汉字点阵字库原理

一、 汉字编码

1. 区位码

在国标 **GD2312—80** 中规定，所有的国标汉字及符号分配在一个 **94** 行、**94** 列的方阵中，方阵的每一行称为一个"区"，编号为 **01** 区到 **94** 区，每一列称为一个"位"，编号为 **01** 位到 **94** 位，方阵中的每一个汉字和符号所在的区号和位号组合在一起形成的四个阿拉伯数字就是它们的"区位码"。**区位码的前两位是它的区号，后两位是它的位号**。**用区位码就可以唯一地确定一个汉字或符号**，反过来说，任何一个汉字或符号也都对应着一个唯一的区位码。汉字"母"字的区位码是 **3624**，表明它在方阵的 **36** 区 **24** 位，问号"**?**"的区位码为 **0331**，则它在 **03** 区 **3l** 位。

2. 机内码

汉字的机内码是指在计算机中表示一个汉字的编码。机内码与区位码稍有区别。如上所述，汉字区位码的区码和位码的取值均在 **1~94** 之间，如直接用区位码作为机内码，就会与基本 **ASCII** 码混淆。为了避免机内码与基本 **ASCII** 码的冲突，需要避开基本 **ASCII** 码中的控制码**(00H~1FH)**，还需与基本 **ASCII** 码中的字符相区别。为了实现这两点，可以先在区码和位码分别加上 **20H**，在此基础上再加 **80H(**此处"**H**"表示前两位数字为十六进制数**)**。经过这些处理，用机内码表示一个汉字需要占两个字节，分别 称为高位字节和低位字节，这两位字节的机内码按如下规则表示：

高位字节 = 区码 + 20H + 80H(或区码 + A0H)

低位字节 = 位码 + 20H + 80H(或位码 + AOH)

由于汉字的区码与位码的取值范围的十六进制数均为 **01H~5EH(**即十进制的 **01~94)**，所以汉字的高位字节与低位字节的取值范围则为 **A1H~FEH(**即十进制的 **161~254)**。

例如，汉字"啊"的区位码为 **1601**，区码和位码分别用十六进制表示即为 **1001H**，它的机内码的高位字节为 **B0H**，低位字节为 **A1H**，机内码就是 **B0A1H**。

二、 点阵字库结构

1. 点阵字库存储

在汉字的点阵字库中，每个字节的每个位都代表一个汉字的一个点，每个汉字都是由一个矩形的点阵组成，**0** 代表没有，**1** 代表有点，将 **0** 和 **1** 分别用不同颜色画出，就形成了一个汉字，常用的点阵矩阵有 **12*12, 14*14, 16*16** 三种字库。

字库根据字节所表示点的不同有分为**横向矩阵**和**纵向矩阵**，目前多数的字库都是**横向矩阵**的存储方式**(**用得最多的应该是早期 **UCDOS** 字库**)**，**纵向矩阵**一般是因为有某些液晶是采用纵向扫描显示法，为了提高显示速度，于是便把字库矩阵做成纵向，省得在显示时还要做矩阵转换。我们接下去所描述的都是指横向矩阵字库。

2. 16*16 点阵字库

对于 **16*16** 的矩阵来说，它所需要的位数共是 **16*16＝256** 个位，每个字节为 **8** 位，因此，每个汉字都需要用 **256/8=32** 个字节来表示。

即每两个字节代表一行的 **16** 个点，共需要 **16** 行，显示汉字时，只需一次性读取 **32** 个字节，并将每两个字节为一行打印出来，即可形成一个汉字。

点阵结构如下图所示：

	第一字节								第二字节							
	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
0		■											■			
1			■	■	■		■	■	■	■	■	■	■	■		
2				■									■			
3	■						■						■			
4		■	■				■						■			
5			■				■					■				
6					■			■				■				
7				■				■			■					
8			■						■		■					
9	■	■	■							■						
10			■						■		■					
11			■					■				■				
12			■				■						■			
13			■			■							■	■	■	
14					■									■		
15																

