# 第9章　实用编程技巧

## 9.1　图形应用技巧

### 9.1.1　显示适配器类型的自动测试

目前PC机及兼容机的显示器及其适配器的类型非常多，有单色的，也有彩色的。这些显示器及适配器的模式对应用程序来说是非常重要的。如何在程序中自动识别显示器的模式，以便更好地使用当前的显示模式是每个微机应用程序开发者的一个重要课题。下面程序可以方便测出当前显示器适配器的模式（有关具体知识，请参见其它相关的技术书籍）。

[例9-1]　测试显示适配器类型。

```
#include  <stdio.h>
#include  <graphics.h>
#define   P(note)            printf(note)
#define   PV(format, value)  printf(format,value)
#define   PM                 printf("mode is ")
#define   PD                 printf("\n\tdetected graphics drive is")

void main( )
{
 int  gdrive , gerror , gmode ;

 detectgraph(&gdrive  , &gmode) ;   /* 标准测试函数 */
 if(gdrive<0)
   {P("No graphics hardware detected !\n");
   return ;
}
  switch (gdrive)
{case  1:  PD;
       P("CGA") ;
       switch(gmode)
         { case 0 :
             PM ;
             P("CGAC0 320X200") ;
             break ;
           case 1 :
             PM ;
             P("CGAC1 320X200") ;
             break ;
           case 2 :
             PM ;
```

```
                P("CGAC2 320×200") ;
                break ;
              case 3 : PM; P("CGAC3 640×200") ; break ;
              case 4 : PM; P("CGAh4 320×200") ; break ;
            }
          break ;
case 2: PD;
        P("MCGA")  ;
        switch(gmode)
          { case 0 : PM ; P("MCGAC0  320×200") ; break ;
            case 1 : PM ; P("MCGAC1  320×200") ; break ;
            case 2 : PM ; P("MCGAC2  320×200") ; break ;
            case 3 : PM ; P("MCGAC3  320×200") ; break ;
            case 4 : PM ; P("MCGAC4  620×200") ; break ;
            case 5 : PM ; P("MCGAC5  620×480") ; break ;
           }
        break ;
case 3 :  PD;
          P("EGA")  ;
          switch(gmode)
            { case 0 :PM ;
                      P("EGALO  640×200") ;
                      break ;
              case 1 :PM ;
                     P("EGALO  640×350") ;
                     break ;
            }
          break ;
case 4 :PD ;
        P("EGA64")  ;
        switch(gmode)
          {case 0 :PM ;
                  P("EGA64LO 640×200") ;
                  break ;
           case 1 : PM ;
                   P("EGA64HI 640×350") ;
                   break ;
          }
        break ;
case 5 :PD ;
        P("EGAMONO")  ;
        PM ;
        P("EGAMONO 640×350") ;
        break ;
   case 6 :PD ;
      P("IMB8614")  ;
      switch(gmode)
       {case 0 : PM ;
                P("IMB8514LO  640×480") ;
```

```
                break ;
         case 1 : PM ;
                P("IMB8514HI  1024×768") ;
                break ;
        }
       break ;
case 7 :PD ;
        P("HERCMONO")  ;
        PM ;
        P("HERCMONO 720×348") ;
        break ;
case 8 :PD ;
        P("ATT400")  ;
        switch(gmode)
          {case 0 : PM ;
                    P("ATT400C0 320×200") ;
                    break ;
           case 1 : PM ;
                    P("ATT400C1 320×200") ;
                    break ;
           case 2 : PM ;
                    P("ATT400C2 320×200") ;
                    break ;
           case 3 : PM ;
                    P("ATT400C3 320×200") ;
                    break ;
           case 4 : PM ;
                    P("ATT400CMCD 640×400") ;
                    break ;
          }
        break ;
case 9 :PD ;
        P("VGA")  ;
        switch(gmode)
          {case 0 : PM ;
                    P("VGALO 640×400") ;
                    break ;
           case 1 : PM ;
                    P("VGALO 640×350") ;
                    break ;
           case 2 : PM ;
                    P("VGALO 640×480") ;
                    break ;
          }
        break ;
case 10 :PD ;
         P("PC3270")  ;
         PM ;
         P("PC3270HI  720×350") ;
```

```
            break ;
        }
       P("\n\n\t\t\t  THANK YOU !");
      }
```

