第 4 章　C 语言程序设计基本技术

4.1　图形状态显示原理

Turbo C 2.0 默认显示方式为文本方式，这种显示方式就是平时看到的情况。但是图形方式和文本方式不同，在图形方式下可以画图，它的显示单位是像素。显示器显示的图形是由一些圆点组成的（其亮度、颜色不同），这些点被称为像素。满屏显示像素多少，则决定了显示的分辨率高低，可以看出像素越小（或个数越多），则显示的分辨率越高。像素在屏幕上的位置则由其所在的（x，y）坐标值决定。

显示屏的图形坐标系统就像一个倒置的直角坐标系，如图 4-1 所示。

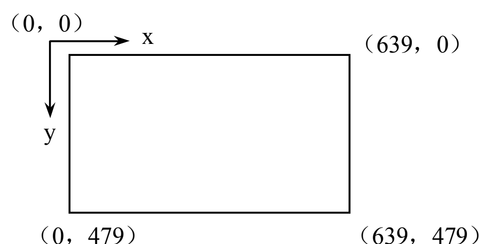

图 4-1　图形绝对坐标系统

定义屏幕的左上角为原点，正 x 轴向右延伸，正 y 轴向下延伸，即（x，y）坐标值均为非负整数，但它的最大值则由显示器的类型和显示方式来决定，即显示的像素大小可以通过设置不同的显示方式来改变。比如在图 4-1 所示的显示方式下，（x，y）最大坐标分别是（639，399），即满屏显示的像素个数为 640×400，则称这种相对屏幕显示的坐标叫物理坐标或绝对坐标。还有一种坐标叫相对坐标，它的参照物是图视窗口。图视窗口就是指在物理坐标区间又开辟一个或多个区间，在这些区间又可定义一个相对坐标系统，这样以后画图均可在此区间进行，并以相对坐标来定义位置。例如，在图 4-1 所示的显示方式下定义了一个左上角坐标为（200，50），右下角坐标为（400，150）的一个区域为图视窗口，则以后处理图形时，就以其左上角为坐标原点（0，0），右下角为坐标（200，100）的坐标系来定位图形上各点位置。

Turbo C 2.0 为用户提供了一个功能很强的画图软件库，它又被称为 Borland 图形接口（BGI），它包括图形库文件（graphics.lib）、图形头文件（graphics.h）和许多图形显示器（图形终端）的驱动程序。还有一些字符集的字体驱动程序。编写图形程序时用到的一些图形库函数均在 graphics.lib 中，执行这些函数时，所需的有关信息（如宏定义等）则包含在 graphics.h 头文件中。因此用户在自己的画图源程序中必须包括 graphics.h 头文件，在进行目标程序连接时，要将 graphics.lib 连接到自己的目标程序中去。

由于计算机画图涉及显示器和驱动它们工作的图形适配器（卡）等许多硬件知识，因此在这里有必要简单地介绍一下。

4.1.1 图形适配器

计算机中显示的字符和图形均以数字形式存储在存储器中，而显示器接收的是模拟信号。插在PC插槽中的图形卡的作用就是将要显示的字符和图形以数字形式存储在卡上的视频存储器 VRAM 中，再将其变成视频模拟信号送往相应适配的显示器显示，这表明适配器在计算机主机和显示器之间起到了信息转换和视频发送作用，但是由于显示器种类不同，因而适配器种类也就不同，同样不同适配器又可支持不同的分辨率显示方式、文本显示方式和颜色设置。因此不同的适配器导致了不同的图形模式，其中常用的适配器主要有下面 3 种：

（1）彩色图形适配器（CGA）。

这是 PC/XT 等微机配用的图形卡，它可以产生单色或彩色字符和图形。在图形方式下，Turbo C 2.0 支持两种分辨率供选择：一种为高分辨方式（CGAHI），像素数为 640×200，这时背景色是黑的（当然也可重新设置），前景色可供选择，但前景色只是同一种，因而图形只显示两色；另一种为中分辨显示方式，像素数为 320×200，其背景色和前景色均可由用户选择，但仅能显示 4 种颜色。在该显示方式下，有 4 种模式供选择，即 CGAC0、CGACl、CGAC2 及 CGAC3，它们的区别是显示的 4 种颜色不同。

（2）增强型图形适配器（EGA）。

该适配器除支持 CGA 的 4 种显示模式外，还增加了 Turbo C 2.0 称为 EGALO（EGA 低分辨显示方式，分辨率为 640×200）的 16 色显示方式，和 640×350 的 EGAHI（EGA 高分辨显示方式，分辨率为 640×350）的 16 色显示方式。

（3）视频图形阵列适配器（VGA）。

它支持 CGA 和 EGA 的所有显示方式，但自己还有 640×480 的高分辨显示方式（VGAHI）、640×350 的中分辨显示方式（VGAMED）和 640×200 的低分辨显示方式（VGALO），它们均可有 16 种显示颜色可供选择。

由于众多生产厂家推出了许多性能优于 VGA 但名字各异的图形显示系统，美国标准协会为此制定了主要性能标准，常将属于这类的显示适配卡统称为 SVGA。目前基本上使用的都是属于 SVGA，它也可以使用 VGA 卡方式进行编程。

4.1.2 显示器工作方式

显示器有两种工作方式，即文本显示方式和图形显示方式。它们的主要差别是选择文本方式时，VRAM 存放要显示字符的 ASCII 码值，并用它作为地址，然后取出字符发生器 ROM（固定存储器）中存放的相应字符的图像（又称字模），变成视频信号在显示器屏上进行显示。EGA、VGA 可以使用几种字符集，如 EGA 下有 3 种字符集，VGA 有 5 种字符集。而当选择图形方式时，则要显示的图形的图像直接存在 VRAM 中，VRAM 中某地址单元存放的数就表示了相应屏幕上某行和列上的像素及颜色。例如，在 CGA 的中分辨图形方式下，每字节代表 4 个像素，即每 2 位表示一个像素及颜色。

4.1.3 图形系统的初始化与关闭

在进入图形方式前，首先要在程序中对使用的图形系统进行初始化，即要选择采用什么类型的适配器驱动程序，采用什么模式的图形方式，以及该适配器驱动程序的寻找路径名。注意所用系统的适配器一定要支持所选用的显示模式；否则将出错。当图形系统初始化后，才可

以进行画图操作。

1．图形系统初始化函数

Turbo C 2.0 提供的 initgraph 函数可完成图形系统初始化的功能。其原型是：

```
void far initgraph(int far *driver,int far *mode,char far *path_for_driver);
```

当使用的存储模式为 tiny（微型）、small（小型）或 medium（中型）时，不需要远指针，因而可以将初始化函数的调用格式改成以下形式：

```
initgraph(&driver,&mode,"");
```

其中驱动程序目录路径为空字符" "时，表示在当前目录下，参数 mode 的用法如表 4-1 所示。

表 4-1　参数 mode 对应表

适配器 Driver	模式 Mode	分辨率	颜色数	页数	标识符
CGA	0	320×200	4	1	CGA0
	1	320×200	4	1	CGA1
	2	320×200	4	1	CGA2
	3	320×200	4	1	CGA3
	4	640×200	2	1	CGHI
EGA	0	640×200	16	4	EGAL0
	1	640×350	16	2	EGAHI
EGA64	0	640×200	16	1	EGA64L0
	1	640×350	4	1	EGA64HI
EGAMONO	0	640×350	2	1	EGAMONOHI
VGA	0	640×200	16	2	VGAL0
	1	640×350	16	2	VGAMED
	2	640×480	16	1	EGAHI
MCGA	0	320×200	4	1	MCGA0
	1	320×200	4	1	MCGA1
	2	320×200	4	1	MCGA2
	3	320×200	4	1	MCGA3
	4	640×200	2	1	MCGAMED
	5	640×280	2	1	MCGAHI
HREC	0	720×348	2	1	HRECMONOH
ATT400	0	320×200	4	1	ATT4000
	2	320×200	4	1	ATT4001
	3	320×200	4	1	ATT4002
	4	320×200	4	1	ATT4003
	5	640×200	2	1	ATT400MED
	5	640×400	2	1	ATT400HI
PC3270	0	720×350	2	1	PC3270HI
IBM8514	0	640×480	25		
	1	1024×768	6		

参数 driver 是一个枚举变量，它属于显示器驱动程序的枚举类型：

enum driver {DETECT,CGA,MCGA,EGA,EGA64,EGAMONO,IBM 8514,HERCMONO, ATT400,VGA,PC3270};

其中枚举成员的值顺序为：DETECT 为 0，CGA 为 1，依次类推。当不知道所用显示适配器名称时，可将 driver 设成 DETECT，它将自动检测所用显示适配器类型，并将相应的驱动程序装入系统，将其最高显示模式作为当前显示模式，如表 4-2 所示。

表 4-2 不同适配器的显示模式

检测到的适配器	选中的显示模式
CGA	4（640×200，2 色，即 CGAHI）
EGA	1（640×350，16 色，即 EGAHI）
VGA	2（640×480，16 色，即 VGAHI）

例 4-1 下面是一般画图程序的开始部分，它包括对图形系统的初始化。

```
#include <graphics.h>
main()
{  int driver=DETECT;
   int mode;
   initgraph(&driver,&mode,"");
   …
}
```

在本例初始化过程中，先由 DETECT 检测所用适配器类型，并将当前目录下相应的驱动程序装入系统，并采用最高分辨率显示模式作为 mode 的值。

若已知所用图形适配器为 VGA 时，想采用 640×480 的高分辨显示模式 VGAHI，则图形初始化部分可改写为：

```
int  driver=VGA;
int  mode=VGAHI;
initgraph(&driver,&mode," ");
```

2. 图形系统检测函数

当 driver=DETECT 时，实际上 initgraph 函数又调用了图形系统检测函数 detectgraph，由它来完成对适配器的检查，并得到显示器类型号和相应的最高分辨率模式，若所设适配器不是规定的那些类型，则返回-2，表示适配器不存在，该函数的原型是：

```
void far detectgraph(int far *graphdriver,int far *graphmode);
```

3. 清屏和恢复显示方式的函数

画图前要清除屏幕，因此必须使用清屏函数。清屏函数的原型是：

```
void far cleardevice(void);
```

该函数作用范围为整个屏幕，如果用函数 setviewport 定义一个图视窗口，则可采用清除图视窗口函数，它仅清除图视窗口区域内的内容，该函数的原型是：

```
void far clearviewport(void);
```

当画图程序结束后，系统要回到文本方式，这时应该关闭图形系统，关闭图形系统的函数原型是：

```
void far closegraph(void);
```

由于进入 Turbo C 2.0 环境编程时，即已经进入文本方式，因而为了在画图程序结束后恢

复原来的最初状况，一般在画图程序结束前调用该函数，使其恢复到文本方式。另外，为了不关闭图形系统，使相应适配器的驱动程序和字符集（字库）仍驻留在内存，但系统又返回到原来所设置的模式下，就可用恢复工作模式函数进行清屏操作，它的原型是：

void far restorecrtmode(void);

该函数常和另一设置图形工作模式函数 setgraphmode 交互使用，使得显示器工作方式在图形和文本方式之间来回切换，这在编制菜单程序和说明程序时很有用处。

4.2　基本绘图方法

4.2.1　基本绘图函数

由于图形是由点、线、面 3 个成分组成的，因此 Turbo C 2.0 提供了一些函数来实现这些操作，其中所谓的面可由对封闭图形填色来实现。当图形系统初始化后，要进行的画图操作均可采用默认值作为参数的当前值。

1. 画点函数

void far putpixel(int x,int y,int color);该函数表示在指定的（x，y）处画一点，点的颜色由设置的 color 值来决定，关于颜色的设置，将在设置颜色函数中介绍。

int far getpixel(int x,int y);该函数与 putpixel()相对应，它得到在（x，y）点位置上的像素的当前颜色值。

2. 有关坐标位置的函数

void far moveto(int x,int y);该函数移动画笔到指定的（x，y）位置，移动过程不画点。

void far moverel(int dx,int dy);该函数从现行位置（x，y）移到一位置增量处（x+dx，y+dx），移动过程不画点。

int far getx(void);得到当前画笔的 x 位置。

int far gety(void);得到当前画笔的 y 位置。

3. 画线函数

这类函数提供了用设定的颜色从一个点到另一个点画一条直线的功能，由于起始点的设定方法不同，因而有下面不同的画线函数：

void far line(int x0,int y0,int x1,int y1);该函数从（x_0，y_0）点到（x_1，y_1）点画一直线。

void far lineto(int x,int y);该函数从现行画笔位置到（x，y）点画一直线。

void far linerel(int dx,int dy);该函数从现行画笔位置（x，y）到位置增量处（x+dx，y+dy）画一直线。

例 4-2　用 line 函数画直线时，将不考虑画笔位置，它也不影响画笔原来的位置，lineto 和 1inerel 要求画笔位置，画线起点从此位置开始，而结束位置就是画笔画线完后停留的位置，故这两个函数将改变画笔的位置。

```
#include <graphics.h>
main()
{   int driver=VGA;
    int mode=VGAHI;
    initgraph(&driver,&mode,"   ");
    cleardevice();
```

```
    moveto(100,20);
    lineto(100,80);
    moveto(200,20);
    lineto(100,80);
    line(100,90,200,90);
    linerel(0,20);
    moverel(-100,0);
    linerel(30,20);
    getch();
    closegraph();
}
```

4. 画矩形和条形图函数

void far rectangle(int xl,int y1,int x2,int y2);该函数将以（x_1，y_1）为左上角，（x_2，y_2）为右下角画一矩形框。

void bar(int x1,int y1,int x2,int y2);该函数将以（x_l，y_1）为左上角，（x_2，y_2）为右下角画一实形条状图，没有边框，图的颜色和填充模式可以设定。若没有设定，则使用缺省模式。

void far drawpoly(int numpoints, int far *polypoints);该函数画一个顶点数为 numpoints，各顶点坐标由 polypoints 给出的多边形。polypoints 整型数组必须至少有 2 倍顶点数个元素。每一个顶点的坐标都定义为（x，y），并且 x 在之前。值得注意的是，当画一个封闭的多边形时，numpoints 的值取实际多边形的顶点数加一，并且数组 polypoints 中第一个点和最后一个点的坐标相同。

例 4-3　用 drawpoly()函数画箭头。

```
# include<graphics. h>
# include<conio.h>
main()
{   int gdriver, gmode, i;
    int arw[16]={200, 102, 300, 102, 300, 107, 330,100, 300, 93, 300, 98, 200, 98, 200, 102};
    gdriver=DETECT;
    initgraph(&gdriver, &gmode, "");
    setbkcolor(BLUE);
    cleardevice();
    setcolor(12);                    /*设置作图颜色*/
    drawpoly(8, arw);                /*画一箭头*/
    getch();
    closegraph();
}
```