3. 14*14 与 12*12 点阵字库

对于 **14*14** 和 **12*12** 的字库，理论上计算，它们所需要的点阵分别为**(14*14/8)=25, (12*12/8)=18** 个字节，但是，如果按这种方式来存储，那么取点阵和显示时，由于它们每一行都不是 **8** 的整位数，因此，就会涉到点阵的计算处理问题，会增加程序的复杂度，降低程序的效率。

为了解决这个问题，有些点阵字库会将 **14*14** 和 **12*12** 的字库按 **16*14** 和 **16*12** 来存储，即，每行还是按两个字节来存储，但是 **14*14** 的字库，每两个字节的最后两位是没有使用，**12*12** 的字节，每两字节的最后 **4** 位是没有使用，这个根据不同的字库会有不同的处理方式，所以在使用字库时要注意这个问题，特别是 **14*14** 的字库。

三、 汉字点阵获取

1. 利用区位码获取汉字

汉字点阵字库是根据区位码的顺序进行存储的，因此，我们可以根据区位来获取一个字库的点阵，它的计算公式如下：

点阵起始位置 = ((区码- 1)*94 + (位码 – 1)) * 汉字点阵字节数

获取点阵起始位置后，我们就可以从这个位置开始，读取出一个汉字的点阵。

2. 利用汉字机内码获取汉字

前面我们已经讲过，汉字的区位码和机内码的关系如下：

机内码高位字节 = 区码 + 20H + 80H(或区码 + A0H)

机内码低位字节 = 位码 + 20H + 80H(或位码 + AOH)