开发图形软件的基本方法

大家都知道，Turbo C 具有汇编语言那样直接控制系统硬件以及调用操作系统资源的功能，同时又具有一般高级语言完成复杂运算的能力。因此， C语言已成为开发图形软件最理想的程序语言。下面主要介绍几个生成基本图形的函数，它们是开发复杂图形软件的基础。

显示方式与色调函数

若要在屏幕上显示图形，首先要把屏幕设置为彩色图形显示方式，常用的方式是：

mode 4 320×200 4 色

在这种显示方式下，可以使用两种不同的配色器来配置色调，见表 9-1。

表9-1 屏幕色调

| 配色器号 | 颜 色 0 | 颜 色 1 | 颜 色 2 | 颜色 3 |
|---|---|---|---|---|
| 0 | 同底色 | 绿 | 红 | 黄 |
| 1 | 同底色 | 青 | 淡红 | 白 |

利用C语言标准库函数int86( )，可以方便地完成显示与色调的设置。这两个函数 setmode( ) 和palet( ) 见程序TX.c。例如，要求设置4号显示模式和使用0号配色器时，调用形式如下：

setmode(4)；

palet(0)；

画点函数和画线函数

在设置屏幕显示和色调后，就可使用各种绘图函数绘制不同颜色不同形状的图形。任何图形都有是由点组成的，所以画点函数是其它函数的基础。

在屏幕上画一个点有两种方法：一是调用 DOS功能，二是直接存取显示缓冲区（视频RAM）。由于后者有更高的执行速度，所以一般人都使用第二种方法。画点函数 point()中有三个参数：x、y、color，其中x、y分别是显示点的行和列坐标， color是点的颜色，取值0~3，对应的颜色如表9-1所示。该函数根据彩色图形显示的原理，直接控制视频 RAM区。它通过指针ptr访问该内存区域。指针初始化时，

```
char  far  ptr=(char far *)0xb8000000
```

视频RAM的首地址赋予了指针ptr，从而通过该指针就可以直接存取视频 RAM的任何单元。(针对不同的显示模式，此地址可能不同 )。

使用point()函数可以在屏幕的任意位置上显示指定颜色的一个点。例如在 20行15列上显示一个红色点时，使用：

```
point(20 , 15 , 2) ;
```

使用画点函数可以编写画直线的函数 line()，其原理是已知直线的两个端点坐标时，用迭代过程确定组成直线各点的位置。函数 line()的参数为x1，y1，x2，y2，color。其中x1、y1和x2、y2分别是直线的起点和终点坐标。 Color是直线的颜色，取值0~3。例如在屏幕的右斜对角上画一条绿色直线，使用的格式为：

line(0 , 0, 119 , 319 , 1) ;

**矩形与填充函数**

矩形是由四条直线组成的。使用直线函数可以矩形。绘制时，只需知道它左上角和右下角的坐标，就可以用直线函数绘出它的四条边，矩形函数 box()的参数为x1、y1、x2、y2、color。其中x1、y1和x2、y2分别是矩形左上角和右下角的坐标，color是四条边的颜色。

矩形填充块，实际是在指定位置上画出具有相同长度和颜色的直线，其填充函数 fillbox()中的参数与box()中的参数相同。

**绘制图形**

使用上述几个基本图形的函数，可以编写出在屏幕上绘制任意图形的程序。它使用键盘上的箭头等功能键，实现显示位置和颜色的控制。为了获取键盘扫描代码和显示当前绘图位置，要使用十字函数xhair()和获取键盘扫描代码函数getkey()。

下面给出一个简单的绘图程序tx.c。它相当于用画笔在屏幕上绘制图形，画笔的位置用十字光标显示。画笔的移动由↓、→、↑、←四个键控制。用 Home、PgUp、PgDn和End键分别控制十字光标向45°方向移动。画笔的抬起落下由字母键O控制，颜色由数字键0～3控制。功能键F1用于设定单步前进，F2用于设定5步前进。

该程序还可以画出矩形、填充矩形和直线。这时需要设置它们的坐标位置，直线需要两个端点坐标，矩形需要两个对角的坐标。例如，在抬笔状态下，把十字光标移至第一个位置后按回车键，然后再移至第二个位置按回车键。之后按下 L键时，则在设定的两个端点的位置上画出一条直线；如果按B键，则以两个位置为对角画出矩形；如果按F键，则画出填充矩形。另外，用P键可改变色调，按Q键结束运行，返回到DOS状态。

