

108. mouse\_release —— 测试鼠标按键是否释放

函数格式:

```
int mouse_release(int which_button, int *x, int *y, int *count);
```

**功能说明:**参阅 get\_mouse\_range。

#### 109. mouserow —— 取鼠标光标纵坐标

**函数格式:**

```
int mouserow(void);
```

**功能说明:**参阅 get\_mouse\_range。

#### 110. movecursor —— 将光标移至指定位置

**函数格式:**

```
void movecursor(int col, int line);
```

**功能说明:**参阅 delightcursor。

#### 111. \_MoveDataEMM —— 扩充存储器与内存交换数据

**函数格式:**

```
void _MoveDataEMM(unsigned size);
```

**功能说明:**参阅 \_GetEMMsize。

#### 112. \_MoveDataXMS —— 扩展存储器与内存交换数据

**函数格式:**

```
int _MoveDataXMS(void);
```

**功能说明:**参阅 \_ChangeXMS。

#### 113. \_MoveImage —— 拷贝屏幕区域

**函数格式:**

```
void _MoveImage(int col1, int y1, int width, int high, int col2, int y2);
```

**功能说明:**本函数用于将一屏幕矩形区域拷贝至屏幕上另一位置(可重叠),无需内存缓冲区,速度快。本函数的拷贝范围为整个显示存储器 VRAM,故可以实现一些特殊功能。

**参数设置:**

col1, y1 —— 原屏幕矩形区域左上角坐标,col1 的单位为字节,y1 的单位为像素;

width, high —— 原屏幕矩形区域的宽与高,width 以字节为单位,high 以像素为单位;

col2, y2 —— 新区域左上角坐标,col2 以字节为单位,y2 以像素为单位。

**源程序:**见程序 1-8。

#### 114. open\_display —— 打开屏幕显示

**函数格式:**

```
void open_display(void);
```

功能说明: 参阅 close\_display。

### 115—123. out ... ——正文方式汉字显示函数族

函数格式:

```
void outc(int ch);
void outchar(int ch);
void outh(unsigned han);
void outhan(unsigned han);
void outh1(unsigned h,int whichhalf);
void outhan1(unsigned h,int whichhalf);
void outs(char *s);
void outstr(char *s);
void putnstr(char *s,int width);
```

功能说明: out... 函数族(包括函数 putnstr)调用字节显示字模函数 OutF 和 OutFont, 用于文本方式下显示字符、汉字或字符串, 速度较快。在 HANENV 系统中的文本模式实际上是在 VGA 的图形模式 12H 下模拟西文文本工作状态, 此时由 HANENV 系统提供一个软件闪烁光标, 字符和汉字的显示位置由光标的位置确定, 并可设置文本窗口。各函数的具体功能说明如下:

outc	——显示 ASCII 字符(包括扩展 ASCII 码);
outchar	——显示 ASCII 字符, 可以设置放大倍数;
outh	——显示汉字或全角字符;
outhan	——显示汉字, 可以设置放大倍数;
outh1	——显示半个汉字;
outhan1	——显示半个汉字, 可以设置放大倍数;
outs	——显示字符串;
outstr	——显示字符串, 可以设置放大倍数;
putnstr	——显示字符串的前 n 个字符, 清背景, 可以设置放大倍数;

参数设置:

ch	——待显示字符;
han	——待显示汉字的机内码;
s	——待显示的字符串。
widthhalf	——在显示半个汉字时表示处理哪半个汉字(0=左, 1=右);
width	——在 putnstr 函数确定输出宽度。

全局变量:

_TextCol	——文本光标的列坐标;
_TextLine	——文本光标的行坐标;
_TextColor	——字符或汉字颜色;
_Background	——背景色(仅用于函数 putnstr);

\_Xtimes           ——汉字或字符的横向放大倍数;  
 \_Ytimes           ——汉字或字符的纵向放大倍数;  
 \_TextWinLeft      ——正文窗口左列坐标(单位为字节);  
 \_TextWinTop       ——正文窗口顶行坐标(单位为象素);  
 \_TextWinRight     ——正文窗口右列坐标(单位为字节);  
 \_TextWinBottom    ——正文窗口底行坐标(单位为象素)。

参 阅: \_OutF 函数及 9.3 节:“正文工作方式”。

源 程 序: outc、outchar、outh 以及 outhan 是定义于头文件 hanenv.h 中的宏, 可参看第 14 章。函数 outs 和 outhl 的源程序见程序 9-3 和程序 9-4。

#### 124. \_OutF ——显示字模点阵

函数格式:

```
void _OutF(int col,int line,int width,int high,int color,unsigned char *font);
```

功能说明: 参看函数 DrawF。

#### 125. \_OutFont ——显示字模点阵

函数格式:

```
void _OutFont(int col,int line,int width,int high,int color,unsigned char *font);
```