反过来说，我们也可以根据机内码来获得区位码：

区码 = 机内码高位字节 - A0H

位码 = 机内码低位字节 - AOH

将这个公式与获取汉字点阵的公式进行合并计就可以得到汉字的点阵位置。

发现这个不错的程序，可以读取显示汉字点阵字库，如果想做个电子书之类的东东，很有参考价值，程序虽然是用 TC20 写的，但是 DOS 时代的 C 和单片机的很相似的，主要的公式部分看明白就行了

```
/*********************************************************
 文件名  :All.c
 描 述  :12,14,16 点阵汉字显示文件
 语 言  :TC2.0
 作 者  :刘利国
 修 改  :
 日 期  :2002-11-5
 说 明  :需要 12,14,16 点阵汉字字库的支持
*********************************************************/
#include <stdio.h>
#include <graphics.h>
#include <fcntl.h>
#include <io.h>
int hzk_p,asc_p;
void open_hzk(char filename[]);
void open_asc(char filename[]);
void get_hz(char incode[],char bytes[],unsigned long uiSize );
void get_asc(char incode[],char bytes[],unsigned long uiSize );
void disasc(int x,int y,char code[],int color,
                                    unsigned int uiDianZhenSize);
void dishz(int x0,int y0,char code[],int color,
                                unsigned int uiDianZhenSize,
                                unsigned int uiDuoYuBit);
/*字模的大小
16*16 点阵   ZI_MO_SIZE=32
14*14 点阵   ZI_MO_SIZE=28
12*12 点阵   ZI_MO_SIZE=24
//该变量一定为 long,否则出错
*/
unsigned long  ZI_MO_SIZE[3]={32,28,24};
/*汉字点阵
16*16 点阵   ZI_MO_SIZE=16
14*14 点阵   ZI_MO_SIZE=14
12*12 点阵   ZI_MO_SIZE=12
*/
unsigned int  DIAN_ZHEN_SIZE[3]={16,14,12};
/*多余位数(对于 12*12 和 14*14 点阵字库,该位有意义)
16*16 点阵   ZI_MO_SIZE=0
14*14 点阵   ZI_MO_SIZE=2
12*12 点阵   ZI_MO_SIZE=4
```

```
*/
unsigned int  DUO_YU_BIT[3]={0,2,4};
#define ZOOMX    1  /*X 方向的放大倍数,1 为原始尺寸*/
#define ZOOMY    1  /*Y 方向的放大倍数,1 为原始尺寸*/

main()
{
      int x=10,i;
      int y=10;
      char *s;
      char *filename[3]={"e:\\hzk\\hzk16","e:\\hzk\\hzk14","e:\\hzk\\hzk12"};
      char *fileasc[3]={"e:\\hzk\\asc16","e:\\hzk\\asc14","e:\\hzk\\asc12"};
      char code[32];/*因为不能动态定位，所以取最大值 32*/
      char tmpcode[3]={0};
      unsigned char mask=0x80;
      int driver=DETECT,errorcode;
      int mode;
      int iOffset;
      initgraph(&driver,&mode,"");
      errorcode=graphresult();
      if(errorcode!=0)
      {
            printf("error
");
            getch();
            exit(1);
      }
      for(i=0;i<3;i++)
      {
            open_hzk(filename[i]);
            open_asc(fileasc[i]);
            s="北京惠 234 悦 12 通 234 电 s 子技术有限 234 公司";
            while(*s!=NULL)
            {
                  while(x<600 && (*s!=NULL))
                  {
                        tmpcode[0]=*s;
                        tmpcode[1]=*(s+1);
                        if(tmpcode[0] & mask)
                        {
                              get_hz(s,code,ZI_MO_SIZE[i]);
                              dishz(x,y,code,GREEN,DIAN_ZHEN_SIZE[i],DUO_YU_BIT[i]);
                         x+=20*ZOOMX;
                         s+=2;
```

```
                    }
                    else
                    {
                     disasc(x,y,s,LIGHTGREEN,DIAN_ZHEN_SIZE[i]);
                     x+=10*ZOOMX;
                     s+=1;
                    }
                }
                x=10;
                y+=20*ZOOMY;
            }
            y+=20;/*换行*/
            close(hzk_p);
            close(asc_p);
        }
        getch();
        closegraph();
}
/*******************************************************
函数名称：open_hzk
函数功能：打开字库文件
传入参数：无
返 回 值：无
建立时间：
修改时间：
建 立 人：
修 改 人：
其它说明：如果失败,则直接退出程序
*******************************************************/
void open_hzk(char filename[])
{
        hzk_p=open(filename,O_BINARY|O_RDONLY);
        if(hzk_p==-1)
        {
            printf("The file HZK not exits!
");
            getch();
            closegraph();
            exit(1);
        }
}
/*******************************************************
函数名称：open_asc
函数功能：打开字库文件
```

```
传入参数：无
返 回 值：无
建立时间：
修改时间：
建 立 人：
修 改 人：
其它说明：如果失败,则直接退出程序
*********************************************************/
void open_asc(char filename[])
{
    asc_p=open(filename,O_BINARY|O_RDONLY);
    if(asc_p==-1)
    {
        printf("The file asc not exits!
");
        getch();
        closegraph();
        exit(1);
    }
}
/*********************************************************
函数名称：
函数功能：
传入参数：
返 回 值：
建立时间：
修改时间：
建 立 人：
修 改 人：
其它说明：
*********************************************************/
void get_hz(char incode[],char bytes[],unsigned long uiSize )
{
    unsigned char qh,wh;
    unsigned long offset;
    qh=incode[0]-0xa0;
    wh=incode[1]-0xa0;
    offset=(94*(qh-1)+(wh-1))*uiSize;
    lseek(hzk_p,offset,SEEK_SET);
    read(hzk_p,bytes,uiSize);
}
/*********************************************************
函数名称：
函数功能：
```

传入参数：
返 回 值：
建立时间：
修改时间：
建 立 人：
修 改 人：
其它说明：
***/
```
void get_asc(char incode[],char bytes[],unsigned long uiSize )
{
    unsigned char qh,wh;
    unsigned long offset;
    offset=incode[0]*uiSize;
    lseek(asc_p,offset,SEEK_SET);
    read(asc_p,bytes,uiSize);
}
```
/***
函数名称：
函数功能：
传入参数：
返 回 值：
建立时间：
修改时间：
建 立 人：
修 改 人：
其它说明：对于 24*24 的点阵字库，存放格式如下：

纵向存放 3 个字节(24 位),横向存放 24 个字节,每个字模占 72 个字节

字符排列顺序如下:

1 4 7 10 .....

2 5 8 11 .....

3 6 9 12 .....

对于 16*16 的点阵字库，存放格式如下：

横向存放 2 个字节(16 位),其中第二个字节没有多余的数据

纵向存放 16 个字节,每个字模占 32 个字节

字符排列顺序如下:

1 2

3 4

5 6

.....