[例9-2] 绘图程序tx.c

```
#include<stdio.h>
#include<dos.h>
#include<ctype.h>
#include<conio.h>
#include<math.h>

void setmode(int);
void palet(int);
void point(int, int , int) ;
void line(int, int , int , int , int) ;
void box(int, int , int , int , int) ;
void fillboX(int, int , int , int , int) ;
void Xhair(int, int) ;
int getkey(void);

void main()
  {
   union{
      char c[2];
      int i;
     }key ;
   int X=10, y=10 , cc=2 , onflag=1 , palnum=1 ;
```

```
intX1=0 , y1=0 , X2=0 , y2=0 , firstpoint=1  ;
int d=1;
setmode(4)  ;
palet(0)  ;
Xhair(X , y) ;
  do {
  key.i=getkey()  ;
  Xhair(X , y) ;
  if(!key.c[0])
    switch(key.c[1]){
       case 75: if(onflag)   /*left*/
                line(X , y, X, y-d , cc) ;
              y-=d ;
              break ;
       case 77: if(onflag)  /* right */
                  line(X , y, X, y+d , cc) ;
            y+=d ;
            break ;
       case 72:if(onflag)  /* up */
                line(X , y, X-d , y, cc) ;
            X-=d ;
            break ;
       case 80: if(onflag)  /* down */
                 line(X , y, X+d , y, cc) ;
            X+=d ;
            break ;
       case 71:if(onflag)  /* Home-up left */
                line(X , y, X-d , y+d , cc) ;
            X-=d ;
            X-=d ;
            break ;
       case 73:if(onflag)  /*PgUp-up right */
                line(X , y, X-d , y+d , cc) ;
            X-=d ;
            y+=d ;
            break ;
       case 79:if(onflag)  /*End-down left */
                line(X , y, X+d , y-d , cc) ;
            X+=d ;
            y-=d ;
            break ;
       case 81:if(onflag)   /* PgUp-down right */
                line(X , y, X+d , y+d , cc) ;
            X+=d ;
            y+=d ;
            break ;
       case 59:     /*F1*/
            d=1 ;
            break ;
```

```
        case  60:   /*F2*/
              d=5 ;
              break ;
        }
        else
          switch(tolower(key.c[0])){
               case 'o':    /* brush on-off */
                 onflag=!onflag   ;
                 break ;
               case '1':    /* color 1*/
                 cc=1 ;
                 break ;
               case '2':   /* color2 */
                 cc=2 ;
                 break ;
               case '3':   /* color 3*/
                 cc=3 ;
                 break ;
               case '0':  /* color 0*/
                 cc=0 ;
                 break ;
               case 'b':  /* set boX */
                 boX(X1 , y1, X2, y2, cc) ;
                 break ;
               case 'f':  /*set fill boX */
                 fillboX(X1  , y1, X2, y2, cc) ;
                 break ;
               case 'l':  /* set line */
                 line(X1 , y1, X2, y2, cc) ;
                 break ;
               case 'r':  /*set endpoint */
                 if(firstpoint){