例 4-4　下面的程序将由 rectangle 函数以（100，20）为左上角，（200，50）为右下角画一矩形，接着又由 bar 函数以（100，80）为左上角，（150，180）为右下角画一实形条状图，用缺省颜色（白色）填充。

```
# include <graphics.h>
main()
{   int driver=DETECT;
    int mode, x;
```

```
    initgraph(&driver,&mode,"  ");
    cleardevice();
    rectangle(100,20,200,50);
    bar(100,80,150,180);
    getch();
    closegraph();
}
```

5．画椭圆、圆和扇形图函数

有关于角的概念，在 Turbo C 2.0 中是这样规定的：屏的 x 轴方向为 0°，当半径从此处逆时针方向旋转时，则依次是 90°、180°、270°，到 360°时，则和 x 轴正向重合，即旋转了一周，如图 4-2 所示。

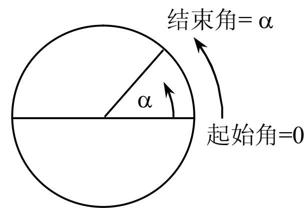

图 4-2　角的概念

void ellipse(int x,int y,int stangle,int endangel,int xradius,int yradius);该函数将以（x，y）为中心，以 xradius 和 yradius 为 x 轴和 y 轴半径，从起始角 stangle 开始到 endangle 角结束，画一椭圆线。当 stangle=0，endangle=360 时，则画出的是一个完整的椭圆，否则画出的将是椭圆弧。关于起始角和终止角规定如图 4-2 所示。

void far circle(int x,int y,int radius);该函数将以（x，y）为圆心，radius 为半径画个圆。

void far arc(int x,int y,int stangle,int endangle,int radius);该函数将以（x，y）为圆心，radius 为半径，从 stangle 为起始角开始，到 endangle 为结束角画一圆弧。

void far pieslice(int x,int y,int stangle,int endangle,int radius);该函数将以（x，y）为圆心，radius 为半径，从 stangle 为起始角，endangle 为结束角，画一扇形图，扇形图的填充模式和填充颜色可以事先设定，否则以缺省模式进行。

例 4-5　该程序将用 ellipse 函数画椭圆，从中心为（320，100），起始角为 0°，终止角为 360°，x 轴半径为 75，y 轴半径为 50 画一椭圆，接着用 circle 函数以（320，220）为圆心，以半径为 50 画圆。然后分别用 pieslice 和 ellipse 及 arc 函数在下方画出了一扇形图和椭圆弧及圆弧。

```
#include <graphics.h>
main()
{   int driver=DETECT;
    int mode,x;
    initgraph(&driver,&mode,"  ");
    cleardevice();
    ellipse(320,100,0,360,75,50);
    circle(320,220,50);
    pieslice(320,340,30,150,50);
    ellipse(320,400,0,180,100,35);
```

```
    arc(320,400,180,360,50);
    getch();
    closegraph();
}
```

4.2.2 颜色设置函数

在 Turbo C 2.0 中，像素的显示颜色，或者说画点、画线、填充面的颜色既可采用默认值，也可用一些函数来设置。图形方式下，像素有前景色和背景色之分，一般用以下的两个函数来设置前景色和背景色：

void far setbkcolor(int color);设置背景色。

void far setcolor(int color);设置前景（作图）色。

其中 color 为图形模式下颜色的规定数值，对 EGA、VGA 显示器的适配器来说，有关颜色的符号常数及数值如表 4-3 所示。

表 4-3 景色值与对应的颜色名

色值	颜色名	颜色	色值	颜色名	颜色
0	BLACK	黑	8	DARKGRAY	深灰
1	BLUE	蓝	9	LIGHTBLUE	淡蓝
2	GREEN	绿	10	LIGHTGREEN	淡绿
3	CYAN	青	11	LIGHTCYAN	淡青
4	RED	红	12	LIGHTRED	淡红
5	MAGENTA	洋红	13	LIGHTMAGENTA	淡洋红
6	BROWN	棕	14	YELLOW	黄
7	LIGHTGRAY	浅灰	15	WHITE	白

例 4-6 Turbo C 2.0 还提供了几个获得现行像素颜色设置情况的函数。

int far getbkcolor(void);返回现行背景颜色值。

int far getcolor(void);返回现行作图颜色值。

int far getmaxcolor(void);返回最高可用的颜色值。

```
# include<graphics. h>
# include<conio.h>
main()
{   int driver, mode, i;
    driver=DETECT;
    registerbgidriver(EGAVGA_DRIVER);     /*建立独立图形运行程序*/
    initgraph(&driver, &mode", "");        /*图形初始化*/
    setbkcolor(0);                         /*设置图形背景*/
    cleardevice();
    for(i=0; i<=15; i++)
    {   setcolor(i);                       /*设置不同作图色*/
        circle(320, 240, 20+i*10);         /*画半径不同的圆*/
    delay(1000);                           /*延迟 1000 毫秒*/
```

```
    }
    for(i=0; i<=15; i++)
    {   setbkcolor(i);                  /*设置不同背景色*/
        cleardevice();
        circle(320, 240, 20+i*10);
        delay(1000);
    }
    closegraph();
}
```

4.2.3　颜色控制函数

表 4-3 显示的只是一般情况下屏幕背景颜色的符号表，其实按照 CGA、EGA、VGA 图形适配器的硬件结构，颜色是可以通过对其内部相应的寄存器进行编程来改变的。

为了形象地说明颜色的设置，一般用所谓调色板来进行描述，它实际上对应着一些硬件的寄存器。从 C 语言的角度来看，调色板就是一张颜色索引表。

对 CGA 显示器，在中分辨显示方式下，有 4 种显示模式，每一种模式对应有一个调色板，可用调色板号区别。每个调色板有 4 种颜色可以选择，颜色可以用颜色值 0、1、2、3 来进行选择，由于 CGA 有 4 个调色板，一旦显示模式确定后，调色板即确定，如选 CGAC0 模式，则选 0 号调色板，但选调色板的哪种颜色则可由用户根据需要从 0、1、2 和 3 中进行选择，表 4-4 就列出了调色板与其对应的颜色值。其中若选调色板的颜色值为 0，表示此时选择的颜色和当时的背景色一样。

表 4-4　CGA 的调色板号与对应的颜色值

模式	调色板号	颜色值			
		0	1	2	3
CGAC0	0	背景色	绿	红	黄
CGAC1	1	背景色	青	洋红	白
CGAC2	2	背景色	淡绿	淡红	棕
CGAC3	3	背景色	淡青	淡洋红	淡灰

对 EGA 显示器，其调色板也是一个颜色索引表，它存有 16 种颜色，经过组合变换最终可以产生 64 种颜色，因此就可以通过对其内部相应的寄存器进行编程来改变这 16 种颜色。EGA 图形系统初始化时，16 个调色板寄存器已装入确定的颜色，它们也被称为标准色。

VGA 显示器，也只有一个调色板，对应 16 个调色板寄存器。但这些寄存器装的内容和 EGA 的不同，它们装的又是一个颜色寄存器表的索引。经过组合变换最终 256 个颜色寄存器供索引。由于 Turbo C 2.0 中没有支持 VGA 的 256 色的图形模式，因此 EGA 和 VGA 的调色板寄存器装的值虽然一样（当图形系统初始化时，指默认值），但含义不同，前者装的是颜色值，后者装的是颜色寄存器索引号，不过它们最终表示的颜色是一致的。

voidfar setpalette(int index, int actual_color);该函数用来对调色板进行颜色设置，一般用在 EGA、VGA 显示方式上。

void far setallpalette(struct palettetype far *palette);改变调色板 16 种颜色的函数。

void far getpalette(struct palettetype far *palette); 将得到调色板的颜色数（即调色板寄存器个数）和装的颜色值。

void far getpalettesize(void); 将得出调色板颜色数，同时 getpalette 函数将把得到信息存入由 palette 指向的结构中。

结构 palettetype 定义如下：

```
#define MAXCOLORS l5
struct palettetype
{   unsigned char size;
    signed char colors[MAXCOLORS+1];
};
```

该定义在头文件 graphics.h 中。size 元素由适配器类型和当前模式下调色板的颜色数决定，即调色板寄存器数。colors 是个数组，它实际上代表调色板寄存器，每个数组元素的值就表示相应调色板寄存器的颜色值。对 VGA 的 VGAHI 模式，size=16，默认的 colors 的各元素值就相当于表 4-5 所列值。

表 4-5　6 个调色板寄存器对应的标准色和值

寄存器号	颜色名	值	寄存器号	颜色名	值
0	EGA_BLACK	0	8	EGA_DARKGRAY	8
1	EGA_BLUE	1	9	EGA_LIGHTBLUE	9
2	EGA_GREEN	2	10	EGA_LIGHTGREEN	10
3	EGA_CYAN	3	11	EGA_LIGHTCYAN	11
4	EGA_RED	4	12	EGA_LIGHTRED	12
5	EGA_MAGENTA	5	13	EGA_LIGHTMAGENTA	13
6	EGA_BROWN	6	14	EGA_YELLOW	14
7	EGA_LIGHTGRAY	7	15	EGA_WHITE	15

例 4-7　下面的一个程序演示了调色板颜色设置函数的作用，首先程序用 setcolor(1)设置了前景颜色，其中参数 1 表示用 1 号调色板寄存器中的颜色，即默认值为蓝色，这样用 rectangle 将画出一个蓝色的方框，然后按任一键，这时用 setpalette(1，i)函数将分别设置 1 号调色板寄存器为绿、青、红、黄、亮白。每按一键，方框颜色改变一次，直到方框变成亮白为止。因为调色板相应的调色板寄存器所装的颜色一旦改变，用 setcolor（调色板寄存器号）设置的颜色也立即改变。

该程序中将 palette 定义成 palettetype 类型结构，palettetype 结构如前所述，在 graphics.h 头文件中已有定义，这样用 getpalette(&pallette)函数得到图形系统初始化的各调色板寄存器的颜色值（即默认值），然后在程序快结束时，用 setallpalette(&palette)恢复各调色板寄存器的原来值。

```
#include <graphics.h>
main()
{   int driver=DETECT,mode;
    struct palettetype palette;
    int i,j;
    initgraph(&driver,&mode,"");
```

```
    getpalette(&palette);
    setcolor(1);
    rectangle(200,200,300,320);
    getch();
    i=2;
    j=2;
    do
    {   printf("color=%d", j);
        setpalette(1,i);
        getch();
        i++;
        j++;
        if(i= =6)     i=20 ;
        if(i= =21)    i=7 ;
        if(i= =8)     i=56;
    } while (j<16);
    getch();
    setallpalette(&palette);
    closegraph();
}
```

4.2.4　画线的线型函数

1．设定线型函数

void far setlinestyle(int linestyle,unsigned upattern,int thickness);

当线的宽度参数（thickness）不设定时，取默认值，即一个像素宽，当设定为 3 时，可取 3 像素宽，取值如表 4-6 所示。当线型参数（linestyle）不设定时，取默认值，即实线；设定时，可有 5 种选择，如表 4-7 所示。upattern 参数只有在 1inestyle 取 4 或 USERBIT_LINE 时才有意义，即表示在用户自定义线型时，该参数才有用。该参数若表示成 16 位二进制数，则每位代表一个像素。是 1 的位，代表的像素用前景色显示；是 0 的位，代表的像素用背景色显示（实际没有显示）。

表 4-6　线宽（thickness）

符号名	值	含义
NORM_WIDTH	1	1 个像素宽
THICK_WIDTH	3	3 个像素宽

表 4-7　直线的形状（linestyle）

符号名	值	含义
SOLID_LINE	0	实线
DOTTED_LINE	1	点线
CENTER_LINE	2	中心线
DASHED_LINE	3	点画线
USERBIT_LINE	4	用户自定义线

例 4-8 画线。

```
#include <graphics.h>
main()
{   int driver=VGA, mode =VGAHI;
    int i,j,x1,y1,x2,y2;
    initgraph(&driver,&mode," "),
    setbkcolor(EGA_BLUE);
    cleardevice();
    setcolor(EGA_GREEN);
    circle(320,240,98);              /*画出一个绿色圆 */
    j=0;
    setcolor(12);                    /*设置颜色为淡红色 */
    for(i=0;i<=90;i=i+6)
    {
        setlinestyle(0,0,j);         /*画出一个套一个的矩形框 */
        x1=440-i; y1=280-i;
        x2=440+i; y2=280+i;
        rectangle(x1,y1,x2,y2);
        j=j+3;
        if(j>4) j=0;
    }
    j=0;
    for ( i=0;i<=180;i=i+16)         /*画出屏幕中心的 4 种线型的 4 色线 */
    {   if(j>3)j=0;
        setcolor(j+2);
        setlinestyle(j,0,3);
        j++;
        x1=0; y1=i,
        x2=640; y2=480-i;
        line(x1,y1,x2,y2);
    }
    setcolor(EGA_WHITE);
    setlinestyle(4,0x1001,1);        /*用户定义线型，1 个像素宽 */
    line(220,240,420,240);           /*画出通过圆心的 y 线 */
    line(320,140,320,340);           /*画出通过圆心的 x 线 */
    getch();
    closegraph();
}
```

2．得到当前画线信息的函数

void far getlinesettings(struct linesettingstype far *lineinfo);

该函数将把当前有关线的信息存放到由 lineinfo 指向的结构中，linesettingstype 结构定义如下：

```
structlinesettingstype
{   int linestyle;
    unsigned upattern;
    int thickness;
};
```

4.2.5 封闭图形的填色函数及有关画图函数

1. 填色函数

void far setfillstyle(int pattern,int color);

该函数将用设定的 color 颜色和 pattern 图模式对后面画出的轮廓图进行填充，这些图轮廓是由待定函数画出的，color 实际上就是调色板寄存器索引号，对 VGAHI 方式为 0~15，即 16 色，pattern 表示填充模式，可用表 4-8 中的值或符号名表示。

表 4-8 填充模式（pattern）的规定

符号名	值	含义
EMPTY_FILL	0	用背景色填充
SOLID_FILL	1	用单色实填充
LINE_FILL	2	用“—”线填充
LTSLASH_FILL	3	用“//”线填充
SLASH_FILL	4	用粗“//”线填充
BKSLASH_FILL	5	用“\\”线填充
LTBKSLASH_FILL	6	用粗“\\”线填充
HATCH_FILL	7	用方网格线填充
XHATCH_FILL	8	用斜网格线填充
INTTERLEAVE_FILL	9	用间隔点填充
WIDE_DOT_FILL	10	用稀疏点填充
CLOSE_DOT_FILL	11	用密集点填充
USER_FILL	12	用用户定义样式填充

当 pattern 选用 USER_FILL 用户自定义样式填充时，setfillstyle 函数对填充的模式和颜色不起任何作用，若要选用 USER_FILL 样式填充时，可选用下面的函数。

2. 用户自定义填充函数

void far setfillpattern(char *upattern,int color);

该函数设置用户自定义可填充模式，以 color 指出的颜色对封闭图形进行填充。这里的 color 实际上就是调色板寄存器号，也可用颜色名代替。参数 upattern 是一个指向 8 个字节存储区的指针，这 8 个字节表示了一个 8×8 像素点阵组成的填充图模，它是由用户自定义的，它将用来对封闭图形填充。8 个字节的图模是这样形成的：每个字节代表一行，而每个字节的每一个二进制位代表该行的对应列上的像素。是 1 则用 color 显示，是 0 则不显示。