功能说明: 参看函数 DrawF。

#### 126—131. outxy ... ——汉字显示函数族

函数格式:

```

void outxyc(int col,int line,int color,int ch);
void outxychar(int col,int line,int color,int ch);
void outxyh(int col,int line,int color,unsigned h);
void outxyhan(int col,int line,int color,unsigned h);
void outxys(int col,int line,int color,char *s);
void outxystr(int col,int line,int color,char *s);

```

功能说明: outxy... 函数族调用字节显示字模函数 \_OutF 和 \_OutFont, 用于显示字符、汉字或字符串。outxy... 函数族的特点是显示速度快, 但水平方向的显示位置只能以字节为单位。各函数的具体功能说明如下:

outxyc       ——显示 ASCII 码字符(包括扩展 ASCII 字符);  
 outxychar    ——显示 ASCII 码字符, 并可设置放大倍数;  
 outxyh       ——显示汉字或全角字符;  
 outxyhan     ——显示汉字或全角字符, 并可设置放大倍数;  
 outxys       ——显示字符串;  
 outxystr     ——显示字符串, 并可设置放大倍数;

参数设置:

col,line   ——汉字或字符点阵左上角坐标, 其中 col 的单位为字节;

color —— 汉字或字符的颜色;  
 ch —— 待显示字符;  
 h —— 待显示汉字的机内码;  
 s —— 待显示的字符串。

**全局变量:**

\_Xtimes —— 汉字或字符的横向放大倍数;  
 \_Ytimes —— 汉字或字符的纵向放大倍数;

**参 阅:** \_DrawF 函数、drawxy... 函数族。

**源 程 序:** outxyc、outxychar、outxyh 和 outxyhan 是定义于头文件 hanenv. h 中的宏,可参看第 14 章。

### 132. palette —— 调色板工具

**函数格式:**

```
unsigned palette(int col,int line,char * buffer);
```

**功能说明:** 本函数在屏幕上弹出一个用滚动条控制的调色板,如图 11-3 所示。

**参数设置:**

col —— 调色板左上角列坐标(以字节为单位);  
 line —— 调色板左上角行坐标(以像素为单位);  
 buffer —— 调色板数据缓冲区(共需 51 字节)。

**返 回 值:** 退出调色板所用的键码,如果是 KEY\_ESC 则表示作废本次调整的结果,恢复原来的调色板数据;如果是 KEY\_ENTER 则表示保留调整的结果(存在 buffer 处)。

**操作说明:** 从图 11-3 中可以看出,在调色板下方有一排各种颜色的小方块,共有 17 个,前 16 个代表当前屏幕上可以使用的 16 种色彩,最右边的一个中空黑色方框代表屏幕边缘颜色。在最左端的黑色小方块外面还套着一个矩形框,使用鼠标操纵调色板最下方的滚动条就可以通过移动矩形框来选择要修改的屏幕色彩。在调色板上还有 3 个横向的滚动条,分别用于调整被选中的颜色的红、绿、蓝三原色分量的强度(其分量值在滚动条左边的括号中显示),调整的效果可以通过被选中的颜色块的变化反映出来(但屏幕边缘色变化的效果只能通过直接观察屏幕边缘得出)。使用鼠标左键点选调色板右边的按键或直接使用 ESC 键均可退出调色板函数。

**源 程 序:** 见程序 11-9。

### 133. prnscr —— 屏幕拷贝

**函数格式:**

```
void prnscr(int x1,int y1,int x2,int y2,int margin,unsigned palette);
```

**功能说明:** 将屏幕上指定区域的图形打印出来。

**参数设置:**

x1,y1 —— 打印区域左上角坐标;  
 x2,y2 —— 打印区域右下角坐标;  
 margin —— 左页边宽度(以像素计);



palette ——打印调色板, palette 的每一位对应一种颜色, 如该位为 1 则表示对应颜色的象素打印, 反之表示不打印。例如 0x7fff 表示除了白色其他颜色的象素均打印, 而 0xffffe 表示除了黑色之外其他颜色的象素均打印

**全局变量:**

\_Xtimes, \_Ytimes ——打印图象的横向和纵向放大倍数。

**注 意:**本函数可在 LQ1600K 打印机(或图形打印命令与其兼容的打印机)上打印屏幕图形。

**源程序:**见程序 11-10。

#### 134. putblock ——恢复屏幕块

**函数格式:**

```
void putblock(int col, int line, int width, int high, char * * block);
```

**功能说明:**参阅 getblock。

#### 135. putblockXMS ——从扩展存储器中恢复屏幕块

**函数格式:**

```
void putblockXMS(int left, int top, int right, int bottom, unsigned handle);
```

**功能说明:**参阅 getblockXMS。

#### 136. putnstr ——显示字符串的前 n 个字符

**函数格式:**

```
void putnstr(char * s, int width);
```

**功能说明:**见函数 outxy。

#### 137. \_PutPixel ——画点

**函数格式:**

```
void _PutPixel(int x, int y, int color);
```

**功能说明:**在指定位置以给定颜色画点。

**参数设置:**

x, y ——点坐标;

color ——点颜色;