对于 14*14 的点阵字库，存放格式如下：

横向存放 2 个字节(16 位),其中第二个字节的后 2 位是多余的数据

纵向存放 14 个字节,每个字模占 28 个字节

字符排列顺序如下:

1 2

3 4

5 6

.....

对于 12*12 的点阵字库，存放格式如下：

横向存放 2 个字节(16 位),其中第二个字节的后 4 位是多余的数据

纵向存放 12 个字节,每个字模占 24 个字节

字符排列顺序如下:

1 2

3 4

5 6

.....

**********/

```
void dishz(int x0,int y0,char mat[],int color,
        unsigned int uiDianZhenSize,
                unsigned int uiDuoYuBit
        )
{
    unsigned char mask[]={0x80,0x40,0x20,0x10,0x08,0x04,0x02,0x01};
    register int x,y,row,col;
    register int byte;
    int zoomX,zoomY;/*x,y 方向*/
    x=x0;y=y0;
    byte=0;
    for(col=0;col<uiDianZhenSize;col++) /*列*/
    {
        for(zoomY=0;zoomY<ZOOMY;zoomY++)  /*纵向放大*/
        {
            for(row=0;row<uiDianZhenSize;row++) /*16 行(2 个 8 位)*/
            {
                for(zoomX=0;zoomX<ZOOMX;zoomX++) /*横向放大*/
                {
                    if((mask[byte%8]& mat[byte/8])!=NULL)
                    {
                        putpixel(x,y,color);
                    }
                    x++; /*x 位置加一*/
                }
                byte++;
            }
            byte+=uiDuoYuBit; /*去掉多余位,对于 16*16 点阵来说，该位为 0*/
            x=x0;
```

```
            y++;/*y 坐标加 1*/
            byte-=16; /*重新修正 byte,重复绘制 y 像点*/
        }
        byte+=16; /*每次写完 1 行(2 个字节，16 位),修正 byte*/
    }
}

void disasc(int x0,int y0,char code[],int color,
                        unsigned int uiDianZhenSize
                        )
{
    unsigned char mask[]={0x80,0x40,0x20,0x10,0x08,0x04,0x02,0x01};
    register int x,y,row,col;
    register int byte;
    int zoomX,zoomY;/*x,y 方向*/
    char mat[16]={0};/*因为不能动态定位，所以取最大值 16*/
    get_asc(code,mat,uiDianZhenSize);
    x=x0;y=y0;
    byte=0;
    for(col=0;col<uiDianZhenSize;col++) /*列*/
    {
        for(zoomY=0;zoomY<ZOOMY;zoomY++)  /*纵向放大*/
        {
            for(row=0;row<8;row++)
            {
                for(zoomX=0;zoomX<ZOOMX;zoomX++) /*横向放大*/
                {
                    if((mask[byte%8]& mat[byte/8])!=NULL)
                    {
                        putpixel(x,y,color);
                    }
                    x++; /*x 位置加一*/
                }
                byte++;
            }
            x=x0;
            y++;/*y 坐标加 1*/
            byte-=8; /*重新修正 byte,重复绘制 y 像点*/
        }
        byte+=8; /*每次写完 1 行(2 个字节，16 位),修正 byte*/
    }
}
```

你是否碰到过用启动盘启动系统后用 DIR 命令得到一串串莫名其妙的字符？有经验的朋友会告诉你：那是汉字。汉字？你不禁会问：怎么一个我一个也不认识。但那确确实实是汉字，如果你启动 UCDOS 或其他的汉字系统后，就会看到那是一个个熟悉的汉字。同样是汉字，为什么前后会看到不同的结果？呵呵，其实在电脑硬件中，根本没有汉字这个概念，也没有英文的概念，这铁玩意认识的概念只有——内码。

汉字的内码

点头表示什么？是"对"、"YES"，偏偏有的地方表示的意义却恰恰相反。一个动作，有不同的诠释；一个问题，有不同的答案；而一个符号，却有不同的意义，关键在于：你是如何地理解。在电脑中亦如此，所有的数据都是以 0 和 1 保存的，按不同的数据操作，可以得到不同的结果。对于显示英文操作，由于英文字母种类很少，只需要 8 位（一字节）即可。而对于中文，常用却有 5000 以上，于是我们的 DOS 前辈想了一个办法，就是将 ASCII 表的高 128 个很少用到的数值以两个为一组来表示汉字，即汉字的内码。而剩下的低 128 位则留给英文字符使用，即英文的内码。不信，你可以用记事本写一 C 文件：