                      X1=X ;
                      y1=y ;
                      }
                   else {
                     X2=X ;
                     y2=y ;
                    }
                 firstpoint = !firstpoint;
                 break ;
               case 'p':   /* set color */
                 palnum = palnum==1 ? 2 :1;
                 palet(palnum)   ;
                 break ;
               }
Xhair(X , y) ;
}while(key.c[0]!='q')    ;
getch()  ;
```

```
    setmode(2)  ;
 }
/*设置显示方式 */
void setmode(mode)
   int mode;
   {
   union REGS regs;
   regs.h.al=mode   ;
   regs.h.ah=0  ;
   int86(0X10  , &regs , &regs) ;
   }
/*设置色调*/
void palet(pn)
   int pn;
   {
   union REGS regs;
   regs.h.bh=1  ;
   regs.h.bl=pn  ;
   regs.h.ah=11  ;
   int86(0X10  , &regs , &regs) ;
   }
/*画点函数*/
void point(X, y, color)
    int X, y, color ;
   {
   union {
       char cc[2];
       int i;
       }mask ;
   int i, indeX , posit ;
   unsigned char t;
   char Xor;
   char far *ptr=(char far *)0Xb8000000;
   mask.i=0Xff3f  ;
   if(X<0||X>199||y<0||y>319)
         return ;
   Xor=color&128  ;
   color=color&127  ;
   posit=y%4  ;
   color<<=2*(3-posit)   ;
   mask.i>>=2*posit   ;
   indeX=X*40+(y/4)   ;
   if(X%2)indeX+=8152   ;
   if(! Xor){
       t=*(ptr+indeX)&mask.cc[0]    ;
       *(ptr+indeX)=t|color   ;
      }
   else{
       t=*(ptr+indeX)|(char)0    ;
```

```
        *(ptr+indeX)=t^color    ;
        }
  }
/*直线函数*/
void line (X1, y1, X2, y2, color)
   int X1, y1, X2, y2, color ;
    {
   register int t, dis ;
   int Xerr=0, yerr=0 , dX, dy ;
   int incX, incy ;
   dX=X2-X1  ;
   dy=y2-y1  ;
   if (dX>0)incX=1;
      else if(dX==0)incX=0;
          else incX=-1;
   if(dy>0)incy=1   ;
      else if(dy==0)incy=0;
          else incy=-1;
   dX=abs(dy)  ;
   dy=abs(dy)  ;
   for (t=0; t<=dis+1  ; t++){
     point(X1  , y1, color) ;
     Xerr+=dX  ;
     yerr+=dy  ;
     if(Xerr>dis){
            Xerr-=dis  ;
            X1+=incX  ;
        }
     if(yerr>dis){
            yerr-=dis  ;
            y1+=incy  ;
        }
    }
  }
/*矩形函数*/
void boX(X1, y1, X2, y2, color)
   int X1, y1, X2, y2, color ;
   {
   line (X1, y1, X2, y1, color) ;
   line(X1  , y1, X2, y2, color) ;
   line(X1  , y2, X2, y2, color) ;
   line(X2  , y1, X2, y2, color) ;
   }
/*矩形填充函数*/
void fillboX(X1, y1, X2, y2, color)
   int X1, y1, X2, y2, color ;
   {
   register int i, begin , end ;
   begin=X1<X2? X1:X2;
```

```
    end=X1>X2? X1:X2;
    for (i=begin; i<=end ; i++)
        line(i , y1 , i, y2 , color) ;
     }
/* 十字光标定位函数 */
void Xhair(X, y)
     int X, y ;
    {
    line(X-4 , y, X+3 , y, 1|128) ;
    line(X , y+4 , X, y-3 , 1|128) ;
     }
/* 获取键盘扫描码函数 */
int getkey()
  {
    union REGS regs;
    regs.h.ah=0  ;
    return int86(0X16, &regs , &regs) ;
     }
```