**源程序:**见程序 6-1 和程序 C-2。

#### 138—143 putxy ——汉字显示函数族

**函数格式:**

```
void putxyc(int col, int line, int color, int bk, int c);
```

```
void putxychar(int col, int line, int color, int bk, int c);
```

```
void putxyb(int col, int line, int color, int bk, unsigned h);
```

```
void putxyhan(int col,int line,int color,int bk,unsigned h);
void putxys(int col,int line,int color,int bk,char * string);
void putxystr(int col,int line,int color,int bk,char * string);
```

功能说明: putxy... 函数族调用字节显示字模函数 \_OutF 和 \_OutFont, 用于显示字符、汉字或字符串。putxy... 函数族的特点是显示时先清字符或汉字下的背景。各函数的具体功能说明如下:

putxyc —— 显示 ASCII 码字符(包括扩展 ASCII 字符);  
 putxychar —— 显示 ASCII 码字符,并可设置放大倍数;  
 putxyh —— 显示汉字或全角字符;  
 putxyhan —— 显示汉字或全角字符,并可设置放大倍数;  
 putxys —— 显示字符串;  
 putxystr —— 显示字符串,并可设置放大倍数;

#### 参数设置:

col,line —— 汉字或字符点阵左上角坐标,其中 col 的单位为字节;  
 color —— 汉字或字符的颜色;  
 bk —— 背景色;  
 ch —— 待显示字符;  
 h —— 待显示汉字的机内码;  
 s —— 待显示的字符串。

#### 全局变量:

\_Xtimes —— 汉字或字符的横向放大倍数;  
 \_Ytimes —— 汉字或字符的纵向放大倍数;

参 阅: \_DrawF 函数、outxy... 函数族。

源 程 序: putxyc、putxychar、putxyh 和 putxyhan 是定义于头文件 hanenv.h 中的宏,可参看第 14 章。

#### 144. reset \_mouse —— 重置鼠标

函数格式: void reset \_mouse(void);

功能说明: 初始化鼠标驱动程序。

注 意: 在 init \_hanenv 函数中已经初始化鼠标驱动程序,所以应用程序中一般不需再调用本程序。

源 程 序: 见程序 2-1。

#### 145. run \_clock —— 安装时钟

函数格式:

void run \_clock(void);

功能说明: 安装时钟处理模块。只有在安装了时钟处理模块之后,才能使用函数 light \_clock 显示时钟。

源 程 序: 见程序 9-6。

## 146. \_SaveEnv —— 保存西文字符模式的屏幕信息

函数格式:

void \_SaveEnv(void);

功能说明: 参阅 \_LoadEnv。

## 147. scroll\_down —— 窗口下滚

函数格式:

void scroll\_down(int high);

相关函数:

void scroll\_up(int high);

功能说明: 这两个函数用于实现正文窗口内容的滚动。函数 scroll\_up 用于向上滚动窗口, 函数 scroll\_down 用于向下滚动窗口。

参数设置:

high —— 正文窗口上滚或下滚的线数。

全局变量:

\_TextWinLeft, \_TextWinTop, \_TextWinRight, \_TextWinBottom —— 正文窗口的边界;

\_TextCol, \_TextLine, —— 光标位置;

\_Background —— 正文窗口的背景色。

源程序: 见程序 9-5。

## 148. scroll\_h\_box —— 仿 WINDOWS 风格的横滚动条

函数格式:

```
void scroll_h_box (int x,int y,int len,int pagesize,unsigned key,int range,
                  int *n,int *oldpos);
```

相关函数:

```
void scroll_v_box (int x,int y,int len,int pagesize,unsigned key,int range,
                  int *n,int *oldpos);
```

功能说明: 滚动条是 WINDOWS 窗口式用户界面的重要特征之一。通过滚动条, 应用程序的操作人员可以利用鼠标方便地操作表格数据的选择、光条的移动和窗口的翻页。我们提供的滚动条函数共有两个, 一个是横向滚动条函数 scroll\_h\_box, 一个是纵向滚动条函数 scroll\_v\_box。这两个函数共享一组字模数据和内部显示函数, 功能基本相同, 均为控制某个量(参数 n)在指定范围(0—参数 range)内变化。

参数设置:

x,y —— 滚动条左上角坐标(以像素为单位);

len —— 滚动条长度;

pagesize —— 页面长度;

key —— 滚动条驱动键码;

- range —— 受滚动条控制的变量的取值范围(0—range);  
 n —— 受滚动条控制的变量;  
 oldpos —— 滑标位置。

滚动条函数的参数表中以下几个参数的设置比较重要且不易掌握:

- page- —— 使用鼠标左键点击滚动条条身时受控制变量的增减数值, 可以用  
 size 来控制编辑器的翻页、快速查找等;  
 key —— 在调用程序中是使用什么键(geth、gethan 等函数的返回值, 包括  
 键盘键和鼠标按钮)进入滚动条函数的。如果是下列编辑键值加上  
 一个固定常数(对纵向滚动条来说是 1536, 对横向滚动条来说是  
 2048), 则可改变受控制变量的值:

- KEY\_Up : 受控制变量值减 1(纵向滚动条);  
 KEY\_Down : 受控制变量值加 1(纵向滚动条);  
 KEY\_Left : 受控制变量值减 1(横向滚动条);  
 KEY\_Right : 受控制变量值加 1(横向滚动条);  
 KEY\_PgUp : 受控制变量值减页面长度;  
 KEY\_PgDn : 受控制变量值加页面长度;  
 KEY\_Home : 受控制变量值变为 0;  
 KEY\_End : 受控制变量值变为 range。

如果 key 之值为鼠标左键, 就可能需要使用鼠标操纵滚动条, 否则只需根据参数 n 的值调整滚动条上滑标的位置;

- n —— 该参数是使用滚动条控制的整型变量的地址。为了方便起见, 我们  
 规定该变量的变化范围为 0—range 之间, 即如果其值为 0, 则滚动  
 条中的滑标位于滚动条的最上(左)端, 如果等于 range, 则滑标位  
 于滚动条的最下(右)端;  
 range —— 使用滚动条控制的变量(参数 n)的最大值。由于该参数为整型, 所  
 以其值最大不能超过 32767;  
 oldpos —— 这实际上是一个滚动条函数的工作变量, 用来保存上次调用滚动  
 条函数时滑标的位置。另外, 我们还用这个变量表示是否是第一次  
 调用滚动条函数。如果该参数的值为 -1, 则表示是第一次调用滚  
 动条函数, 此时只是画出滚动条后即退出滚动条函数。

调用方法: 使用滚动条需要如下程序结构:

```
/* -----
   滚动条应用举例
   ----- */
unsigned key;
int      oldpos = -1;
int      n      = 0;
.....
```

```

scroll_h_box(x,y,len,10,color,range,&n,&oldpos); /* 显示滚动条 */
循环((条件))
{
    key = getch(); /* 或 key = gethan(); */
    ....
    scroll_h_box(x,y,len,10,color,range,&n,&oldpos);
}

```

其中 n 可以用来控制表格的当前行和列、数据库的当前记录、当前字段等。

**全局变量:**

\_BoxLineColor —— 框的轮廓线色;  
\_BarColor —— 滚动条按键及滑标颜色;

**操作方法:**滚动条使用鼠标,其操作方法有以下三种:

(1)用鼠标左键点选滚动条两端的按键,可以使变量 n 的值增大或减小 1,同时修改滑标的位置;

(2)用鼠标左键点击滚动条条身(不包括滑标),可以使变量 n 的值加或减一个页面长度,同时修改滑标的位置;

(3)使鼠标光标指向滑标,然后按下鼠标左键不松手,可以拖动滑标在滚动条上滑动,待松手后即可按比例调整量 n 的值。

**源程序:**见程序 11-3。

#### 149. scroll\_up —— 窗口上滚

**函数格式:**

```
void scroll_up(int high);
```

**功能说明:**参阅 scroll\_down。

#### 150. scroll\_v\_box —— 仿 WINDOWS 风格的竖滚动条

**函数格式:**

```
void scroll_v_box (int x,int y,int len,int pagesize,unsigned key,int range,
                  int *n,int *oldpos);
```

**功能说明:**参阅 scroll\_h\_box。

#### 151~161. set ... —— 环境变量设置函数族

**函数格式:**

```

void set_background(int color);
void set_text_color(int color);
void set_edit_color(int color);
void set_edit_bk(int color);
void set_title_bk(int color);

```

```

void set_title_color(int color);
void set_tab_bk(int color);
void set_tab_color(int color);
void set_bar_color(unsigned bar_color);
void set_xtimes(int times);
void set_ytimes(int times);

```

**功能说明:**这组函数用于设置全局变量的值。具体说明如下:

```

set_background  —— 设置_TextColor(正文色);
set_text_color  —— 设置_Background(背景色);
set_edit_color  —— 设置_EditColor(编辑区正文色);
set_edit_bk     —— 设置_EditBk(编辑区背景色);
set_title_bk    —— 设置_TitleColor(标题区正文色);
set_title_color —— 设置_TitleBk(标题区背景色);
set_tab_bk      —— 设置_TabColor(制表区正文色);
set_tab_color   —— 设置_TabBk(制表区背景色);
set_bar_color   —— 设置_BarColor(立体按键所用的 4 种颜色);
set_xtimes      —— 设置_Xtimes(字符横向放大倍数);
set_ytimes      —— 设置_Ytimes(字符纵向放大倍数);

```

注意这些全局变量的解释带有一定的随意性,如果某个函数使用了这些全局变量,其解释在相应的函数说明条目中,同时应用程序设计人员也可以使用这些全局变量传递自己的数据,但要注意不能和使用这些全局变量的函数冲突。