例 4-9 下面的程序演示了如何用 setfillstyle(SOLID_FILL，color)对用 bar 生成的条状图进行填充。对 VGA 显示器，由于可显示 16 色，因而通过 getpalette(&palette)函数，可得出 palette，size 为 16。这样 for 循环，将使得用 bar 函数生成的 16 个小条分别填充上调色板上的 16 种颜色，其顺序为缺省时（即标准的）调色板各寄存器的顺序颜色。

do 循环又利用随机函数 random(palette.size)随机产生 0~palette.size 的整数，对调色板各寄存器的颜色重新进行设置，这样一旦 setpalette 函数对某调色板寄存器进行了颜色的重新设置，则代表相应号调色板寄存器的小条颜色立即变成新设的颜色。若随机遇到 random 产生一

个 0，则 setpalette 立即对 0 号调色板用第二次 random 产生的数进行颜色设置，因而此时整个屏幕显示的背景色立即发生变化，颜色为第二次 random 产生的颜色。关于调色板的设置问题，可参阅前面已介绍过的调色板设置函数。

```
#include <graphics.h>
#include <stdio.h>
#include <stdlib.h>
main()
{   int driver=DETECT,mode;
    struct palettetype palette;
    int color;
    initgraph(&driver,&mode,"");
    getpalette(&palette);
    for (color=0; color<palette.size; color++ )
     /*对 16 个小条用 16 种颜色填充*/
    {   setfillstyle(SOLID_FILL,color);
        bar(20*(color-1),0,20*color,20);
    }
    if(palette.size>1)
    {     do     /* 对调色板 16 种颜色重新进行设置*/
        setpalette(random(palette.size), random(palette.size));
        while (!kbhit() );
        getch();
    }
    setallpalette(&palette);
    closegraph();
}
```

例 4-10 下面的程序演示了用不同填充图模（pattern）对由 bar 和 pieslice 函数产生的条状和扇形图进行颜色填充。运行程序，可以看出第 1 个 bar(0,0,100,100)产生的方条将由蓝色的斜线填充，即以 LTSlASH_FILL(3)图模填充。接着将由红色的网格（HATCH_FILL，RED）图模填充一个扇形。由于缺省时，前景颜色为白色，故该扇形将用白色边框画出，接着用户自定义填充模式，因而用 bar(100,100,200,200)画出的方条，将用用户定义的图模（用字符数组 gray50[]表示的图模），用黄色进行填充。

```
#include <graphics.h>
main()
{   int driver=VGA,mode=VGAHI;
    struct fillsettingstype save;
    char savepattern[8];
    char gray50[]={0xff,0x00,0x00,0x00,0x00,0x00,0x00,0x81};
    initgraph(&driver,&mode,"");
    getfillsettings(&save);              /* 得到初始化时填充模式*/
    if(save.pattern != USER_FILL )   setfillstyle(3,BLUE);
    bar(0,0,100,100);
    setfillstyle(HATCH_FILL,RED);
    pieslice(200,300,90,180,90);
    setfillpattern(gray50,YELLOW);   /* 设定用户自定义图模进行填充*/
    bar(100,100,200,200);
```

```
    if(save.pattern==USER_FILL)
    setfillpattern(savepattern,save.color);
    else
    setfillpattern(savepattern, save.color); /* 恢复原来的填充模式 */
    getch();
    closegraph();
}
```

3. 得到填充模式和颜色的函数

void far fillsettings(struct fillsettingstype far *fillinfo);

它将得到当前的填充模式和颜色，这些信息存在结构指针变量 fillinfo 指出的结构中该结构定义是：

```
struct fillsettingstype
{   int pattern;        /*当前填充模式*/
    int color;          /*填充颜色*/
};
```

void far getfillpattern(char *upattern);

该函数将把用户自定义的填充模式和颜色存入由 upattern 指向的内存区域中。

4. 与填充函数有关的作图函数

void far bar3d(int x1,int y1,int x2,int y2,int depth,int topflag);

该函数画三维立体直方图，当 topflag 非 0 时，画出三维顶，否则将不画出三维顶，depth 决定了三维直方图的长度。

viod far sector(int x,int y,int stangle,int endang1e,int xradius,int yradius);

该函数将以（x，y）为圆心，以 xradius 和 yradius 为 x 轴和 y 轴半径，从起始角 stangle 开始到 endangle 角结束，画一椭圆扇形图，并按设置的填充模式和颜色填充。当 stangle 为 0，endangle 为 360 时，则画出一完整的椭圆图。

void far fillellipse(int x,int y,int xradius,int yradius);

该函数将以（x，y）为圆心，以 xradius 和 yradius 为 x 轴和 y 轴半径，画一椭圆图，并以设定或默认模式和颜色填充。

void far fillpoly(int numpoints,int far *polypoints);

该函数将画出一个顶点数为 numpoints，各顶点坐标由 polypoints 给出的多边形，也即边数为 polypoints-1，当为一封闭图形时，numpoints 应为多边形的顶点数加 1，并且第一个顶点坐标应和最后一个顶点的坐标相同。

例 4-11　下面程序用 bar3d 函数画出了一个立方图，并且画面用蓝色斜线填充，接着由第二个 bar3d 函数又在相邻位置画出一个没有顶的三维图，画面用红色方格填充。该函数的 topflag=0。在屏幕下方，由 sector 函数画出了一个不完整的椭圆，并用绿色填充，可以看出差 120°就是一个完整的椭圆了。在其相邻位置则是由 fillellipse 函数画出的一个椭圆，它用淡红色填充，屏幕的右上半是由 fillpoly 函数画出的一个六边图形，被填以洋红色，由于最初顶点坐标和最后一个顶点坐标相同（同为（420，20）），所以是一个封闭的图形。

```
#include <graphics.h>
main()
{   int driver=VGA,mode=VGAHI;
    struct fillsettingstype save;
```

```
    char savepattern[8];
    int d[]={420,20,330,45,330,145,420,120,510,145,510,55,420,20};
    initgraph(&driver,&mode,"");
    getfillsettings(&save);
    setfillstyle(3,BLUE);
    bar3d(100,50,150,120,30,1);
    setfillstyle(HATCH_FILL,RED);
    bar3d(200,50,250,120,30,0);
    setfillstyle(1,GREEN);
    sector(200,300,0,250,100,40);
    setfillstyle(1,LIGHTRED);
    fillellipse(420,300,100,40);
    setfillstyle(1,5);
    fillpoly(7,d);
    getch();
    setfillstyle(save.pattern,save.color);
    closegraph();
}
```

5. 可对任意封闭图形填充的函数

前面介绍的填充函数，只能对由上述特定函数产生的图形进行颜色填充，对任意封闭图形均可进行填充的还有一函数，其原型说明为：

```
void far floodfill(int x,int y,int border);
```

该函数将对一封闭图形进行填充，其颜色和模式将由设定的或默认的图模与颜色决定。其中参数（x，y）为封闭图形中的任一点，border 是封闭图形的边框颜色。编程时该函数位于画图形的函数之后，即要填充该图形。需要注意的是：

（1）若（x，y）点位于封闭图形边界上，该函数将不进行填充。

（2）若对不是封闭的图形进行填充，则会填到别的地方，即会溢出。

（3）若（x，y）点在封闭图形之外，将对封闭图形外进行填充。

（4）由参数 border 指出的颜色必须与封闭图形的轮廓线的颜色一致；否则会填到别的地方去。

例 4-12 下面的程序首先用白色线画出一个长方体，并用设定的亮红色（LIGHTRED）和实填充模式填充该长方体的正面，然后使用两个 floodfill 函数用同样的模式和颜色填充该长方体能看得见的另两面，然后将画线颜色由 setcolor(LIGHTGREEN)设置为亮绿色，并由 setfillstyle 设置填充模式为棕色和用平直线填充，由于该函数对 rectangle 画出的矩形框不起作用，所以并不执行填充，当画好矩形框后，其后的 floodfill 函数将完成对该矩形框的填充，即以棕色的平直线进行填充。可以作实验，当将 floodfill 中的 x、y 参数设在被填图形框外时，结果会将该框外的所有区域填充。当 floodfill 中的 border 指定的颜色和画图框的颜色不符时，也会将颜色填到图外边去。

```
#include <graphics.h>
main()
{   int driver=VGA,mode=VGAHI;
    initgraph(&driver,&mode,"");
    setbkcolor(BLUE);
    setcolor(WHITE);                    /* 用白色画线 */
```

```
    setfillstyle(1,LIGHTRED);              /* 设填充模式和颜色 */
    bar3d(100,200,400,350,100,1);          /* 画长方体并填正面*/
    floodfill(450,300,WHITE);              /* 填侧面 */
    floodfill(250,150,WHITE);              /* 填顶部 */
    setcolor(LIGHTGREEN);
    setfillstyle(2,BROWN);
    rectangle(450,400,500,450);            /* 画矩形 */
    floodfill(470,420,LIGHTGREEN);         /* 填矩形 */
    getch();
    closegraph();
}
```

4.2.6　图视窗口操作函数

在图形方式下可以在屏幕上某一区域设置一个窗口，这样以后的画图操作均在这个窗口内进行，且使用的坐标以此窗口顶左上角为（0，0）作参考，而不再用物理屏幕坐标［屏左角为（0，0）点］。在图视窗口内画的图形将显示出来，超出图视窗口的部分可以不让其显示出来，也可以让其显示出来（不剪断）。

1．图视窗口设置函数

void far setviewport(int xl,int y1,int x2,int y2, clipflag);

其中，（x1，y1）为图视窗口的左上角坐标，（x2，y2）为所设置的图视窗口右下角坐标，它们都是以原屏幕物理坐标为参考的。clipflag 参数若为非 0，则所画图形超出图视窗口的部分将被切除而不显示出来。若 clipflag 为 0，则超出图视窗口的图形部分仍将显示出来。

2．图视窗口清除与取信息函数

void far clearviewport(void);

该函数将清除图视窗口内的图像。

void far getviewsettings(struct viewport type far *viewport);

该函数将取得当前设置的图视窗口的信息，它存于由结构 viewporttype 定义的结构变量 viewport 中，结构 viewporttype 定义如下：

struct viewporttype {int left,top,right,bottom;int clipflag;};

使用图视窗口设置函数 setviewport，可以在屏上设置不同的图视窗口，甚至部分可以重叠，然而最近一次设置的窗口才是当前窗口，后面的图形操作都视为在此窗口中进行，其他窗口均无效。若不清除那些窗口的内容，则它们仍在屏上保持，当要对它们处理时，可再一次设置那个窗口一次，这样它就又变成当前窗口了。使用 setbkcolor 设置背景色时，对整个屏幕背景起作用，它不能只改变图视窗口内的背景，在用 setcolor 设置前景色时，它对图视窗口内画图起作用。若下一次设置那个图视窗口没有设置颜色，那么上次在另一图视窗口内设置的颜色在本次设置的图视口内仍起作用。

例 4-13　下面程序首先用 setviewpon 函数设置了一个左上角为（0，0）、右下角为（639，199）的图视窗口，并设置 clipflag 参数为 1，即超出图视窗口的图形将被切除，接着画了一个方框和一个边与方框一边相切的圆，它将完整地显示出来。按任意键后，开一个图视窗口，用棕色画了和第一个窗口相同的方框和圆，超出图视窗口的部分被剪切。由于在开此窗口前，没有用 clearviewport()清除上次的窗口内容，所以上次画的洋红色的方框和圆仍保留，当再按任一键后，调用了 clearviewport()函数，因而将窗口中的棕色方框和圆清除了。接着又建立了一

个图视窗口，该窗口（50,50,200,125）和最初窗口（0,0,639,199）有部分重合，因而重合部分的内容将在当前窗口中显示出来。新画的方框和圆也显示出来（窗外部分内容仍保留），由于选择了 clipflag=1，所以超出窗口的方框和圆的部分被剪切，不显示出来，但再按任一键后，由于用了 clearviewport，所以窗口中内容被清除，并又在同一位置开辟了同样大小的窗口，但由于 clipflag=0，所以超出的部分没被剪掉，因而可看见。

```
#include <graphics.h>
main()
{   int i,driver,mode,size,page;
    driver=VGA;
    mode=VGAHI;
    initgraph(&driver,&mode," ");
    cleardevice();
    setviewport(0,0,639,199,1);            /* 设图视窗口 */
    setcolor(5);
    rectangle(50,50,125,100);
    circle(100,75,50);
    getch();
    setviewport(150,150,639,239,1);       /* 又设一图视窗口 */
    setcolor(6);
    rectangle(50,50,125,100);
    circle(100,75,50);
    getch();
    clearviewport();     /* 方框超出部分被切掉 */
    setviewport(50,50,200,125,1);
    rectangle(50,50,125,100);
    circle(100,75,50);
    getch();
    clearviewport();
    setviewport(50,50,200,125,0);
    rectangle(50,50,125,100);
    circle(100,75,50);
    getch();
    clearviewport();     /* 开窗口，超出部分不切除 */
    closegraph();
}
```

4.2.7 图形方式下的文本输出函数

在图形模式下，只能用标准输出函数，如 printf()、puts()、putchar() 函数输出文本到屏幕。除此之外，其他输出函数（如窗口输出函数）都不能使用，即使是可以输出的标准函数，也只以前景色为白色，按 80 列、25 行的文本方式输出。

为此 C 语言另外提供了一些专门用于在图形显示模式下的文本输出函数。下面将分别进行介绍。

1．文本输出函数

void far outtext(char far *textstring);

该函数在现行位置输出字符串指针 textstring 所指的文本。

void far outtextxy(int x, int y, char far *textstring);

该函数输出字符串指针 textstring 所指的文本在规定的（x，y）位置。其中 x 和 y 为像素坐标。

说明：这两个函数都是输出字符串，但经常会遇到输出数值或其他类型的数据,此时就必须使用格式化输出函数 sprintf()。

sprintf()函数的调用格式为：int sprintf(char *str, char *format, variable-list);它与 printf()函数不同之处是将按格式化规定的内容写入 str 指向的字符串中，返回值等于写入的字符个数。

例如：sprintf(s, "your TOEFL score is %d", mark);

这里 s 应是字符串指针或数组，mark 为整型变量。

2. 有关文本字体、字型和输出方式的设置

有关图形方式下的文本输出函数，可以通过 setcolor()函数设置输出文本的颜色。另外，也可以改变文本字体大小以及选择是水平方向输出还是垂直方向输出。

void far settexjustify (int horiz, int vert);