### 9.1.2 屏幕图像的存取技巧

Turbo C提供了丰富的图形操作函数，利用这些函数可以很容易编写图形和图像处理程序，但是Turbo C没有提供屏幕图像存储和恢复的函数，而在许多情况下需要将屏幕上的图像全部或部分的以文件的形式保存在磁盘上，在需要时快速地从磁盘调入内存并重现在屏幕上。下面介绍的程序就是用来对屏幕上任一块矩形区域进行存取的程序 tx1.c。saveimage( )函数首先将屏幕上的一块矩形区域图像的数据写入内存某地址，然后创建一个二进制文件，并把该地址的数据写入此文件中。这样屏幕上的图像就以文件的形式存储在磁盘上了。 loadimage( )函数首先将磁盘上的图像数据文件打开并读入到内存某地址，然后从该地址将这些数据读到视屏缓冲区。这样，原先保存的图像就重现在屏幕上了。

tx1.c程序首先测试硬件的图形适配卡类型，并根据其值进入相应的图形方式，然后在屏幕上画一个彩色饼形统计图，并将该图像存入文件 graph.dat中，最后打开graph.dat文件并读入内存，将这幅图像重现在屏幕上。

[例9-3] 存取任意屏幕图像程序tx1.c

```
/* 存取任意屏幕图像头文件 tX1.h */
#include<stdio.h>
#include<graphics.h>
#include<conio.h>
#include<fcntl.h>
#include<alloc.h>
#include<stdlib.h>
#include<io.h>

 void saveimage(X1, y1, X2, y2, f)
 int X1, y1, X2, y2 ;
 char *f;
```

```
 {
    char *ptr;
    unsigned size;
     FILE *fn;
    size=imagesize(X1  , y1 , X2 , y2) ;
    ptr=malloc(size)   ;
    getimage(X1  , y1 , X2 , y2 , ptr) ;
    if ((fn=fopen(f, "wb"))==NULL) {
       restorecrtmode()   ;
   printf("Cannot create file %s\n",f) ;
   eXit(1) ;
     }
   fwrite(ptr  , size , 1, fn) ;
   fclose(fn)  ;
   free(ptr)  ;
    }

void loadimage(X1, y1 , f)
   int X1, y1 ;
   char *f;
     {
   char *ptr;
   unsigned size;
   FILE  *fn;
   if((fn=fopen(f  , "rb"))==NULL){
          restorecrtmode()   ;
   printf("Cannot open file %s\n",f) ;
   eXit(1) ;
      }
   size = 0;
   while(fgetc(fn) != EOF) size++;
   ptr=malloc(size)   ;
        rewind(fn)  ;
   fread(ptr  , size , 1, fn) ;
   fclose(fn)  ;
   putimage(X1  , y1 , ptr , COPY_PUT)  ;
   free(ptr)  ;
     }
/* 图像存取示例 */
void initialize(void);
void quit(void);
int X, y ;
char *f="graph.dat";

void main(void)
    {
   initialize()  ;
   outteXtXy(X/2  , -10 , "Saving Image");
```

```
    saveimage(0 , 0, X, y, f) ;
    outteXtXy(X+50 , y+y/2+10 , "Loading Image");
    loadimage(X , y/2 , f) ;
    quit( );
  }

void initialize( )
  {
    int gdrive=DETECT, gmode , errorcode ;
    int angle=360/MAXCOLORS, color ;
    initgraph(&gdrive , &gmode , "") ;
    errorcode=graphresult( );
    if (errorcode!=grOk)
     {
      printf("graphics error:%s\n",grapherrormsg(errorcode))    ;
      printf("press any key to halt:")
      getch( );
      eXit(1) ;
     }
    X=getmaXX() *2/5;
    y=getmaXy( ) *2/5;
    setviewport(X/2 , y/2 , getmaXX( )-X/2, getmaXy( )-y/2, 0) ;
    setteXtjustify( CENTER_TEXT,CENTER_TEXT) ;
    rectangle(0 , 0, X, y) ;
    for (color=0; color<MAXCOLORS ; color++){
           setfillstyle(SOLID_FILL , color) ;
           pieslice(X/2 , y/2 , color*angle , (color+1)*angle , y/3) ;
        }
      }

void quit( )
    {
    getch( );
    closegraph( );
    }
```

### 9.1.3 屏幕显示格式的控制方法

**一个良好的屏幕格式能给操作者提供很大的方便，也给人们一种赏心悦目的感觉。**

**为了控制屏幕显示格式，需要编写两个屏幕控制。我们可以借助这两个函数，设计出用户所需要的屏幕显示格式。**

**下面是一个简单的演示程序。程序中借助这两个函数，在屏幕中间显示变动的数字。**

[例9-4] **控制显示格式** tx3.c

```
# include<stdio.h>
# include<bios.h>
#include<dos.h>
```

```
void cls(int);
void gotoXy(int, int) ;

void cls(line)
   int line;
  {
    union REGS in, out ;
    in.X.aX=0600  ;
    in.X.cX=0000  ;
    in.h.dh=line-1  ;
    in.h.dl=79  ;
    in.h.bh=07  ;
    int86(0X10  , &in , &out) ;
   }
 void gotoXy (X, y)
     int X, y ;
   {
   union REGS in, out ;
   in.h.dh=X  ;
   in.h.dl=y  ;
   in.h.ah=02  ;
   in.h.bh=0  ;
   int86 (0X10, &in , &out) ;
   }
   void main( )
  {
    int i;
    gotoXy(0 , 0) ;
   cls(11)  ;
   gotoXy(4 , 20) ;
   printf("-----------------------")      ;
   gotoXy(5 , 20) ;
   printf("|proceeding record No: |;")
   gotoXy(6 , 20) ;
   printf("-----------------------")      ;
   for (i=1; i<=1000  ;i++)
          {
           gotoXy(5  , 45) ;
           printf("%4d"  , i) ;
           }
    }
```