**参数设置:**

```

color      —— 颜色(0-15);
bar_color  —— 立体按键所用的 4 种颜色,可以使用如下宏:
                BLUE_BAR      : 蓝色按键;
                GREEN_BAR     : 绿色按键;
                CYAN_BAR      : 青色按键;
                RED_BAR        : 红色按键;
                MAGENTA_BAR   : 洋红按键;
                BROWN_BAR     : 棕色按键;
                GRAY_BAR      : 灰色按键。

```

当然也可以按下列原则自行构造此参数:0-3 位为键名字符 颜色,4-7 位为键帽颜色,8-11 位为键帽左、上边颜色,12-15 位为键帽右、下边颜色;

```

times      —— 放大倍数。

```

**源程序:**这一组函数均为定义于头文件 hanenv.h 中的宏,参看第 14 章。

## 162. set\_alarm —— 设置定时器参数

**函数格式:**

```

int set_alarm(struct date date, struct time time, int type, unsigned (*fun)());

```

**功能说明:** 本函数用于设置一个由时钟触发的定时事件。在使用本函数之前,应该先使用函数 `run_clock` 安装时钟。

**参数设置:**

`date` —— 定时器事件启动日期;  
`time` —— 定时器事件启动时间;  
`type` —— 定时器事件类型。可以使用的值为  
`PER_SECOND` —— 定时器事件每秒钟激活一次;  
`PER_MINATE` —— 定时器事件每分钟激活一次;  
`PER_HOUR` —— 定时器事件每小时激活一次;  
`PER_DAY` —— 定时器事件每天激活一次;  
`PER_MONTH` —— 定时器事件每月激活一次;  
`PER_YEAR` —— 定时器事件每年激活一次;  
`ONCE_ONLY` —— 定时器事件只在指定时间激活一次。  
`fun` —— 定时器事件函数的入口地址。定时器事件函数由用户自己编写,其格式为

```
unsigned fun(unsigned h)
{
    .
    .
    .
}
```

参数 `h` 的值恒为 1(为触发器的触发条件)。

**返回值:** 定时器事件号码(可供删除定时器事件用)。

**参 阅:** 函数 `del_alarm`。

**源 程 序:** 见程序 9-6。

### 163. `set_asc_mode` —— 设置全角 ASCII 码输入法

**函数格式:**

```
void set_asc_mode(void);
```

**功能说明:** 见函数 `free_han_mode` 的说明。

### 164. `set_clock` —— 设置时钟的显示参数

**函数格式:**

```
void set_clock(int col,int line,int xtimes,int ytimes,int color,int bk);
```

**功能说明:** 本函数用于设置时钟的显示参数。

**参数设置:**

`col,line` —— 时钟显示位置,其中 `col` 的单位为字节。省缺值均为 0;  
`xtimes,ytimes` —— 时钟字符的横向和纵向放大倍数,省缺值均为 1;  
`color` —— 时钟字符颜色,省缺值为浅灰;  
`bk` —— 时钟背景色,省缺值为黑色。

**源 程 序:** 见程序 9-6。

## 165. set\_color —— 改变显示颜色

函数格式:

```
void set_color(int color_no,int red,int green,int blue);
```

功能说明:见函数 get\_color 的说明。

## 166. setcursor —— 设置光标参数

函数格式:

```
void setcursor(int width,int high,int line,int speed,int color);
```

功能说明:设置光标的大小和颜色。

参数设置:

width —— 光标宽度,单位为字节,省缺值为 1;

high —— 正文行高度,单位为线,省缺值为 CHAR\_HIGH;

line —— 光标线数,即光标的实际高度。光标总是显示在字符的下部。省缺值为 2;

speed —— 光标闪烁速度,数值越小速度越快,最小为 0。省缺值为 1;

color —— 光标颜色,省缺值为白色。注意光标实际上是以异或方式写到屏幕上的,所以其实际颜色由这里设置的光标颜色和屏幕背景颜色共同决定。

源程序:见程序 9-2。

## 167. \_SetDestAddr —— 设置目标数据块地址

函数格式:

```
void _SetDestAddr(long addr,unsigned size);
```

功能说明:参阅 \_GetEMMsize。

## 168. set\_double\_press\_time —— 设置鼠标按钮双击速度

函数格式:

```
void set_double_press_time(int time);
```

功能说明:设置测试鼠标按钮双击时的时间间隔。

参数设置:

time —— 以毫秒为单位的时间间隔,省缺值为 400;

参 阅:函数 double\_press。

源程序:该函数实际上是定义于头文件 hanenv.h 中的宏,见第 14 章。

## 169—170. set\_font... —— 设置字模点阵输出参数

函数格式:

```
void set_font_direction(int direction);
```

```
void set_font_bk_style(int style);
```



**功能说明:**这两个函数用于设置使用 DrawFont 函数写点阵时的方向和虚实。

**参数设置:**

direction —— 字模点阵的显示方向,可以使用符号常数:  
           HORIZONTAL : 横向显示字模点阵(按行输出);  
           VERTICAL : 纵向显示字模点阵(按列输出);  
 style       使用实线还是虚线显示点阵,可以使用符号常数:  
           SOLIDLINE : 字模笔划用实线填充;  
           DOTTEDLINE: 字模笔划用虚线填充。