该函数用于定位输出字符串。对使用 outtextxy (int x, int y, char far *str textstring) 函数所输出的字符串；其中哪个点对应于定位坐标（x，y）在 Turbo C 2.0 中是有规定的。如果把一个字符串看成一个长方形的图形，在水平方向显示时，字符串长方形按垂直方向可分为顶部，中部和底部 3 个位置，水平方向可分为左、中、右 3 个位置，两者结合就有 9 个位置。

settextjustify()函数的第一个参数 horiz 指出水平方向 3 个位置中的一个，第二个参数 vert 指出垂直方向 3 个位置中的一个，二者就确定了其中一个位置。当规定了这个位置后，用 outtextxy()函数输出字符串时，字符串长方形的这个规定位置就对准函数中的（x，y）位置。而对用 outtext()函数输出字符串时，这个规定的位置就位于现行游标的位置。有关参数 horiz 和 vert 的取值参见表 4-9。

表 4-9　数 horiz 和 vert 的取值

符号常数	数值	用于
LEFT_TEXT	0	水平
RIGHT_TEXT	2	水平
BOTTOM_TEXT	0	垂直
TOP_TEXT	2	垂直
CENTER_TEXT	1	水平或垂直

void far settextstyle(int font, int direction, int charsize);

该函数用来设置输出字符的字形（由 font 确定）、输出方向（由 direction 确定）和字符大小（由 charsize 确定）等特性。C 语言对该函数中各个参数的规定见表 4-10 至表 4-12 所示。

表 4-10　font 的取值

符号常数	数值	含义
DEFAULT_FONT	0	8×8 点阵字（默认值）
TRIPLEX_FONT	1	3 倍笔画字体
SMALL_FONT	2	小号笔画字体
SANSSERIF_FONT	3	无衬线笔画字体
GOTHIC_FONT	4	黑体笔画字

表 4-11 irection 的取值

符号常数	数值	含义
HORIZ_DIR	0	从左到右
VERT_DIR	1	从底到顶

表 4-12 charsize 的取值

符号常数或数值	含义
1	8×8 点阵
2	16×16 点阵
3	24×24 点阵
4	32×32 点阵
5	40×40 点阵
6	48×48 点阵
7	56×56 点阵
8	64×64 点阵
9	72×72 点阵
10	80×80 点阵
USER_CHAR_SIZE=0	用户定义的字符大小

例 4-14 图形屏幕下文本输出和字体字型设置函数的用法。

```
# include<graphics.h>
main()
{   int i, driver, mode;
    char s[30];
    driver=DETECT;
    initgraph(&driver, &mode, "");
    setbkcolor(BLUE);
    cleardevice();
    setviewport(100, 100, 540, 380, 1);   /*定义一个图形窗口*/
    setfillstyle(1, 2);                   /*绿色以实填充*/
    setcolor(YELLOW);
    rectangle(0, 0, 439, 279);
    floodfill(50, 50, 14);
    setcolor(12);
    settextstyle(1, 0, 8);    /*三重笔画字体，水平放大 8 倍*/
    outtextxy(20, 20, "Good Better");
    setcolor(15);
    settextstyle(3, 0, 5);   /*无衬笔画字体，水平放大 5 倍*/
    outtextxy(120, 120, "Good Better");
    setcolor(14);
    settextstyle(2, 0, 8);
    i=620;
```

```
    sprintf(s, "Your score is %d", i);      /*将数字转化为字符串*/
    outtextxy(30, 200, s);                  /*指定位置输出字符串*/
    setcolor(1);
    settextstyle(4, 0, 3);
    outtextxy(70, 240, s);
    getch();
    closegraph();
}
```

3．用户对文本字符大小的设置

前面介绍的 settextstyle()函数，可以设定图形方式下输出文本字符的字体和大小，但对于笔画型字体（除 8×8 点阵字以外的字体）只能在水平和垂直方向以相同的放大倍数放大。为此 C 又提供了另一个 setusercharsize() 函数，对笔画字体可以分别设置水平和垂直方向的放大倍数。该函数的调用格式为：

void far setusercharsize(int mulx, int divx, int muly, int divy);

该函数用来设置笔画型字和放大系数，它只有在 settextstyle() 函数中的 charsize 为 0(或 USER_CHAR_SIZE)时才起作用，并且字体为函数 settextstyle()所规定的字体。调用函数 setusercharsize()后，每个显示在屏幕上的字符都以其缺省大小乘以 mulx/divx 为输出字符宽，乘以 muly/divy 为输出字符高。

例 4-15　setusercharsize()函数的用法。

```
# include<graphics.h>
main()
{   int dirver, mode;
    driver=DETETC;
    initgraph(&driver, &mode, "");
    setbkcolor(BLUE);
    cleardevice();
    setfillstyle(1, 2);                     /*设置填充方式*/
    setcolor(WHITE);                        /*设置白色作图*/
    rectangle(100, 100, 330, 380);
    floodfill(50, 50, 14);                  /*填充方框以外的区域*/
    setcolor(12);                           /*作图色为淡红*/
    settextstyle(1, 0, 8);                  /*三重笔画字体，放大 8 倍*/
    outtextxy(120, 120, "Very Good");
    setusercharsize(2, 1, 4, 1);            /*水平放大 2 倍，垂直放大 4 倍*/
    setcolor(15);
    settextstyle(3, 0, 5);                  /*无衬字笔画，放大 5 倍*/
    outtextxy(220, 220, "Very Good");
    setusercharsize(4, 1, 1, 1);
    settextstyle(3, 0, 0);
    outtextxy(180, 320, "Good");
    getch();
    closegraph() ;
}
```

4.3 动画技术

大家知道电影或动画片是由一张张图像组成的，它利用人眼不能够分辨出时间间隔在 25 毫秒内的动态图像变化这一特性，在这些连续图像被放映时，从视觉效果上给人以动的感觉。所以在计算机屏幕上产生运动的效果需要动画技术。

4.3.1 采用延迟与清屏交错的实现方法

这种方法利用 cleardevice()和 delay()函数相互配合，先画一幅图形，让它延迟一段时间，然后清屏，再画另一幅，如此反复，就形成动态效果。

例 4-16 通过函数 graphone()、graphtwo()和 graphthree()实现了 3 个简单的动画画面，这 3 个画面不停地进行切换。

```
#include<graphics.h>
#include<stdlib.h>
#include<dos.h>
int x,y,maxcolor;
void graphone(char *str);  /*使字符串 str 左右运动，线条上下运动*/
void graphtwo(char *str);  /*使字符串 str 上下运动，线条左右运动*/
void graphthree(char *str); /*使字符串 str 由小变大，再由大变小，直线也随之变化*/
main()
{  int i,driver,mode;
   char *str="W E L C O M E !";
   driver=DETECT;
   mode=0;
   initgraph(&driver,&mode,"");        /*系统初始化*/
   cleardevice();                      /*清屏*/
   settextjustify(CENTER_TEXT,CENTER_TEXT);
   x=getmaxx();                        /* 返回当前图形模式下的最大有效的 x 值*/
   y=getmaxy();                        /* 返回当前图形模式下的最大有效的 y 值*/
   maxcolor=getmaxcolor();             /* 返回当前图形模式下最大有效的颜色值*/
   while(!kbhit())
   {  graphone(str);                   /* 第一个动画*/
      graphtwo(str);                   /* 第二个动画*/
      graphthree(str);                 /* 第三个动画*/
   }
   getch();
   closegraph();                       /* 关闭图形模式*/
}
void  graphone(char *str)
{  int i;
   for(i=0;i<40;i++)
   {  setcolor(1);
      settextstyle(1,0,4);
      setlinestyle(0,0,3);
      cleardevice();
```

```
        line(150,y-i*15,150,y-300-i*15);
        line(170,y-i*15-50,170,y-350-i*15);
        line(130,y-i*15-50,170,y-i*15-50);
        line(150,y-300-i*15,190,y-300-i*15);
        line(x-150,i*15,x-150,300+i*15);
        line(x-170,i*15-50,x-170,250+i*15);
        line(x-150,i*15,x-190,i*15);
        line(x-130,250+i*15,x-170,250+i*15);
        outtextxy(i*25,150,str);
        outtextxy(x-i*25,y-150,str);
        delay(5000);
    }
}
void graphtwo(char *str)
{   int i;
    for(i=0;i<30;i++)
    {   setcolor(5);
        cleardevice();
        settextstyle(1,1,4);
        line(i*25,y-100,300+i*25,y-100);
        line(i*25,y-120,300+i*25,y-120);
        line(x-i*25,100,x-300-i*25,100);
        line(x-i*25,120,x-300-i*25,120);
        outtextxy(150,i*25,str);
        outtextxy(x-150,y-i*25,str);
        delay(5000);
    }
}
void graphthree(char *str)
{   int i,j,color,width;
    color=random(maxcolor);          /*  随机得到颜色值*/
    setcolor(color);
    settextstyle(1,0,1);             /*  设置字符串的格式*/
    outtextxy(x/2,y/2-100,str);      /*  显示字符串*/
    delay(8000);
    for(i=0;i<8;i++)                 /*  字符串由小变大*/
    {
    cleardevice();                   /*清屏*/
    settextstyle(1,0,i);
    outtextxy(x/2,y/2-i*10-100,str);
    outtextxy(x/2,y/2+i*10-100,str);
    width=textwidth(str);            /*得到当前字符串宽度*/
    setlinestyle(0,0,1);             /*  设置画线格式*/
    line((x-width)/2+10*(8-i),y/2+i*15-70,(x+width)/2-10*(8-i),y/2+i*15-70);
    line((x-width)/2+5*(8-i),y/2+i*15-60,(x+width)/2-5*(8-i),y/2+i*15-60);
    line((x-width)/2,y/2+i*15-50,(x+width)/2,y/2+i*15-50);
    line((x-width)/2,y/2+i*15-20,(x+width)/2,y/2+i*15-20);
    line((x-width)/2+5*(8-i)-10,y/2+i*15-10,(x+width)/2-5*(8-i),y/2+i*15-10)
```

```
        line((x-width)/2+10*(8-i),y/2+i*15,(x+width)/2-10*(8-i),y/2+i*15);
        delay(8000);
        }
        for(i=7;i>=0;i--)                /*字符串由大变小*/
        {
            cleardevice();               /*清屏*/
            settextstyle(1,0,i);
            outtextxy(x/2,y/2-i*10-100,str);
            outtextxy(x/2,y/2+i*10-100,str);
            width=textwidth(str);
            setlinestyle(0,0,1);
            line((x-width)/2+10*(8-i),y/2+i*15-70,(x+width)/2-10*(8-i),y/2+i*15-70);
            line((x-width)/2+5*(8-i),y/2+i*15-60,(x+width)/2-5*(8-i),y/2+i*15-60);
            line((x-width)/2,y/2+i*15-50,(x+width)/2,y/2+i*15-50);
            line((x-width)/2,y/2+i*15-20,(x+width)/2,y/2+i*15-20);
            line((x-width)/2+5*(8-i),y/2+i*15-10,(x+width)/2-5*(8-i),y/2+i*15-10);
            line((x-width)/2+10*(8-i),y/2+i*15,(x+width)/2-10*(8-i),y/2+i*15);
            delay(8000);
        }
    }
```

程序中用到的库有 graphics.h、dos.h 和 stdlib.h，其中 graphics.h 中的图形函数，除 initgraph()、cleardevice()、closegraph()、settextjustify()、settextstyle()、setlinestyle()、outtextxy()、setcolor()和 line()外，还包括：

void far getmaxx (void);

功能：返回当前图形模式下的最大有效的 x 值（即最大的横坐标）。

void far getmaxy (void);

功能：返回当前图形模式下的最大有效的 y 值（即最大的纵坐标）。

void far getmaxcolor(void);

功能：返回当前图形模式下的最大有效的颜色值。

void far textwidth(char far *str);

功能：以像素为单位，返回由 str 所指向的字符串宽度，针对当前字符的字体与大小。该程序用到的 dos.h 中的库函数有 delay()，其原型说明为：

void delay(unsigned milliseconds);

功能：该函数将程序的执行暂停一段时间（毫秒）。

void far random(int num);

功能：此函数返回一个 0~num 范围内的随机数。该函数在 stdlib.h 的库中。

4.3.2 动态开辟图视窗口的方法

还可以利用图视窗口设置技术来实现图视窗口动画效果。具体方法是：在不同图视窗口中设置同样的图像，然后让图视窗口沿 x 轴方向移动设置，这次出现前要清除上次图视窗口的内容，这样就会出现图像沿 x 轴移动的效果。也就是说，在位置动态变化，但大小不变的图视窗口中［用 setviewpot()函数］，设置固定图形（也可是微小变化的图像），这样虽呈现在观察者面前的是当前图视窗口位置在动态变化，但视觉上却像是看到图像在屏幕上动态变化一样。

例 4-17 就是这样做的，不断地沿 x 轴开辟图视窗口，就像一个大小一样的窗口沿 x 轴在移动，由于总有 clearviewport 函数清除上次窗口的相同立方体，因而视觉效果上，就像一个立方体从左向右移动一样。程序中定义的 movebar 函数作用是开辟一个图视窗口，并画一个填色的立方体，保留一阵［delay(250000)］然后清除它，主程序不断调用它，因每次顶点 x 坐标在增加，因而效果是立方体沿 x 轴从左向右在运动。

例 4-17　动态开辟图视窗口。

```
#include <graphics.h>
#include <dos.h>
main()
{   int i,driver,mode;
    graphdriver=DETECT;
    initgraph(&driver,&mode,"");
    for(i=0;i<25;i++)
    {   setfillstyle(1, i);
        movebar(i * 20);
    }
    closegraph();
}
movebar(int xorig)      /*设窗口并画填色小立方体*/
{   setviewport(xorig,0,639,199,1);
    setcolor(5);
    bar3d(10,120,60,150,40,1);
    floodfill(70,130,5);
    floodfill(30,110,5);
    delay(250000);
    clearviewport();
}
```

采用上面的两种方法对较复杂图形不宜，一则画图形要占较长时间，二则图视窗口位置切换的时间就变得较长，因而动画效果就会变差。

4.3.3　屏幕图像存储再放的方法

在图形方式下，与文本方式类似，除了清屏函数 cleardevice()外，还有其他的对屏幕图像操作的函数，其中一类是屏幕图像存储和显示函数，包括：

1．存屏幕图像到内存区

void far getimage(int x1,int y1,int x2,int y2,void far *bitmap);

该函数将把屏幕左上角为（x_1，y_1），右下角为（x_2，y_2）矩形区内的图像保存到指针 bitmap 指向的内存区去。为了能开辟一个内存缓冲区，使它恰能存下所指矩形区中的图像，则必须首先要知道所存图像占多少字节，则内存缓冲区也可设这样多的字节，这可用下面的函数：

2．测定图像所占字节数的函数

unsigned far imagesize(int x1,int y1,int x2,int y2);

该函数将得到屏幕上左上角为（x_1，y_1），右下角为（x_2，y_2）矩形区内图像所占的字节数。

3．将所存图像显示的函数

void far putimage(int x1,int y1,void far *bitmap,int op);

该函数将把指针 bitmap 指向的内存区中所装图像，与屏上现有左上角为（x_1，y_1）的矩形区内图像进行 op 规定的操作（参见表 4-13）。该函数进行各种图像的逻辑操作如同二进制操作一样。

表 4-13　op 逻辑操作结果

符号名	值	含义
COPY_PUT	0	复制
XOR_PUT	1	进行异或操作
OR_PUT	2	进行或操作
AND_PUT	3	进行与操作
NOT_PUT	4	进行非操作