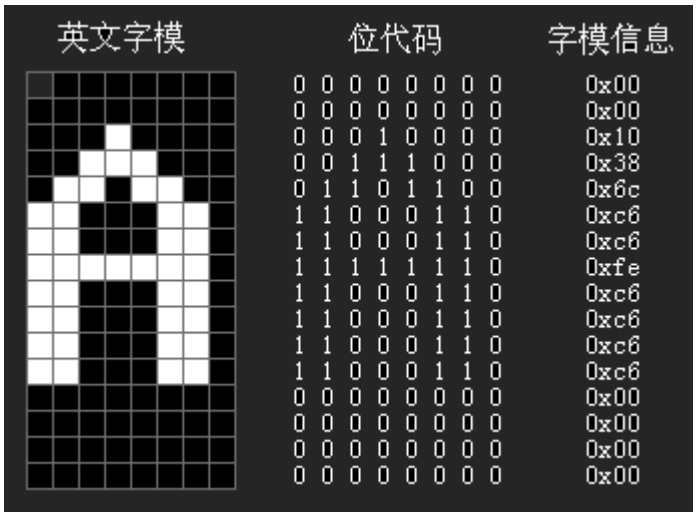
```
main()
```

```
{
    unsigned char *s,*e="ABcd",*c="你好";
    clrscr();
    printf("English char =");
    s=e;
    while(*s!=0) /*C 的字符串以 0 为结束符*/
    {
       printf("%3d,",*s);
       s++;
    }
    printf("\nChinease char=");
    s=c;
    while(*s!=0)
    {
       printf("%3d,",*s);
       s++;
    }
    getch();
}
```

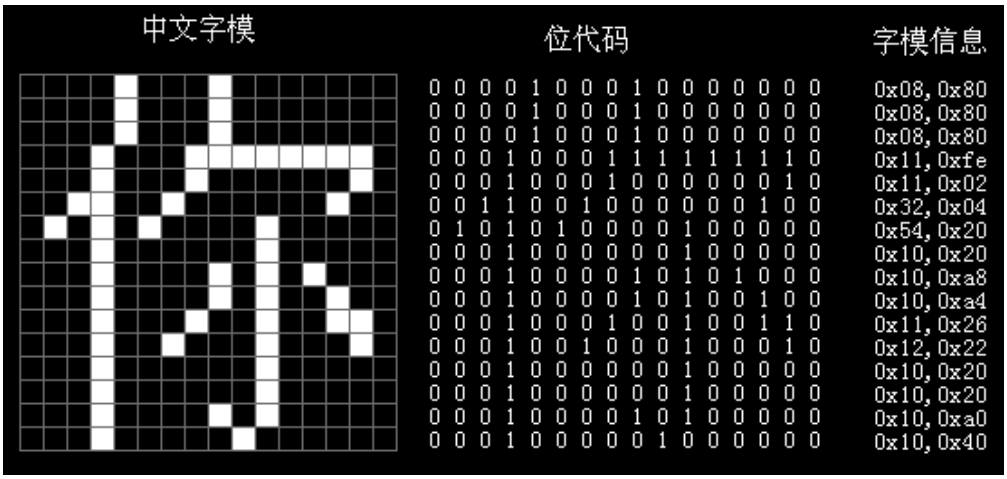
再用 TC 输入*.txt 打开运行，看见了没有，那些数值即英文和汉字的各字节内码。

汉字字模">汉字字模

得到了汉字的内码后，还仅是一组数字，那又如何在屏幕上去显示呢？这就涉及到文字的字模，字模虽然也是一组数字，但它的意义却与数字的意义有了根本的变化，它是用数字的各位信息来记载英文或汉字的形状，如英文的'A'在字模中是这样记载的：



而中文的"你"在字模中却是这样记载的：



在硬件系统内，英文的字模信息一般固化在 ROM 里，即使在没有进入系统的 CMOS 里，也可以让你看到英文字符。而在 DOS 下，中文的字模信息一般记录在 汉字库 文件 HZK16 里。