**源程序:**这两个函数实际上是定义于头文件 hanenv.h 中的宏,见第14章。

### 171. set\_han\_mode —— 设置汉字输入法

**函数格式:**

```
int set_han_mode (int altfn, char * modename, unsigned (* get_han)(),
                  char * codefile, int wherecodes);
```

**功能说明:**参阅 free\_han\_mode。

### 172—175. set\_mouse... —— 设置鼠标光标参数

**函数格式:**

```
void set_mouse_color(int color, int bk);
void set_mouse_pos(int x, int y);
void set_mouse_range(int left, int top, int right, int bottom);
void set_mouse_speed(int speed);
```

**功能说明:**该组函数用于设置鼠标工作参数。各函数的具体功能为

set\_mouse\_color —— 设置鼠标光标的颜色;  
 set\_mouse\_pos —— 设置鼠标光标的位置;  
 set\_mouse\_range —— 设置鼠标光标的活动范围;  
 set\_mouse\_speed —— 设置滚动条等函数中的鼠标响应速度。

**参数设置:**

color, bk —— 鼠标光标颜色和边缘颜色省缺值为黑边白箭头;  
 x, y —— 鼠标光标的坐标,省缺值为 320, 240;  
 left, top, right, bottom —— 鼠标光标活动范围,省缺值为 0, 0, 639, 479;  
 speed —— 鼠标响应延迟时间,缺省值为 200 毫秒。

**参 阅:**函数 delight\_mouse, get\_mouse\_range 和 till\_mouse\_pop 的说明。

**源程序:**函数 set\_mouse\_color 和 set\_mouse\_speed 是定义于 hanenv.h 中的宏。

### 176. set\_mouse\_KB —— 设置虚拟键盘显示参数

**函数格式:**

```
void set_mouse_KB (int x, int y, unsigned color, int left, int top,
                   int right, int bottom);
```

**功能说明:**该函数用于设置虚拟键盘的显示参数。

**参数设置:**

- x       —— 虚拟键盘显示位置的横坐标;
- y       —— 虚拟键盘显示位置的纵坐标;
- color   —— 虚拟键盘的颜色配置。可以使用定义于头文件 `hanenv.h` 中的下列符号常数:
  - BLUE\_BAR       : 蓝色键盘;
  - GREEN\_BAR      : 绿色键盘;
  - CYAN\_BAR       : 青色键盘;
  - RED\_BAR         : 红色键盘;
  - MAGENTA\_BAR    : 洋红键盘;
  - BROWN\_BAR      : 棕色键盘;
  - GRAY\_BAR       : 灰色键盘;
- left   —— 虚拟键盘活动范围的左边界;
- top     —— 虚拟键盘活动范围的上边界;
- right   —— 虚拟键盘活动范围的右边界;
- bottom—— 虚拟键盘活动范围的下边界。

**参 阅:**函数 `load_MKB` 和函数 `disableMKB` 的说明。

177—178. `set_prompt ...`——设置提示行参数

**函数格式:**

```
void set_prompt_style(int color,int bk,int style);
void set_prompt_line(int line);
```

**功能说明:**这两个函数用于设置提示行的工作状态。

**参数设置:**

- color,bk —— 提示行字符颜色和背景色,省缺值为浅灰底黑字;
- style     —— 提示行风格,可以使用下列符号常数:
  - WIN\_STYLE : WINDOWS 风格;
  - DOS\_STYLE : DOS 风格。
 省缺值为 WIN\_STYLE;
- line      —— 提示行在屏幕上的位置,省缺值为 450。一般情况下无需变动此值。但如果设置了屏幕高度较大的逻辑屏幕后,为保证提示行仍在屏幕下方,可以设置分割屏幕,并设置 line 为 -30。

**源 程 序:**这两个函数实际上是定义于头文件 `hanenv.h` 中的宏,见第 14 章。

179—180. `set_screen ...`——设置屏幕尺寸

**函数格式:**

```
void set_screen_high(int high);
void set_screen_width(int width);
```

**功能说明:**这两个函数用于设置逻辑屏幕的大小。通常屏幕尺寸为 640×480 像素,但 HANENV 系统允许设置一个更大的逻辑屏幕,此时实际屏幕就相当于逻辑屏幕中的一个窗口。可以使用鼠标控制窗口的移动,也可以在程序中控制窗口的移动(见程序 smooth\_scroll 的说明)。

**参数设置:**

high —— 逻辑屏幕的高,以像素为单位,应大于等于 480,省缺值为 480;

width —— 逻辑屏幕的宽,以字节为单位,应大于等于 80(640 像素),省缺值为 80;