例 4-18 演示了表 4-13 中的逻辑操作，for 循环用来在屏幕上方产生连续的 5 个方框，方框中套一用洋红色填充的小方块，5 个图像全一样。循环结束后，又在屏幕下方画出两个框，小框用洋红色填充并在大框内。程序运行后，立即在屏上显示出上述图案，当按任一键后，则由函数 imagesize 得到屏幕下方大框套一填充框区域内图像所占字节数，然后由 malloc 函数按字节数分配一内存缓冲区 buffer，再由 getimage 函数将图像存到 buffer 中，然后复制到屏幕上方左边第一个框位置。按任一键后，又将 buffer 中图像和第二个框图像进行与操作后显示，再按任一键，buffer 中的图像又和第三个方框内图像进行或操作并显示，如此重复，则可将 5 种逻辑操作结果均显示在屏上。注意 COPY 和 NOT 操作将与原来屏上的图像无关，buffer 中图像经过这两种操作，将覆盖掉原屏上图像，并将结果进行显示。

例 4-18　表 4-13 中的逻辑操作。

```
#include <graphics.h>
main()
{  int i,j,driver,mode,size;
   void *buffer;
   driver=DETECT;
   initgraph(&driver,&mode,"");
   setbkcolor(BLUE);
   cleardevice();
   setcolor(YELLOW);
   setlinestyle(0,0,1);                /*用细实线 */
   setfillstyle(1,5);                  /* 用洋红实填充 */
   for(i=0;i<5;i++)                    /*  产生连续的 5 个方框中套小框的*/
   {  j=i *110;
      rectangle(80+j,100,130+j,150);   /* 产生小框且用洋红色填充  */
      floodfill(110+j,140,YELLOW);
      rectangle(50+j,100,130+j,180);   /*画大框*/
   }
   rectangle(50,340,100,420);          /* 产生一个小框 */
   floodfill(80,360,YELLOW);           /*  用洋红色填充 */
   rectangle(50,340,130,420);          /*  产生一个大框  */
   getch();
```

```
    size=imagesize(40,300,132,430);          /*取得（40，300）右下角（132，430）区域图像字节数*/
    buffer=malloc(size);                     /*分配缓冲区（按字节数）*/
    getimage(40,300,132,430,buffer);         /*存图像*/
    putimage(40,60,buffer,COPY_PUT);         /*重新复制*/
    getch();
    j=110;
    putimage(40+j,60,buffer,AND_PUT);        /*和屏上的图与操作*/
    getch();
    putimage(40+2*j,60,buffer,OR_PUT);
    getch();
    putimage(40+3*j,60,buffer,XOR_PUT);
    getch();
    putimage(40+4*j,60,buffer,NOT_PUT);
    getch();
    closegraph();
}
```

同制作幻灯片一样，将整个动画过程变成一个个片段，然后存到显示缓冲区内，当把它们按顺序重放到屏幕上时，就出现了动画效果，这可以用 getimage()和 putimage()函数来实现，这种方法较快，因它已事先将要重放的画面画好了。余下的问题，就是计算应在什么位置重放的问题了。

例 4-19　演示了利用这种方法产生的两个洋红色小球碰撞、弹回、又碰撞的动画效果。

```
#include <graphics.h>
main()
{   int i,driver,mode,size;
    void *buffer;
    driver=DETECT;
    initgraph(&driver,&mode,"");
    setbkcolor(BLUE);
    cleardevice();
    setcolor(YELLOW);
    setlinestyle(0,0,1);
    setfillstyle(1,5);
    circle(100,200,30);
    floodfill(100,200,YELLOW);               /*填充圆*/
    size=imagesize(69,169,131,231);          /*指定图像占字节数*/
    buffer=malloc(size);                     /*分配缓冲区（按字节数）*/
    getimage(69,169,131,231,buffer);         /*存图像*/
    putimage(500,169,buffer,COPY_PUT);       /*重新复制*/
    do{
        for(i=0;i<185;i++)
            { putimage(70+i,170,buffer,COPY_PUT);     /*左边球向右运动*/
              putimage(500-i,170,buffer,COPY_PUT);    /*右边球向左运动*/
            }/*两球相撞后循环停止*/
        for(i=0;i<185;i++)
            { putimage(255-i,170,buffer,COPY_PUT);    /*左边球向左运动*/
              putimage(315+i,170,buffer,COPY_PUT);    /*右边球向右运动*/
            }
```

```
    }while (!kbhit());    /*当不按键时重复上述过程*/
  getch();
  closegraph();
}
```

4.3.4 利用页交替的方法

对屏幕图像操作的函数，还有一类是设置显示页函数。曾在前面提到了显示适配器的显示存储器 VRAM，图形方式下存储在 VRAM 中的一满屏图像信息称为一页。每个页一般为 64KB，VRAM 可以存储要显示的图像几个页（视 VRAM 容量而定，最大可达 8 页），Turbo C 2.0 支持页的功能有限，按在图形方式下显示的模式最多支持 4 页（EGALO 显示方式），一般为两页（注意对 CGA 仅有一页），因存储图像的页显示时，一次只能显示一页，因此必须设定某页为当前显示的页（又称可视页），缺省时定为 0 页，如图 4-3 所示。

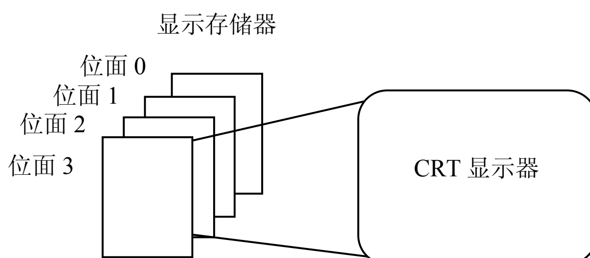

图 4-3 显示页

正在由用户编辑图形的页称为当前编辑页（又称激活的页），这个页不等于显示页，即若用户不设定该页为当前显示页时，在该页上编辑的图形将不会在屏上显示出来。缺省时，设定 0 页为当前编辑页，即若不用下述的页设置函数进行设置，就认定 0 页既是编辑页又是当前显示页。

设置激活页和显示页的函数如下：

void far setactivepage(int pagenum);

void far setvisualpage(int pagenum);

这两个函数只能用于 EGA、VGA 等显示适配器。前者设置由 pagenum 指出的页为激活的页，后者设置可显示的页。当设定了激活的页，即编辑页后，则程序中其后的画图操作均在该页进行，若它不定为显示页，则其上的图像信息并不会在屏上显示出来。

例 4-20 下面的程序演示了设置显示页函数的应用。首先用 setactivepage(1)设置 1 页为编辑页，在上面画出一个红色边框、用淡绿色填充的圆，此图并不显示出来（因缺省时，定义 0 页为可视页）。接着又定义 0 页为编辑页并清屏（即清 0 页），也定义 0 页为可视页，并在其上画出一个用洋红色填充的方块，该方块将在屏上显示出来。接着进入 do 循环，设置 1 页为可视页，因而其上的圆便在屏上显示出来，方块的图像消失，用 delay(2000)将圆图像保持 2000 毫秒即 2 秒，当不按键时，下一次循环又将 0 页设为可视页，因而方块的图像显示出来，圆图像又消失。保持 2 秒后，又重复刚开始的过程。这样就会看到：屏上同一位置洋红色圆和淡绿色方块交替出现，若将 delay 时间变少，将会出现动画的效果。

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

```
main()
{   int i,graphdriver,graphmode,size,page;
    graphdriver=DETECT;
    initgraph(&graphdriver,&graphmode,"");
    cleardevice();
    setactivepage(1);               /*设置 1 页为编辑页 */
    setbkcolor(BLUE);
    setcolor(RED);
    setfillstyle(1,10);
    circle(130,270,30);             /* 画圆*/
    floodfill(130,270,4);           /*用淡绿色填充圆 */
    setactivepage(0);               /*设置 0 页为编辑页 */
    cleardevice();                  /*清 0 页 */
    setfillstyle(1,5);
    bar(100,210,160,270);           /*画方块并填充洋红色 */
    setvisualpage(0);               /*设置 0 页为可视页*/
    page=1;
    do
    {   setvisualpage(page);        /*显示设定页的图像*/
        delay(2000);                /*延迟 2000ms */
        page=page-1;
        if(page<0)
        page=1;
    } while(!kbhit());
    getch();
    closegraph();
}
```

正如上面的程序所示，将当前显示页和编辑页分开［用 setvisualpage()和 setactivepage()函数］，在编辑页上画好图形后，立即令该页变为显示页显示，然后在上次的显示页上（现在变为编辑页）进行画图，画好后，又再次交换，如此编辑页和显示页反复地交换，在观察者的视觉上，就出现了动画的效果。要让页的交替速度快，唯一的办法是缩短在页上的画图时间。

4.4　中断技术

键盘原理：在键盘上按下一个键位时，键盘上的处理器首先向计算机主机发出硬件中断请求，然后将该键位的扫描码以串行的方式传送给计算机主机，计算机主机在硬件中断的作用下，调用 INT09H 中断把键盘送来的扫描码读入，并转换为 ASCII 码值存入键盘缓冲区。按下一个键位，送出一个闭合码，键位被释放时送出一个断开码，键盘处理中断程序从键盘 I/O 端口（端口地址为 60H）读取一个字节的数据，如果读取的数据的第 7 位为 1 时表示按键已放开（送出断开码），如第 7 位为 0 表示键按下（送出闭合码），数据的第 0 ～6 位则是按键的扫描码。在编程中是通过 bioskey()函数使用原有的键盘中断程序。使用 PC 原有的键盘处理程序可以很方便地处理键盘，但是因为它是调用 BIOS，所以反应比较慢，另外当要同时处理几个按键时，原有的键盘中断程序就不能满足要求，这时就需要编写一个适合要求的键盘中断程序。

bioskey()库函数实际是调用了 BIOS 中断 INT16H 的功能，同样也可通过 int86()直接调用 INT16H。总之，还是调用了系统的中断服务程序。在大型程序的实现中往往需要自己编写中断。在中断向量表中，有些中断向量，用户是可以在自己的程序中使用的，如 60～67H 号中断。绝对地址为 180～19FH 的一段，是专为用户保留的，它可用作用户的软中断。例如，想使用中断号为 60H 的软中断，那么用户只要在 4×60H=180H 和 181H 处填入中断服务程序的偏移地址，在 182H～183H 单元填入段地址即可。这样在程序中执行到 geninterrupt(0x60)时，便执行这个中断服务程序。对于硬件中断，若要通过 IRQi 线产生硬中断，用户必须制作接口电路，以能产生由低到高的中断请求信号，并通过插头接到微机的扩充槽相应的 IRQ5 脚上，用户可用的是 IRQ2、IRQ10、IRQ11、IRQ12 和 IRQ15，它们的中断向量号分别是 10、72、73、74 和 77，因此可以在中断向量表中填入硬中断服务程序的段和偏移值。上述的硬件中断向量号若不用作硬件中断时，也可将其当作软件中断，不过产生中断要靠发软件中断命令，此时相应的 IRQi 脚就无作用了。

这部分将介绍这一更高级的中断技术，即如何用 Turbo C 实现自己的中断服务。

4.4.1 编写中断程序

用 Turbo C 2.0 实现编写中断程序的方法可用 3 部分来实现，即编写中断服务程序、安装中断服务程序、激活中断服务程序。

我们的任务是，当产生中断后，脱离被中断的程序，使系统执行中断服务的程序，它必须打断当前执行的程序，急需完成一些特定操作，因此该程序中应包括一些能完成这些操作的语句和函数。由于产生中断时，必须保留被中断程序中断时的一些现场数据，即保存断点，这些值都在寄存器中（若不保存，当中断服务程序用到这些寄存器时，将改变它的值），以便恢复中断时，使这些值复原，以继续执行原来中断了的程序。

Turbo C 2.0 为此提供了一种新的函数类型 interrupt，它将保存由该类型函数参数指出的各寄存器的值，而在退出该函数，即中断恢复时，再复原这些寄存器的值，因而用户的中断服务程序必须定义成这种类型的函数。如中断服务程序名定为 myp，则必须将这个函数说明成这样：

```
void interrupt myp(unsigned bp,unsiened di,unsigned si,unsigned ds,
unsigned es,unsigned dx,unsigned cx,unsigned bx,
unsigned ax,unsigned ip,unsigned cs,un3igned flags);
```

若是在小模式下的程序，只有一个段，在中断服务程序中用户就可以像用无符号整数变量一样，使用这些寄存器。若中断服务程序中不使用上述的寄存器，也就不会改变这些寄存器原来的值，因而也就不需保存它们，这样在定义这种中断类型的函数时，可不写这些寄存器参数，如可写成：

```
void interrupt myp()
{
    …
}
```

对于硬中断，则在中断服务程序结束前要送中断结束命令字给系统的中断控制寄存器，其口地址为 0x20，中断结束命令字也为 0x20，即

```
outportb(0x20,0x20);
```

在中断服务程序中，若不允许别的优先级较高的中断打断它，则要禁止中断，可用函数 disable()来关闭中断。

若允许中断，则可用开中断函数 enable()来开放中断。

4.4.2　安装中断服务程序

定义了中断服务函数后，还需将这个函数的入口地址填入中断向量表中，以便产生中断时程序能转入中断服务程序去执行。为了防止正在改写中断向量表时，又产生别的中断而导致程序混乱，可以关闭中断，当改写完毕后，再开放中断。一般的，常定义一个安装函数来实现这些操作，如：

```
void install(void interrupt (*faddr)(),int inum)
{
    disable();
    setvect(intnum,faddr);
    enable();
}
```

其中 faddr 是中断服务程序的入口地址，其函数名就代表了入口地址，而 inum 表示中断类型号。setvect()函数就是设置中断向量的函数，上述定义的 install()函数，将完成把中断服务程序入口地址填入中断向量 inum 中去。

setvect(intnum，faddr)：把第 intnum 号中断向量指向所指的函数，也即指向 faddr()这个函数。注意：faddr()必须是一个 interrupt 类型函数。

getvect(intnum)：返回第 int num 号中断向量的值，即 intnum 号中断服务程序的进入地址（4B 的 far 地址）。该地址是以 pointer to function 的形式返回的。

getvect()使用的方式为：ivect =getvect(intnum);其中 ivect 是一个 function 的 pointer，存放 function 的地址，其类型必须是 interrupt 类型，由于声明时，此 pointer 无固定值，所以冠以 void。

4.4.3　中断服务程序的激活

当中断服务程序安装完后，如何产生中断，从而执行这个中断服务程序呢？对硬件中断，就要在相应的中断请求线（IRQi，i=0，1，2，…，7）产生一个由低到高的中断请求电平，这个过程必须由接口电路来实现，但如何激励它产生这个电平呢？可以用程序来控制实现，如发命令［outportb(口地址,命令)］。然后主程序等待中断，当中断产生时，便去执行中断。