### 9.1.4 使图形软件脱离BGI的方法

**大家知道，用Turbo编译的图形软件，在运行时，当前目录下必须要有相应的 BGI文件。例如CGA.BGI、EGAVGA.BGI等。这对于应用程序是不太方便的。为了解决这个问题，可使用以下方法。**

1) **用Turbo C提供的BTIOBJ.EXE把*.BGI编译成目标文件*.OBJ。例如：**

```
C> BGIOBJ  CGA
```

这样就产生了一个CGA.OBJ。同样，将EGAVGA.BGI进行编译，再把这些*.OBJ拷入TC目录下的LIB 子目录中。

使用时，先编译一个 project文件，把需要的 OBJ文件列入 project文件中。例如对BGIDEMO .C，相应BGIDEMO.PRJ可写为：

```
BGIDEMO.C
EGAVGA.OBJ
```

这样在集成环境下调试BGIDEMO.C时，当前目录下不需要有EGAVGA.BGI，最后生成的EXE文件，运行时也不需要EGAVGA.BGI。

2) 对命令行编译TCC.EXE只要在编译时列入相应的EGAVGA.BGI，最后生成的EXE文件。运行时也不需要EGAVGA.BGI的EXE文件。

3) 对于经常使用TCC.EXE的用户，可用Turbo C提供的TLIB.EXE将上述*.obj扩充进图形库GRAPHICS.LIB。方法为：

```
C> TLIB  GRAPHICS.LIB +EGAVGA.OBJ
```

这样，以后在编译时，只要联贯新的GRAPHICS.LIB就可编译出不需要BGI的图形软件。

### 9.1.5 拷贝屏幕图形的方法

在图形方式下，有时需将屏幕信息在打印机上输出。为了输出屏幕图形，要有一个内存驻留程序。这时要考虑内存驻留程序的激活问题，激活时机的控制以及 TSR的初始化问题。

所有TSR程序都靠键来激活，因此需用自己的键盘中断程序代替 DOS的键盘中断程序，激活TSR程序是在DOS不忙时进行的。当DOS正在使用时，有一个字节被置1，当它未被使用时，该字节为0，这个地址可用34H号中断获取，该中断返回后， ES寄存器存放段地址， BX寄存器放位移，因而，当该字节为0时，TSR程序才允许激活。

实现时，使用了interrupt类型说明、寄存器伪变量和程序终止并驻留的技术。

首先由main( )函数完成初始化工作，它先取得中断 5子程序的地址，然后设置新的 5号中断程序，这由getvect和setvect来实现的。最后用keep(0，size)使程序驻留，并保存16*size大小的数据空间。

interrupt类型说明符允许我们编写中断处理程序，说明该类型的函数进入时，会自动保存各寄存器的值，退出时恢复各寄存器的值，新的中断处理程序先判断当前的显示方式，若是文本方式，则执行老中断程序，若是图形方式，就执行新的图形拷贝程序。

每当发生中断调用时， DOS转移到一个内部很小的数据栈上工作，为了确保程序能正常工作，必须建立起自己的数据栈，用寄存器伪变量 SP和SS实现。打印完屏幕后，恢复原环境(此程序针对EPSON LQ系列打印机)。

[例9-5] 屏幕图形硬拷贝程序tx4.c

```
#include <dos.h>
#include<stdio.h>
#include<graphics.h>
#include<bios.h>

#define STK_SIZE 0X1000
```

```
#define print(ch)    biosprint(0,ch, 0)

void set_graphics(int cols);
void print_scr(int X1, int y1, int X2, int y2);
char video_mode(void);
void interrupt new_int5(void);
void interrupt (*old_int5)();
unsigned char stack [STK_SIZE];
unsigned sp, ss ;
main()
{
   union REGS r;
   struct SREGS s;
   old_int5=getvect(5)   ;
   keep(0 , 2000) ;
   return 0;
}
void interrupt new_int(void)
{
   char vmode;
   vmode=video_mode()   ;
   if((vmode==2)|(vmode==3)|(vmode==7))
      setvect(5  , old_int5)  ;
     else{
          disable()  ;
          ss=_SS  ;
          sp=_SP  ;
          _SS=_DS  ;
          _SP=(unsigned)&stack[STK_SIZE-2]      ;
          enable( );
          print_scr(0  , 0, 639 , 349) ;
          disable()  ;
          _SP=sp  ;
          _SS=ss  ;
          enable( ); }
}
void print_scr(int X1, int y1, int X2, int y2)
   {
   register int i, X, y, pX ;
   int cols, color , sum ;
   X2++ ;
   y2++ ;
   cols=X2-X1  ;
   for(y=y1  ; y<y2 ; y+=8){
      set_graphics(cols)   ;
          for(X=X1  ; X<=X2 ; X++){
             sum=0 ;
             for(i=0  ; i<8 ; i++){
```

```
                if(y+i<y2){
                 color=getpiXel(X   , y+i) ;
                 if(color)sum+=1<<(7-i)    ;
                 }
               }
              print(sum)  ;
            }
        printf("\n")  ;
       }
    }

     void set_graphics(int cols)
      {
        char den_code;
        union aa{
       unsigned char c[2];
       unsigned int i;
        }u ;
        u.i=cols  ;
        print(27)  ;
        print(65)  ;
        print(8)  ;
        print(27)  ;
        print(76)  ;
        print(u.c[0])  ;
        print(u.c[1])  ;
     }
  char video_mode(void)
    {
     union REGS r;
     r.h.ah=15  ;
     return int86(0X10, &r, &r)&255  ;
    }
```