**注 意:**这两个函数必需在函数 init\_hanenv 之前使用。

**参 阅:**函数 smooth\_scroll 和图 2-1:“逻辑屏幕与实际屏幕窗口的关系”。

**源 程 序:**这两个函数实际上是定义于头文件 hanenv.h 中的宏,见第 14 章。

181. \_SetSourAddr —— 设置源数据块地址

**函数格式:**

```
void _SetSourAddr(long addr,unsigned size);
```

**功能说明:**参阅 \_GetEMMsize。

182. set\_window —— 设置正文窗口参数

**函数格式:**

```
void set_window(int left,int top,int right,int bottom);
```

**功能说明:**参阅 get\_window。

183. \_SetWriteMode —— 设置写屏模式

**函数格式:**

```
void _SetWriteMode(int write_mode);
```

**功能说明:**改变当前写模式。此函数对各写汉字、画线、矩形等函数有效。

**参数设置:**

write\_mode —— 写模式。可以使用下列符号常数:

PUT\_MODE : 覆盖模式,值为 0;

AND\_MODE : 与模式,值为 0x08;

OR\_MODE : 或模式,值为 0x10;

XOR\_MODE : 异或模式,值为 0x18; 省缺值为覆盖模式。

**源 程 序:**见程序 1-9。

184. sizeofXMS —— 查询扩展存储器剩余大小

**函数格式:**

```
int _SizeofXMS(void);
```

**功能说明:**参阅 \_ChangeXMS。

**185. skip space —— 去掉字符串中所有的空格****函数格式:**

```
char * skip space(char * string);
```

**功能说明:** 去掉字符串中所有的空格。**参数设置:**

string —— 字符串。

**返回值:** 指向字符串的指针。**参 阅:** 函数 ltrim。**186. smooth \_scroll —— 平滑移动屏幕****函数格式:**

```
void smooth _scroll(int x,int y);
```

**函数说明:** 在大于实际屏幕的逻辑屏幕中移动当前屏幕窗口。**参数设置:**

x,y —— 当前屏幕左上角在逻辑屏幕中的坐标。

**参 阅:** 函数 set \_screen... 和图 2-1: “逻辑屏幕与实际屏幕窗口的关系”。**源 程 序:** 该函数实际上是定义于头文件 hanenv.h 中的宏, 见第 14 章。**187. space —— 生成空白字符串****函数格式:**

```
char * space (int len);
```

**功能说明:** 生成一个空白字符串。**参数设置:**

len —— 空白字符串的长度。

**返回值:** 指向所生成的空白字符串的指针。**188. SP \_list —— 查询区位码表****函数格式:**

```
void _Cdecl SP _list(int col,int line);
```

**功能说明:** 在屏幕指定位置显示一个代码表式提示窗口, 内容为区位码与汉字、全角符号的对照表(如图 5-4 所示), 供操作人员在屏幕上直接查看。**参数设置:**

col —— 区位码表左上角列坐标, 以字节为单位;

line —— 区位码表左上角行坐标, 以像素为单位;

**全局变量:**

\_TextColor —— 正文色;

\_Background —— 背景色;

\_TitleColor —— 标题文字色;

- \_TitleBk       —— 标题背景色;
- \_BoxLineColor —— 框的轮廓线色;
- \_BarColor      —— 框色;
- \_Xtimes        —— 所有字符或汉字的横向放大倍数;
- \_Ytimes        —— 所有字符或汉字的纵向放大倍数;

操作说明: 移动光条、翻页、退出等操作见函数 getcode 的操作说明。

### 189. split\_screen —— 分割屏幕

函数格式:

```
void split_screen(int high);
```

功能说明: 将屏幕分割为两部分, 上面的部分是活动的(逻辑屏幕), 下面的部分是一个固定部分(例如提示行和菜单)。

参数设置:

high —— 屏幕活动部分的高度。

注 意: 该函数需要和设置逻辑屏幕大小的函数配合才可能得到预期效果。本函数必须在函数 init\_hanenv 之前使用。

参 阅: 函数 set\_screen... 和图 2-1: “逻辑屏幕与实际屏幕窗口的关系”。

源 程 序: 该函数实际上是定义于头文件 hanenv.h 中的宏, 见第 14 章。

### 190. takefont —— 取汉字点阵

函数格式:

```
int takefont(unsigned h);
```

功能说明: 将汉字或全角字符的点阵由当前汉字库取到汉字字模缓冲区中供各种汉字显示函数使用(字符串显示函数中已经调用了这个函数)。

参数设置:

h —— 欲显示的汉字或全角字符的机内码。

返 回 值: 如果函数执行成功返回非零值, 否则返回零。

源 程 序: 见程序 7-3。

### 191. takehan —— 从字符串中取汉字机内码

函数格式:

```
unsigned char * takehan(unsigned char * string, unsigned * h);
```

功能说明: 从字符串中取汉字机内码或 ASCII 码。

参数设置:

string —— 字符串;

h —— 取到的统一键盘码。

返 回 值: 指向字符串中的下一个查找位置, 如果串中再没有汉字或字符则返回空指针。

程序举例:

```

/* -----
   函数 takehan 应用举例:
   ----- */

.....

while((p=takehan(s,&h))!=NULL)
{
    if(isSP(h))
        ...
    else
        ...
}