由于中断类型的函数不同于用户定义的一般函数，因此也不能用调用一般函数的方法来调用它，这样一般软中断调用可用以下方法：

（1）使用库函数geninterrupt(中断类型号)

在主函数中适当的地方，用 setvect 函数将中断服务程序的地址写入中断向量表中，然后在需要调用的地方用 geninterrupt()函数调用。

（2）直接调用

如已用 setvect(类型号，myp)设置了中断向量值，则可用 myp();直接调用，或用指向地址的方法调用：

```
(* myp)();
```

（3）也可用在 Turbo C 程序中插入汇编语句的方法来调用，如：

```
setvect (inum,myp)
...
asm int inum;
...
```

不过用这种方法的程序生成执行程序稍麻烦点。

通常上述的调用可定义成一个中断激活函数来完成，该函数中可附加一些别的操作，主程序在适当的地方调用它就可以了。

（4）恢复被修改的中断向量

这一步视情况而定，当用户采用系统已定义过的中断向量，并且将其中断服务程序进行了改写，或用新的中断服务程序代替了原来的中断服务程序，为了在主程序结束后，恢复原来的中断向量以指向原中断服务程序，可以在主程序开始时，存下原中断向量的内容，这可以用取中断向量函数 getvect()来实现，如 j=(char *)getvect(0xlc)，这样 j 指针变量中将是 0xlc 中断服务程序的入口地址，由于 DOS 已定义了 0xlc 中断的服务程序入口地址，但它是一条无作用的中断服务，因而可以利用 0xlc 中断来完成一些用户想执行的操作，实际上就是用户自己的中断服务程序代替了原来的。当主程序要结束时，为了保持系统的完整性，可以恢复原来的中断服务入口地址，如可用 setvect(0xlc，j)，也可以调用 install()函数再一次进行安装。一般情况下可以不加这一步。

PC286、386、486 上使用的硬中断请求信号和对应的中断向量号与外设的关系如表 4-14 所示。

表 4-14　C X86 硬中断搞求信号与中断向量号及外设的关系

中断请求信号	中断向量号	使用的设备
IRQ0	8	系统定时器
IRQ1	9	键盘
IRQ2	10	对 XT 机保留，AT 总线扩充为 IRQ8～15
IRQ3	11	RS-232C（COM2）
IRQ4	12	RS-232C（COMl）
IRQ5	13	硬盘中断
IRQ6	14	软盘中断
IRQ7	15	打印机中断
IRQ8	70	实时时钟中断
IRQ9	71	软中断方式重新指向 IRQ2
IRQ10	72	保留
IRQ11	73	保留
IRQ12	74	保留
IRQ13	75	协处理器中断
IRQ14	76	硬盘控制器
IRQ15	77	保留

中断优先级从高到低排列顺序为 IRQ0、IRQl、IBQ 2、IRQ8、IRQ9、IRQ10、IRQ1l、IRQ12、

IRQ13、IRQ14、IRQ15、IRQ3、IRQ4、IRQ5、IRQ6 和 IRQ7。

4.4.4　应用——硬中断演示秒表程序

例 4-21 是一个硬中断程序，我们知道 PC 系统以每秒 18.2 次的频率进行时钟硬中断（使用中断请求 IRQ8），即执行 8 号中断。这个中断周而复始地在进行，在它的中断服务程序中除了进行日时钟计数和磁盘驱动器超时检测控制外，接着又进行 0xlc 的软中断调用，0xlc 软中断只有一条返回指令，它不做什么事情，因而可以改写它的内容，使其变为一个有用的软中断服务程序。在这个例子中，利用每秒 18.2 次的定时硬中断。每秒要调用 18.2 次的软中断 0xlc，将中断 0xlc 中断服务程序改写为对进入该中断的次数进行计数的程序，每到 18 次时，在屏幕的右上角开一个窗口（window(50，1，54，3)），在窗口的中间位置显示 0～9 十个数字中的一个，频率接近于秒表数（不过只显示 10 个数）。由于这是一个硬中断演示程序，计时并不准确，若要精确计时，则应 91 次 0xlc 中断为 5 秒。

该程序中用 programsize 表示程序长度，设置为 400 节，由于在小模式下，Turbo C 使用 64K 的一个段。

例 4-21　一个硬中断程序。

```
#include <dos.h>
#include <conio.h>
#define   programsize  400
#define TRUE 1
void interrupt(*oldtimer)();
void interrupt newtimer();
void install();
static struct SREGS seg;
int b=0;int b1=0;
unsigned intsp,intss;
unsigned myss,stack,x0,y0;
int busy=0;
void on_timer();
void goxy();
void rexy();
void prt();
main()
{   char ch;
    char *p;
    p=(char *)newtimer;
    on_timer(p);
    while(TRUE)
    {   ch=getch();
        switch(ch)
        {   /*键盘输入 0、1 或 q 进行功能选择*/
            case '0': install(oldtimer); break;
            case '1': b1=47; install(newtimer); break;
            case 'q': install(oldtimer); exit(1);
            default: printf("%c",ch);     /*若是其他字符就打印出来*/
        }
```

```
    }
}
void on_timer(int (*p)())                /* p 指向中断函数*/
{   segread(&seg);
    stack=programsize*16;                /*设置堆栈*/
    myss=_SS;
    oldtimer=getvect(0x1c);              /*得到原 0xlc 中断向量*/
}
void install(void interrupt  (* faddr) () )
{   disable();
}
setvect(0x1c,faddr);
enable();
/*设置 0xlc 新中断向量*/
void interrupt newtimer()
{
(* oldtimer)();                          /* 指向原中断程序*/
if(busy==0)
{   busy=1;
    disable();
    intsp=_SP;
    intss=_SS;                           /*保存 SP、SS 值*/
    _SP=stack;
    _SS=myss;                            /*设置 SP、SS 为新值*/
    enable();
    rexy();                              /*得到当前光标位置*/
    clrscr();
    window(50,1,54,3);
    textcolor(YELLOW);
    textbackground(RED);
    b+=1;                                /* 如果中断 18 次，则打印秒计数值 */
    if(b==18)
    {   gotoxy(3,2);
        b1++;
        b=0;
        if(b1>57) b1=48;                 /*计数值超过 9，则令 bl 为 0 */
        prt(b1);                         /*显示计数值*/
    }
    goxy(x0,y0);                         /*光标返回原处*/
    disable();
    _SP=intsp;                           /*恢复 SP、SS 的原值*/
    _SS=intss;
    enable();
    busy=0;
    }
}
void goxy(int x,int y)                   /*光标回到 x、y 处*/
{   union REGS rg;
```

```
    rg.h.ah=2;
    rg.h.dl=y;
    rg.h.dh=x;
    rg.h.bh=0;
    int86(0x10,&rg,&rg);
}
void rexy()                         /*得到光标坐标*/
{   union REGS rg;
    rg.h.ah=3;
    rg.h.bh=0;
    int86(0x10,&rg,&rg);
    y0=rg.h.dl;
    x0=rg.h.dh;
}
 void prt(p)                        /*显示字符*/
{   union REGS rg;
    rg.h.al=p;
    rg.h.ah=14;
    int86(0x10,&rg,&rg);
}
```

4.5　发声技术

播放歌曲就是意味着让计算机发声，由于声音是从 PC 内的扬声器发出的，因此要首先了解一下计算机发声原理。在 PC 的系统板上装有定时与计数器 8253 芯片，还有 8255 可编程并行接口芯片，由它们组成的硬件电路可用来产生 PC 内扬声器的声音，对于 286、386、486、586 等 PC，由于采用了超大规模集成电路，因而看不到这些芯片，它们均集成在外围电路芯片上了。当操作计算机时，常常听到的发声，就是由软件控制这些电路而产生的。声音的长短和音调的高低，均可由程序进行控制。在扬声器电路中，定时器的频率决定了扬声器发音的频率，所以可通过设定定时器电路的频率来使扬声器发出不同的声音。对定时器电路进行频率设定时，首先对其命令寄存器(口地址为 0x43)写命令字，如写入 0xb6，这可用 outporb(0x43,0xb6);来实现，则表示选择该定时器的第二个通道，计数频率先送低 8 位（二进制），后送高 8 位。接着用口地址 0x42 送频率计数值，先送低 8 位，后送高 8 位，即用 outportb（0x42，低 8 位频率计数值）和 outportb（0x42，高 8 位频率计数值）来实现。通过这两步使定时器电路产生一系列方波信号，此信号是否能推动扬声器发声，还要看由 8255 产生的门控信号和送数信号是否为 1，而它们也可编程，口地址为 0x61。为了不影响 8255 口地址 61H 中的其他高位，应先输入口地址 61H 的现有值 bits，即用 bits= inportb(0x61)来实现，然后就可用 outportb(0x61，bits|3)来允许发声，而用 outportb(0x61，bits&0xfc)来禁止发声，且不改变 8255 其他位原来的值。

4.5.1　声音函数

编写音乐程序播放歌曲，最简单的方法是可以直接使用 Turbo C 2.0 在 dos.h 中提供的有关发声的函数 sound()和 nosound()。

void sound(unsigned frequency);

该函数用于产生声音，其入口参数为扬声器要产生声音的频率。

void nosound(void);

该函数关闭扬声器，该函数没有入口和出口参数，它只是简单地把口地址 61H 中的低 2 位清 0。

在利用函数 sound 产生指定频率的声音后，一般要过一段时间后再调用函数 nosound 关闭扬声器，这样才能清楚地听到一个声音。如果扬声器刚打开就关闭，是很难听到一个声音的。某个频率的声音延续时间的长短是很重要的，它将直接影响音响效果。这需要使用 Turboc 提供的专门的延时函数 delay，其原型说明如下：

void delay (unsigned milliseconds);

该函数中断程序的执行，中断的时间由 milliseconds 指定。

例 4-22 该程序每间隔 10000 milliseconds PC 扬声器发出不同频率的声音，直到频率为于 5000Hz。

```
#include<dos.h>
main()
{   int freq;
    for(freq=50;freq<5000;freq+=50)
    { sound (freq);
        delay(10000);
    }
    nosound();
}
```

4.5.2 乐谱的计算机表示方法

音乐程序设计中有两个最重要的因素：音符和音长。进行音乐程序设计时必须要解决如何用“曲调定义语言”来表示音符和如何控制音符的持续时间问题。

1．音符

音调由音符构成，音调的高低由音频决定，频率越高，音调也越高。音乐中使用的频率一般为 131～1976Hz，它包括中央 C 调及前后的 4 个 8 度音程。这 4 个 8 度中各音符的频率如表 4-15 所示。

表 4-15 频率与音阶的对照表

音调	低音	频率	中音	频率	高音	频率	最高音	频率
C	1	131	1	262	1	523	1	1047
D	2	147	2	296	2	587	2	1175
E	3	165	3	330	3	659	3	1319
F	4	176	4	349	4	699	4	1397
G	5	196	5	392	5	784	5	1568
A	6	220	6	440	6	880	6	1760
B	7	247	7	494	7	988	7	1976

为了在程序中输入方便，常用英文字母表示音符的频率。用 asdfghj 表示中音段

CDEFGAB；zxcvbnm 代表低音段 CDEFGAB；qwertyu 代表高音段 CDEFGAB。对于很不常用的最高音的 7 个音符，可以再用 7 个其他符号代替。

表示音符频率的另一个方案是用枚举类型常量来定义表 4-15 中各音符的频率。

例如，定义 enum music

```
{
    C0=131, D0=147, E0=165, F0=175, G0=196, A0=220, B0=247,
    C1=262, D1=294, E1=330, F1=349, G1=392, A1=440, B1=494,
    C2=523, D2=587, E2=659, F2=698, G2=784, A2=880, B2=988,
    C3=1047,D3=1175,E3=1319,F3=1397,G3=1568,A3=1760,B3=1976,
}
```

2. 音长

音长表示音符的持续时间。演奏时，每一个音符必须有一个频率用 sound 去发声，且必须有适当的时间延时，形成拍子。在乐曲中，音长分全音符、半音符、4 分音符、8 分音符等。通常以 4 分音符为一拍，这样可以用 1 拍的时间来推出其他节拍。

为了在程序中表示音长，一般可以有 2 个方案来解决这个问题。一是在曲谱中找出持续最短的音符，把它作为一个字母写出，然后时长是其一倍的就用两个同样的字母表示，以此类推，两倍的用 3 个同样的字母表示，4 倍的就用 4 个同样的字母表示。

例如，对于 **2.3 1 16|56 5** 两小节乐谱，可以表示成：

s s s d a a n bb n n bbbb。

另一个方案是采用宏定义来解决：

```
# define m1 32
# define m2 m1/2
# define m4 m1/4
# define m8 m1/8
# define m16 m1/16
```

上述两小节乐谱，可以表示成如表 4-16 所示。

表 4-16　乐谱示例

音符	音高	音长
2	D1	m8+m16
3	E1	m16
1	C1	m8
1	C1	m16
6	A0	m16
5	G0	m8
6	A0	m8
5	G0	m4

知道了这些知识，就容易编制一个乐谱程序。

4.5.3　应用

例 4-23　《雪绒花》歌曲程序。

源代码如下：

```
# include<conio.h>
# include<dos.h>
# define speed 2
void sound1(int freq,int time);
void pause(int time);
main()
{   int i,freq;
    int time=4*speed;
    char *qm="iddgwwwqqgfffddddfghhhggg ddgwwwqqgfff\
    ddgghjqqqqqwpggjhgddgqqq hhqwwqjjjggg\
    ddgwwwqqgfffddgghjqqqqqq";
    gotoxy(40,20);
    cprintf("Snow Flower");
    while(*qm++!='\0')
    {   i=1;
        switch(*qm)
        {   case'k':   time=1*speed; i=0; break;
            case 'l':   time=2*speed; i=0; break;
            case 'i':   time=4*speed; i=0; break;
            case 'o':   time=6*speed; i=0; break;
            case 'p':   pause(time); i=0; break;
            case 'a':   freq=523; break;
            case 's':   freq=587; break;
            case 'd':   freq=659; break;
            case 'f':   freq=698; break;
            case 'g':   freq=784; break;
            case 'h':   freq=880; break;
            case 'j':   freq=988; break;
            case 'z':   freq=262;break;
            case 'x':   freq=294;break;
            case 'c':   freq=330;break;
            case 'v':   freq=349;break;
            case 'b':   freq=392;break;
            case 'n':   freq=440;break;
            case 'm':   freq=494;break;
            case 'q':   freq=1047;break;
            case 'w':   freq=1175;break;
            case 'e':   freq=1319; break;
            case 'r':   freq=1397; break;
            case 't':   freq=2568; break;
            case 'y':   freq=1760; break;
            case 'u':   freq=1976; break;
            default:   i=0;   break;
        }
        if(i)
        sound1(freq,time);
```

```
    }
}
void sound1(int freq,int time)
{   int n;
    sound(freq);
    n=time+clock();
    while(n>clock());
    nosound();
}
void pause(int time)
{   int n;
    n=time+clock();
    while(n>clock())
    nosound();
}
```