### 9.1.6 随意改变VGA显示器显示颜色的技巧

VGA显示适配器是一种使用很普遍的高性能图形适配器，最多有 261244（64×64×64）种颜色，可同时使用其中的任意种。但是目前来说，普遍使用的仍是非曲直 6种颜色的显示模式。

若仅仅用现有的16种颜色编制图形软件或窗口软件，画面则显得单调，能否根据需要自由设置这16种颜色呢？答案是肯定的。在显示器上某一色号所显示的颜色仅由显示卡上的DAC颜色寄存器中的值决定。DAC颜色寄存器是一个18位的寄存器，红、绿、蓝各占六位，卡上共有256个这样的寄存器，分别对应于256个色号。开机时的16种颜色（即0~15号）被设置成如下寄存器及比色：

| 色号 | 对应寄存器号码 | 红 | 绿 | 蓝 |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |

| | | | | |
|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 42 |
| 2 | 2 | 0 | 42 | 0 |
| 3 | 3 | 0 | 42 | 42 |
| 4 | 4 | 42 | 0 | 0 |
| 5 | 5 | 42 | 0 | 42 |
| 6 | 20 | 42 | 21 | 0 |
| 7 | 7 | 42 | 42 | 42 |
| 8 | 56 | 21 | 21 | 21 |
| 9 | 57 | 21 | 21 | 63 |
| 10 | 58 | 21 | 63 | 21 |
| 11 | 59 | 21 | 63 | 63 |
| 12 | 60 | 63 | 21 | 21 |
| 13 | 61 | 63 | 21 | 63 |
| 14 | 62 | 63 | 63 | 21 |
| 15 | 63 | 63 | 63 | 63 |

如果改变这16个寄存器中的值，即可改变在屏幕上显示的16种颜色，而对程序运行没有任何其它影响。

具体实现可以通过调用VGA BIOS中断进行，也可以通过VGA寄存器编程实现。在西文方式下以上两种方法都可以使用，但在中文系统下，由于中文系统修改了视频中断10H，因此只能通过VGA寄存器编程实现。

下面两个程序，一个用于设置颜色，另一个用于检查设置。设置的颜色可从261244种颜色中任意设定0~15号颜色。在程序中：red，green，blue取值为0~63；colornum为要改变颜色所对应的寄存器号。使用的格式为：

```
setcolor  <寄存器中>  <红>  <绿>  <蓝>
getcolor  <寄存器号>
```

例如，对亮绿色（10号）进行改色，可在DOS提示符下，键入：

```
setcolor   58   35   25   15
```

回车后，10号码色将成为由红色35、绿色25、蓝色15、调成的新颜色。若要检查10号的设置，可在DOS提示符下，键入：

```
getcolor   58
```

回车后，屏幕上将出现寄存器号、红、绿、蓝的颜色值。

[例9-6] 设置新色彩setcolor.c。

```
/* 格式：setcolor< 寄存器中>< 红 > < 绿 > < 蓝 >*/
   #include<dos.h>
   #include<stdlib.h>
   #include<stdio.h>

   void main(int argc, char *argv[])
   {
     int  colornum, read0 , green0 , blue0 ;
     union REGS r;
```