汉字库文件">汉字库**文件**

了解字母和汉字是按字模位信息显示的原理后，那如何得到汉字的字模信息呢？难道要我们自己去做？NO。DOS 前辈们经过艰辛的努力，将制作好的字模放到了一个个标准的库中以免去后辈的麻烦，这就是点阵字库文件。一般我们使用 16*16 的点阵宋体字库，所谓 16*16，是每一个汉字在纵、横各 16 点的区域内显示的。不过后来又有了 HZK12、HZK24，HZK32 和 HZK48 字库及黑体、楷体和隶书字库。虽然 汉字库 种类繁多，但都是按照区位的顺序排列的。前一个字节为该汉字的区号，后一个字节为该字的位号。每一个区记录 94 个汉字，位号则为该字在该区中的位置。因此，汉字在 汉字库 中的具体位置计算公式为：94*(区号-1)+位号-1。减 1 是因为数组是以 0 为开始而区号位号是以 1 为开始的。这仅为以汉字为单位该汉字在 汉字库 中的位置，那么，如何得到以字节为单位得到该汉字在 汉字库 中的位置呢？只需乘上一个 汉字字模 占用的字节数即可，即：(94*(区号-1)+位号-1)*一个 汉字字模 占用字节数，而按每种 汉字库 的汉字大小不同又会得到不同的结果。以 16*16 点阵字库为例，计算公式则为：(94*(区号-1)+(位号-1))*32。 汉字库 文该从该位置起的 32 字节信息即记录了该字的字模信息。

汉字库文件

了解 点阵汉字 及 汉字库 的构成原理后，显示汉字就变得简单。以 16*16 点阵字库为例，通常的方法是：将文件工作指针移到需要的 汉字字模 处、将 汉字库 文件读入一 2*16 数组再用 for 循环一位位地显示。以使用 VGAHI 模式显示"我"字为例，程序如下：

```
#include "graphics.h"
#include "stdio.h"
main()
{
    int i=VGA,j=VGAHI,k;
    unsigned char mat[16][2],chinease[3]="我";
    FILE *HZK;
    if((HZK=fopen("hzk16","rb"))==NULL)
```

```
        exit(0);
    initgraph(&i,&j,"");
    i=chinease[0]-0xa0;j=chinease[1]-0xa0;   /*获得区码与位码*/
    fseek(HZK,(94*(i-1)+(j-1))*32l,SEEK_SET);
    fread(mat,32,1,HZK);

    for(j=0;j<16;j++)
        for(i=0;i<2;i++)
            for(k=0;k<8;k++)
                if(mat[j][i]&(0x80>>k))   /*测试为 1 的位则显示*/
                    putpixel(i*8+k,j,WHITE);

    getch();
    closegraph();
    fclose(HZK);
}
```

怎么样？只要掌握了正确的方法，显示汉字并不复杂。

打印字库文件和 HZK12

如果你有 UCDOS 的 HZK24S（宋体）、HZK24K（楷体）或 HZK24H（黑体），你还可以使用不同字体的大字模汉字了。HZK24 系列是 24*24 的点阵字库，每字模占用 3*24 字节。如果你按照 HZK16 的显示方法的话，你会看到……呵呵，字被放倒了。这是因为该类字库与一般的 汉字库 不同，这类大字模 汉字库 是专供打印的打印字库，为了打印的方便将字模都放倒了，你使用时，只要将字模的位信息纵横转置显示即可。例如你如果定义为 mat[24][3]则应该这样输出：

```
    for(i=0;i<24;i++)
        for(j=0;j<24;j++)
            if((0x80>>i%8)&mat[j][i/8])   /*转置显示*/
```

putpixel(j+x,y+i,color);

还有一类字库 HZK12，虽然属于标准字库类型，但如果你将它的字模当作 12*12 位计算的话，根本无法正常显示汉字。因为字库设计者为了使用的方便，字摸每行的位数均补齐为 8 的整数倍，于是实际该字库的位长度是 16*12，虽然每行都多出了 4 位，但这 4 位都是 0（不显示），并不影响显示效果。