注释：clock()测得从程序开始到调用处处理机所用的时间，其头文件为 time.h。

函数 sound()可以用指定的频率控制机内扬声器发出指定的声响，直到用函数 nosound()去关闭它。另外，分别用 k、l、i 表示 1/2、1/4、3/2 节拍。P 表示休止符。演奏速度由 SPEED 控制。具体识音过程采用 switch case 语句。本例中 i=0 表示不发音；i=1 表示发音。

例 4-24　**2.3 1 16|56 5** 两小节计算机乐谱。

```
# include <stdio.h>
#include<dos.h>
# define m1    4000000000
# define m2    4000000000/2
# define m4    4000000000/4
# define m8    4000000000/8
# define m16 4000000000/16
enum music
{   C0=131, D0=147, E0=165, F0=175, G0=196, A0=220, B0=247,
    C1=262, D1=294, E1=330, F1=349, G1=392, A1=440, B1=494,
    C2=523, D2=587, E2=659, F2=698, G2=784, A2=880, B2=988,
    C3=1047,D3=1175,E3=1319,F3=1397,G3=1568,A3=1760,B3=1976,
}s1[ ]={D1,E1,C1,C1,A0,G0,A0,G0};
main()
{   int i ;
    int s2[ ]={m8+m16,m16,m8,m16,m16,m8,m8,m4};
    for(i=0;i<8;i++)
    {     sound(s1[i]);
          delay(s2[i]);
    }
    nosound();
}
```

4.6 数据库

4.6.1 编程中如何解决数据的保存问题

在编写大型软件时，系统中经常涉及大量数据，例如学籍管理系统方面的学生学籍情况就是一个很大的数据集合，该数据集合一旦建立后就需要保留，以后每次运行学籍管理系统，都可以对此数据集合进行增、减、改等操作，而不必每次都重新建立。那么采用什么办法来保存系统中大量数据呢？在 C 语言中最常采用的方法是利用文件类型，将数据以文件的形式保存在磁盘里，这样就可以达到长期保存数据的目的。由于可以对文件进行读出、写入、修改、增加和删除等操作，因而也就可以实现对数据库进行插、删、改等操作。

在系统运行中使用的大批数据，主要通过两种数据结构来暂存，一种是数组结构，另一种是链表结构。

下面以数字类型为例，进一步说明利用文件类型长期保存数据的方法。需要建立一个数据文件。

例 4-25 从键盘输入一组数据，然后把它们保存在数据文件 in4.dat 中。程序如下：

```
# include<conio.h>
# include<stdio.h>
# include<string.h>
main()
{  FILE  * rt;
   int  x, i
   cirscr();
   if ((rf = fopen("in4.dat","r"= =NULL)
   {  printf("Cannot open file\n");
      return;
   }
   for(i = 0 ; i<10 ; i ++)
   {  fscanf(rf, "%d", &x);
      printf("%d",x);
   }
   fclose(rt);
}
```

上面的例子比较简单，这里需要着重强调一点：查看保存在 in4.dat 文件中的数据的方法。在 Turbo C 环境下，选择 File→Open 命令，只需要将文件名及扩展名 dat 完整输入进去，即可在编辑窗口显示文件内容。

4.6.2 数据库的发展

虽然可以用 C 语言开发一个简单的数据库管理软件，但是它与专业化的数据库管理系统无论在规模上还是在复杂性上都有天壤之别，不过用 C 语言开发简单的数据库管理软件对全面了解和运用 C 语言的知识却是大有益处的。为了更好地解决如何用 C 语言编制数据库的问题，本节有必要对数据库的发展有个大概的了解。

我们说数据管理指的就是对数据的组织、编目、存储、检索和维护。随着计算机技术的发展，数据管理也经历了由低级向高级的发展过程。

- 人工管理阶段（20 世纪 50 年代中期以前）。
- 文件管理阶段（20 世纪年代后期至 60 年代后期）。
- 数据库系统阶段（20 世纪 70 年代初以后）。

早期的数据管理就是人工处理，即通过人工对数据进行组织、编目、存储、检索和维护，这一切都需要人对处理数据的物理结构了解清楚，这个阶段耗时费力，工作量非常大。

到了文件管理阶段，由于可以通过文件系统与设备介质来管理和保存信息，并把信息的逻辑结构映像成设备介质上的物理结构。这样就可以使用户不必过多地考虑物理细节，而将精力集中于算法上。

文件系统中的文件基本上是对应于一个或几个应用程序，或者说数据是面向应用的。它仍然是一个不具有弹性的无结构信息集合，因此也就存在以下几个方面的问题：

（1）冗余度大。

文件系统下的用户各自建立自己的文件，相互之间数据不能共享，造成数据大量重复存储。不仅浪费存储空间，更严重的是容易造成数据的不一致性。

（2）数据独立性差。

数据和程序相互之间的依赖仍较严重。

（3）数据无集中管理。

各个文件没有统一的管理机构，其安全性和完整性等无法得到保证。

所有这些问题，文件系统本身都无法解决，这严重地阻碍了数据处理技术的进展，同时，也成为数据库技术产生的原动力和背景。于是数据库系统便应运而生，并在 20 世纪 60 年代末期诞生了第一个商品化的数据库系统——美国 IBM 公司的 IMS（Information Management System）系统。

数据库系统的目标首先就是克服文件系统的这些弊病，用一个软件来集中管理所有的文件，以实现数据的共享，保证数据的完整性、安全性。

4.6.3　数据库系统的特点

与文件系统比较，数据库系统有以下特点：

（1）数据的结构化。

在文件系统中，各个文件不存在相互联系。从单个文件来看，数据一般是有结构的；但是从整个系统来说，数据在整体上又是没有结构的。数据库系统则不同，在同一数据库中的数据文件是有联系的，且在整体上服从一定的结构形式。

（2）数据共享。

共享是数据库系统的目的，也是它的重要特点。一个库中的数据不仅可为同一企业或机构之内的各个部门所共享，也可为不同单位、地域甚至不同国家的用户所共享。而在文件系统中，数据一般是由特定用户专用的。

（3）数据的独立性。

在文件系统中，数据结构和应用程序相互依赖，一方的改变总是要影响另一方的改变。数据库系统则力求减小这种相互依赖，实现数据的独立性。虽然目前还未能完全做到这一点，但较之文件系统已大有改善。

（4）可控冗余度。

数据专用时，每个用户拥有并使用自己的数据，难免有许多数据相互重复，这就是冗余。实现共享后，不必要的重复将全部消除，但为了提高查询效率，有时也保留少量重复数据，其冗余度可由设计人员控制。

表 4-17 以对照表的的形式，列出了数据库系统与一般文件应用系统的主要性能差别。由此可以看出，用 C 语言编写的数据库管理软件就属于文件管理阶段。

表 4-17 数据库系统与一般文件应用系统性能对照表

序号	文件应用系统	数据库系统
1	文件中的数据由特定的用户专用	库内数据由多个用户共享
2	每个用户拥有自己的数据，导致数据重复存储	原则上可消除重复，为方便查询允许少量数据重复存储
3	数据从属于程序，二者相互依赖	数据独立于程序，强调数据的独立性
4	各数据文件彼此独立，从整体看为“无结构”	各文件的数据相互联系，从总体看是“有结构”

4.6.4 数据库基本概念

数据库技术是一门综合性技术，它涉及操作系统、数据结构、算法设计、程序设计等基础理论知识；它是计算机科学中一项专门的学科。它的基本知识主要有数据库、数据库系统、数据库管理系统、数据模型等方面的基本概念。

但是这里谈的基本概念只是与文件应用系统有关的数据库的最底层概念。

数据：通常使用各种各样的物理符号来表示客观事物的特性和特征，这些符号及组合就是数据。数据的概念包括两个方面：数据内容和数据形式。

数据内容：是指所描述客观事物的具体特性，即数据的“值”。

数据形式：是指数据内容存储在媒体上的具体形式，即数据的“类型”。数据主要有数字、文字、声音、图形和图像等多种形式。

实体：客观事物在信息世界中称为实体，它是现实世界中任何可区分、可识别的事物。实体可以是具体的人或物，如张三同学、石景山业余大学；也可以是抽象概念，如一个人、一所学校。

属性：实体的特性称为属性。一个实体可用若干属性来刻画。每个属性都有特定的取值范围，即值域。值域的类型可以是整数型、实数型、字符型等。

实体型：实体型就是实体的结构描述，通常是实体名和属性名的集合；具有相同属性的实体，有相同的实体型。实体值是一个具体的实体，是属性值的集合。如学生实体型是：学生（学号，姓名，性别，年龄）；学生李建国的实体值是：（011110，李建国，男，19）。

属性值：就是属性在其值域中所取的具体值。如“李建国”是学生实体中的姓名的属性值。

关系：在数据库理论中，关系是指由行与列构成的二维表。在关系模型中，实体和实体间的联系都是用关系表示的。也就是说，二维表格中既存放着实体本身的数据，又存放着实体间的联系。

关系模型：是建立在关系代数基础上的，具有坚实的理论基础。与层次模型和网状模型

相比，具有数据结构单一、理论严密、使用方便、易学易用的特点。目前，绝大多数数据库系统的数据模型均采用关系模型。

4.6.5　文件存储

4.6.5.1　文件概念

对于数据库来说，一般还要求把单表中的数据存储在外部介质（比如硬盘）上。这就要求对文件存储类的函数功能也要有个详细的了解。

所谓“文件”是指一组相关数据的有序集合。这个数据集有一个名称，叫做文件名。实际上在前面的各章中已经多次使用了文件，如源程序文件、目标文件、可执行文件、库文件（头文件）等。文件通常是驻留在外部介质（如磁盘等）上的，在使用时才调入内存中。从不同的角度可对文件作不同的分类。从用户的角度看，文件可分为普通文件和设备文件两种。

普通文件是指驻留在磁盘或其他外部介质上的一个有序数据集合，可以是源文件、目标文件、可执行程序；也可以是一组待输入处理的原始数据，或者是一组输出的结果。对于源文件、目标文件、可执行程序可以称为程序文件，对输入/输出数据可称为数据文件。

设备文件是指与主机相连的各种外部设备，如显示器、打印机、键盘等。在操作系统中，把外部设备也看作是一个文件来进行管理，把它们的输入、输出等同于对磁盘文件的读和写。通常把显示器定义为标准输出文件。如前面经常使用的 printf、putchar 函数就是这类输出。键盘通常被指定标准的输入文件，从键盘上输入就意味着从标准输入文件上输入数据。scanf、getchar 函数就属于这类输入。

从文件编码的方式来看，文件可分为 ASCII 码文件和十进制码文件两种。

ASCII 文件也称为文本文件，这种文件在磁盘中存放时每个字符对应一个字节，用于存放对应的 ASCII 码。例如，数 5678 的存储形式如表 4-18 所示。

表 4-18　数 5678 的存储形式

ASCII 码	00110101	00110110	00110111	00111000
十进制码	5	6	7	8

十进制码：5678 共占用 4 个字节。ASCII 码文件可在屏幕上按字符显示，比如源程序文件就是 ASCII 文件，用 DOS 命令 TYPE 可显示文件的内容。由于是按字符显示，因此能读懂文件内容。

二进制文件是按二进制的编码方式来存放文件的。比如，数 5678 的存储形式为 00010110 00101110，只占两个字节。二进制文件虽然也可在屏幕上显示，但其内容无法读懂。C 在处理这些文件时，并不区分类型，都看成是字符流，按字节进行处理。输入/输出字符流的开始和结束只受程序控制而不受物理符号（如回车符）的控制。因此也把这种文件称为“流式文件”。

4.6.5.2　文件指针

C 语言中用一个指针变量指向一个文件，这时这个指针就称为文件指针。通过文件指针就可对它所指的文件进行各种操作。定义文件指针的一般形式为：FILE* 指针变量标识符；其中 FILE 应为大写，它实际上是由系统定义的一个结构体，该结构体中含有文件名、文件状态和文件当前位置等信息。在编写源程序时不必关心 FILE 结构体的细节。例如，FILE *fp；表示 fp 是指向 FILE 结构的指针变量，通过 fp 即可找到存放某个文件信息的结构变量，然后按结构变量提供的信息找到该文件，实施对文件的操作。习惯上也笼统地把 fp 称为指向一个文

件的指针。文件的打开与关闭在进行读、写操作之前要先打开，使用完毕要关闭。所谓打开文件，实际上是建立文件的各种有关信息，并使文件指针指向该文件，以便进行其他操作。关闭文件则断开指针与文件之间的联系，也就禁止再对该文件进行操作。

4.6.5.3 文件打开与关闭

在C语言中，文件操作都是由库函数来完成的。下面将介绍主要的文件操作函数。

文件的打开操作表示将给用户指定的文件在内存分配一个FILE结构区，并将该结构的指针返回给用户程序，这样用户程序就可用此FILE指针来实现对指定文件的存取操作了。当使用打开函数时，必须给出文件名、文件操作方式（读、写或读写），如果该文件名不存在，就意味着建立（只对写文件而言，对读文件则出错），并将文件指针指向文件开头。若已有一个同名文件存在，则删除该文件，若无同名文件，则建立该文件，并将文件指针指向文件开头。当用fopen()成功地打开一个文件时，该函数将返回一个FILE指针，如果文件打开失败，将返回一个NULL指针。

1. 文件打开函数

其调用的一般形式为：fopen(char *filename,char *type)，即fopen(文件名，使用文件方式)。其中，“文件名指针”必须是被说明为FILE 类型的指针变量；“文件名”是被打开文件的文件名；“使用文件方式”是指文件的类型和操作要求。“文件名”是字符串常量或字符串数组。

例如，FILE *fp；fp=("filea","r"); 其意义是在当前目录下打开文件filea，只允许进行“读”操作，并使fp指向该文件。

又如，FILE *fphzk；fphzk=("c:\\hzk',"rb")；其意义是打开C驱动器磁盘的根目录下的文件hzk，这是一个二进制文件，只允许按二进制方式进行读操作。两个反斜线“\\ ”中的第一个表示转义字符，第二个表示根目录。使用文件的方式共有12种，表4-19给出了它们的符号和意义。

对于文件使用方式有以下几点说明：

（1）文件使用方式由r、w、a、t、b和+6个字符拼成，各字符的含义如下：

r（read）：读。

w（write）：写。

a（append）：追加。

t（text）：文本文件，可省略不写。

b（banary）：二进制文件。

+：读和写。

（2）凡用“r”打开一个文件时，该文件必须已经存在，且只能从该文件读出。

（3）用“w”打开的文件只能向该文件写入。若打开的文件不存在，则以指定的文件名建立该文件，若打开的文件已经存在，则将该文件删去，重建一个新文件。

（4）若要向一个已存在的文件追加新的信息，只能用“a”方式打开文件。但此时该文件必须是存在的，否则将会出错。

（5）在打开一个文件时，如果出错，fopen将返回一个空指针值NULL。在程序中可以用这一信息来判别是否完成打开文件的工作，并作相应的处理。

（6）把一个文本文件读入内存时，要将ASCII码转换成二进制码，而把文件以文本方式写入磁盘时，也要把二进制码转换成ASCII码，因此文本文件的读写要花费较多的转换时间。对二进制文件的读写不存在这种转换。