```

源程序:见程序 6-7。

192. `till_mouse_pop` —— 当鼠标指定键按下时,延迟至该键释放

函数格式:

```
void till_mouse_pop(int which_button);
```

功能说明:当鼠标指定键按下时,延迟至该键释放。

参数设置:

`which_button` —— 测试哪个鼠标按钮。可以使用符号常数:

`LEFT_BUTTON` : 鼠标左键;

`RIGHT_BUTTON` : 鼠标右键。

参 阅:函数 `delight_mouse`、`get_mouse_range` 和 `set_mouse...` 的说明。

193. `trim` —— 删除字符串尾部空格符

函数格式:

```
char * trim(char * string);
```

功能说明:参阅 `ltrim`。

194—195. `unget...` —— 将统一键盘码或字符串存入键盘缓冲区

函数格式:

```
void ungeth(unsigned h);
```

```
void ungetstr(char * string);
```

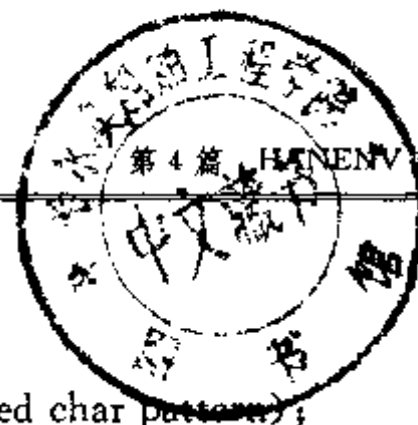
功能说明:函数 `ungeth` 用于将单个统一键盘码(如字符、功能键、鼠标按钮或汉字)送入键盘缓冲区,函数 `ungetstr` 用于将一个字符串送入键盘缓冲区。

参数设置:

`h` —— 欲送入键盘缓冲区的键码;

`string` —— 欲送入键盘缓冲区的字符串。

源程序:见程序 3-5 和程序 3-6。



## 196. \_V\_Line —— 画竖线

函数格式:

```
void _V_Line(int x,int y,int len,int color,unsigned char pattern);
```

功能说明:参阅 \_H\_Line。

## 参 考 文 献

- [1] T. Hogan. PC 软硬件技术资料大全. 清华大学出版社,1993
- [2] G. Suttly & S. Blair 著. EGA/VGA 程序员手册. 董士海等译. 北京大学出版社,1991
- [3] 求伯君. 新编深入 DOS 编程. 学苑出版社,1994
- [4] 刘路放. C 语言的汉字处理与图文数据库技术. 西安交通大学出版社,1995
- [5] 彭寿全等. 汉字信息处理. 电子科技大学出版社,1994
- [6] 郭平欣,张淞芝主编. 汉字信息处理技术. 国防工业出版社,1985
- [7] 尹彦芝编著. C 语言常用算法与子程序. 清华大学出版社,1991
- [8] 徐士良编著. PC 机 C 图形编程手册. 清华大学出版社,1994
- [10] 王天义. VGA/TVGA 卡的 VRAM 技术与应用. 中国计算机用户,1994. 7
- [11] 刘海等. 再谈“改变 VGA 设置的 16 种显示色”. 计算机世界月刊,1993. 9
- [12] 金永涛. VGA 的边缘颜色控制技巧. 电脑,1994. 11
- [13] 周彤. VGA 数模转换器编程. 计算机世界月刊,1993. 6
- [14] 吾鸣. 灵活编程属性控制器. 中国计算机用户,1994. 12
- [15] 李祥生. EGA/VGA/TVGA 像素级分屏平滑显示技术. 计算机世界月刊,1993. 5
- [16] 刘平西. EGA/VGA 平滑移动技术. 计算机世界月刊,1993. 9
- [17] 陈石. 安装特殊事件处理程序扩展鼠标驱动程序功能. 计算机世界月刊,1994. 3
- [18] 李汉生. VGA 上图象在 VRAM 中的移动. 计算机世界月刊,1994. 12
- [19] 金永涛. 充分利用键盘缓冲区. 计算机世界月刊,1994. 1
- [20] 尚海等. 如何在程序中使用 XMS. 计算机世界月刊,1994. 3
- [21] 朱炯波. 扩展内存使用技巧. 电脑技术,1995. 2
- [22] 曹晓光. “扩展内存”和“扩充内存”的误区. 计算机世界月刊,1993. 12
- [23] 言丹. 深入理解 PC 内存. 计算机世界,1995. 4. 19
- [24] 杨子云等. 超时使用 BIOS 1CH 中断的方法. 微型计算机应用,1994. 3

[ G e n e r a l   I n f o r m a t i o n ]

书名 = C 语言的窗口式图形界面设计——自带汉字环境的应用软件编程

作者 =

页数 = 3 9 2

S S 号 = 1 0 2 0 5 0 3 0

出版日期 =



封面
书名
版权
前言
目录
正文