（7）标准输入文件（键盘），标准输出文件（显示器），标准出错输出（出错信息）是由系统打开的，可直接使用。

表 4-19　文件使用方式

文件使用方式	意　义
“rt”	只读打开一个文本文件，只允许读数据
“wt”	只写打开或建立一个文本文件，只允许写数据
“at”	追加打开一个文本文件，并在文件末尾写数据
“rb”	只读打开一个二进制文件，只允许读数据
“wb”	只写打开或建立一个二进制文件，只允许写数据
“ab”	追加打开一个二进制文件，并在文件末尾写数据
“rt+”	读写打开一个文本文件，允许读和写
“wt+”	读写打开或建立一个文本文件，允许读写
“at+”	读写打开一个文本文件，允许读，或在文件末追加数据
“rb+”	读写打开一个二进制文件，允许读和写
“wb+”	读写打开或建立一个二进制文件，允许读和写
“ab+”	读写打开一个二进制文件，允许读，或在文件末追加数据

2．文件关闭函数

文件操作完成后，必须要用 fclose()函数进行关闭，这是因为对打开的文件进行写入时，若文件缓冲区的空间未被写入的内容填满，这些内容因为不会写到打开的文件中去而丢失。只有对打开的文件进行关闭操作时，停留在文件缓冲区的内容才能写到该文件中去，从而使文件完整。再者一旦关闭了文件，该文件对应的 FILE 结构将被释放，从而使关闭的文件得到保护，因为这时对该文件的存取操作将不会进行。文件的关闭也意味着释放了该文件的缓冲区。

文件关闭函数调用的一般形式为：int fclose(FILE *stream); 它表示该函数将关闭 FILE 指针对应的文件，并返回一个整数值。若成功地关闭了文件，则返回一个 0 值；否则返回一个非 0 值。当打开多个文件进行操作，而又要同时关闭时，可采用 fcloseall()函数，它将关闭所有在程序中打开的文件。同时将各文件缓冲区未装满的内容写到相应的文件中去，接着释放这些缓冲区，并返回关闭文件的数目。

4.6.5.4　文件的顺序读写

在 C 语言中提供了多种文件顺序读写函数：下面分别予以介绍。使用以上函数都要求包含头文件 stdio.h。

1．读字符函数 fgetc

fgetc 函数的功能是从指定的文件中读一个字符，函数调用的形式为：字符变量=fgetc(文件指针)；例如，ch=fgetc(fp);的意义是从打开的文件 fp 中读取一个字符并送入 ch 中。

注意：对于 fgetc 函数的使用有以下几点说明：

（1）在 fgetc 函数调用中，读取的文件必须是以读或读写方式打开的。

（2）读取字符的结果也可以不向字符变量赋值，如 fgetc(fp);但是读出的字符不能保存。

（3）在文件内部有一个位置指针，用来指向文件的当前读写字节。在文件打开时，该指针总是指向文件的第一个字节。使用 fgetc 函数后，该位置指针将向后移动一个字节。因此可连续多次使用 fgetc 函数，读取多个字符。应注意文件指针和文件内部的位置指针不是一回事。

文件指针是指向整个文件的，须在程序中定义说明，只要不重新赋值，文件指针的值是不变的。文件内部的位置指针用以指示文件内部的当前读写位置，每读写一次，该指针均向后移动，它不需在程序中定义说明，而是由系统自动设置的。

2．写字符函数 fputc

fputc 函数的功能是把一个字符写入指定的文件中，函数调用的形式为：fputc(字符量,文件指针)；其中，待写入的字符量可以是字符常量或变量，如 fputc('a',fp);的意义是把字符 a 写入 fp 所指向的文件中。

注意：对于 fputc 函数的使用也要说明几点：

（1）被写入的文件可以用写、读写、追加方式打开，用写或读写方式打开一个已存在的文件时将清除原有的文件内容，写入字符从文件首开始。如需保留原有文件内容，希望写入的字符以文件末尾开始存放，必须以追加方式打开文件。被写入的文件若不存在，则创建该文件。

（2）每写入一个字符，文件内部位置指针向后移动一个字节。

（3）fputc 函数有一个返回值，如写入成功则返回写入的字符；否则返回一个 EOF。可用此来判断写入是否成功。

3．字符串读写函数 fgets

读字符串函数 fgets 函数的功能是从指定的文件中读一个字符串到字符数组中，函数调用的形式为：fgets(字符数组名,n,文件指针)；其中的 n 是一个正整数。表示从文件中读出的字符串不超过 n-1 个字符。在读入的最后一个字符后加上串结束标志'\0'。例如，fgets(str,n,fp);的意义是从 fp 所指的文件中读出 n-1 个字符送入字符数组 str 中。

注意：对 fgets 函数有两点说明：

（1）在读出 n-1 个字符之前，如遇到了换行符或 EOF，则读出结束。

（2）fgets 函数也有返回值，其返回值是字符数组的首地址。

4．写字符串函数 fputs

fputs 函数的功能是向指定的文件写入一个字符串，其调用形式为：fputs(字符串,文件指针)其中，字符串可以是字符串常量，也可以是字符数组名或指针变量，例如，fputs("abcd",fp);的意义是把字符串“abcd”写入 fp 所指的文件之中。

5．数据块读写函数 fread 和 fwrite

C 语言还提供了用于整块数据的读写函数。可用来读写一组数据，如一个数组元素、一个结构变量的值等。

读数据块函数调用的一般形式为：fread(buffer,size,count,fp);

写数据块函数调用的一般形式为：fwrite(buffer,size,count,fp);

其中，buffer 是一个指针，在 fread 函数中，它表示存放输入数据的首地址；在 fwrite 函数中，它表示存放输出数据的首地址；size 表示数据块的字节数；count 表示要读写的数据块块数；fp 表示文件指针。

例如，fread(fa,4,5,fp);的意义是从 fp 所指的文件中，每次读 4 个字节（一个实数）送入实数数组 fa 中，连续读 5 次，即读 5 个实数到 fa 中。

例 4-26　从键盘输入两个学生数据，写入一个文件中，再读出这两个学生的数据显示在屏幕上。

```
#include<stdio.h>
struct stu
```

```
{   char name[10];
    int num;
    int age;
    char addr[15];
}boya[2],boyb[2],*pp,*qq;
main()
{   FILE *fp;
    char ch;
    int i;
    pp=boya;
    qq=boyb;
    if((fp=fopen("stu_list","wb+"))==NULL)
    {   printf("Cannot open file strike any key exit!");
        getch();
        exit(1);
    }
    printf("\ninput data\n");
    for(i=0;i<2;i++,pp++)
    scanf("%s%d%d%s",pp->name,&pp->num,&pp->age,pp->addr);
    pp=boya;
    fwrite(pp,sizeof(struct stu),2,fp);
    rewind(fp);
    fread(qq,sizeof(struct stu),2,fp);
    printf("\n\nname\tnumber age addr\n");
    for(i=0;i<2;i++,qq++)
    printf("%s\t%5d%7d%s\n",qq->name,qq->num,qq->age,qq->addr);
    fclose(fp);
}
```

本例程序定义了一个结构 stu，说明了两个结构数组 boya 和 boyb 以及两个结构指针变量 pp 和 qq。pp 指向 boya，qq 指向 boyb。程序第 16 行以读写方式打开二进制文件 stu_list，输入二个学生数据之后，写入该文件中，然后把文件内部位置指针移到文件首，读出两块学生数据后，在屏幕上显示。

6. 格式化读写函数 fscanf 和 fprintf

fscanf 函数、fprintf 函数与前面使用的 scanf 和 printf 函数的功能相似，都是格式化读写函数。两者的区别在于 fscanf 函数和 fprintf 函数的读写对象不是键盘和显示器，而是磁盘文件。这两个函数的调用格式为：

fscanf(文件指针,格式字符串,输入表列);

fprintf(文件指针,格式字符串,输出表列);

例 4-27　读写函数示例。

```
#include<stdio.h>
struct stu
{   char name[10];
    int num;
    int age;
    char addr[15];
```

```
}boya[2],boyb[2],*pp,*qq;
main()
{   FILE *fp;
    char ch;
    int i;
    pp=boya;
    qq=boyb;
    if((fp=fopen("stu_list","wb+"))==NULL)
    {   printf("Cannot open file strike any key exit!");
        getch();
        exit(1);
    }
    printf("\ninput data\n");
    for(i=0;i<2;i++,pp++)
    scanf("%s%d%d%s",pp->name,&pp->num,&pp->age,pp->addr);
    pp=boya;
    for(i=0;i<2;i++,pp++)
    fprintf(fp,"%s %d %d %s\n",pp->name,pp->num,pp->age,pp->addr);
    rewind(fp);
    for(i=0;i<2;i++,qq++)
    fscanf(fp,"%s %d %d %s\n",qq->name,&qq->num,&qq->age,qq->addr);
    printf("\n\nname\tnumber age addr\n");
    qq=boyb;
    for(i=0;i<2;i++,qq++)
    printf("%s\t%5d %7d %s\n",qq->name,qq->num, qq->age,
    qq->addr);
    fclose(fp);
}
```

本程序中 fscanf 和 fprintf 函数每次只能读写一个结构数组元素，因此采用了循环语句来读写全部数组元素。还要注意指针变量 pp、qq 由于循环改变了它们的值，因此在程序的 25 和 32 行分别对它们重新赋予了数组的首地址。

4.6.5.5　文件的随机读写

前面介绍的对文件的读写方式都是顺序读写，即读写文件只能从头开始，顺序读写各个数据。但在实际问题中常要求只读写文件中某一指定的部分。

为了解决这个问题，可移动文件内部的位置指针到需要读写的位置，再进行读写，这种读写称为随机读写。实现随机读写的关键是要按要求移动位置指针，这称为文件的定位。

1．文件定位函数

文件定位移动文件内部位置指针的函数主要有两个，即 rewind 函数和 fseek 函数。

rewind 函数的调用形式为：rewind(文件指针);

它的功能是把文件内部的位置指针移到文件首。

fseek 函数调用形式为：fseek(文件指针,位移量,起始点);

它的功能是用来移动文件内部位置指针。“位移量”表示移动的字节数，要求位移量是 long 型数据，以便在文件长度大于 64KB 时不会出错。当用常量表示位移量时，要求加后缀“L”。“起始点”表示从何处开始计算位移量，规定的起始点有 3 种，即文件首、当前位置和文件尾。

其表示方法见表 4-20。

表 4-20　起始点的表示符号

起始点	表示符号	数字表示
文件首	SEEK-SET	0
当前位置	SEEK-CUR	1
文件尾	SEEK-END	2

例如，fseek(fp,100L,0);的意义是把位置指针移到离文件首 100 个字节处。还要说明的是 fseek 函数一般用于二进制文件。在文本文件中由于要进行转换，故往往计算的位置会出现错误。文件的随机读写在移动位置指针之后，即可用前面介绍的任一种读写函数进行读写。由于一般是读写一个数据块，因此常用 fread 和 fwrite 函数。下面用例题来说明文件的随机读写。

例 4-28　在学生文件 stu list 中读出第二个学生的数据。

```
#include<stdio.h>
struct stu
{   char name[10];
    int num;
    int age;
    char addr[15];
}boy,*qq;
main()
{   FILE *fp;
    char ch;
    int i=1;
    qq=&boy;
    if((fp=fopen("stu_list","rb"))==NULL)
    {   printf("Cannot open file strike any key exit!");
        getch();
        exit(1);
    }
    rewind(fp);
    fseek(fp,i*sizeof(struct stu),0);
    fread(qq,sizeof(struct stu),1,fp);
    printf("\n\nname\tnumber age addr\n");
    printf("%s\t%5d %7d %s\n",qq->name,qq->num,qq->age,
    qq->addr);
}
```

2. 文件检测函数

C 语言中常用的文件检测函数有以下几个：

（1）文件结束检测函数 feof 的调用格式：feof(文件指针);

功能：判断文件是否处于文件结束位置，如文件结束，则返回值为 1；否则返回值为 0。

（2）读写文件出错检测函数 ferror 的调用格式：ferror(文件指针);

功能：检查文件在用各种输入/输出函数进行读写时是否出错。 如 ferror 返回值为 0 表示未出错；否则表示有错。

（3）文件出错标志和文件结束标志置 0 函数 clearerr 的调用格式：clearerr(文件指针);

功能：本函数用于清除出错标志和文件结束标志，使它们为 0 值。

4.6.6 对数据库记录的操作

这里所说的数据库是一个基于命令行的简单数据库系统。它的基本功能有建立数据库表；对数据库中的单表进行数据导入；对数据库中的单表进行插入；对数据库中的单表进行查询；对数据库中的单表进行更新；对数据库中的单表进行删除。另外，当数据量比较大时大家还可以建立索引，并通过索引加快查询速度。

4.6.6.1 建立数据库表

例 4-29 应用数组类型，建立 30 名新生的学籍，并保存在文件名为 jsj091.dat 的文件中。假设用以下结构体类型表示某学生的学号、姓名、性别、出生日期和家庭住址等信息。

```
# include<conio.h>
# include<stdio.h>
# include<string.h>
#define N 1
struct date
{ int    day;
     int    month;
     int    year
};
struct student
{ int num;
     char name [10 ];
     char sex [2 ];
     struct date bdate;
     char addr [15 ];
};
struct    student    stu [30 ];
void save()
{    FILE * fp;
     int i ;
     if ((fp =fopen("jsj901.dat", "wb"))= =NULL);
     { printf("Cannot open file! Press any key return\n");
          return;
     }
     for(i = 0 ; i<N ; i+ +)
          fwrite( & stu [i ],sizeof(struct student),1,fp);
     fclose(fp);
     }
main()
{    int    i;
     FILE      *fp;
     clrscr();
     printf("\n—请输入学生信息—\n: ");
     for(i = 0 ; i<N ;I + +)
```

```
        {   printf ("\n 学号：");scanf("%d", ＆stu [i ].num);
            printf ("\n 姓名：");scanf("%s", stu [i ].name);
            printf ("\n 性别：");scanf("%s", stu [i ].sex);
            printf ("\n 出生日期：年/ 月 / 日");
            scanf(" / %d / %d / %d", ＆stu[i].bdate.year
                ＆stu[i]bdate.month  ＆stu[i].bdate.day);
            printf(\n 家庭住址：");
            scanf("%s",stu[i].addr);
        }
            save();
    }
```

4.6.6.2 随机修改某条记录

修改某条记录的操作时经常遇到的，若随机修改某条记录，首先需要将指针定位于该记录，然后才能对其修改，所以必须使用指针定位函数 fseek。但注意 fseek 函数一般只用于二进制文件，因为文本文件要发生字符转换，计算位置时往往会发生混乱。

例 4-30 若磁盘文件 jsj091.dat 中有 30 名学生的数据，根据输入的学号，修改对应学生的姓名，并将修改结果保存。

程序如下：

```
# include<conio.h>
# include<stdio.h>
# include<string.h>
struct date
{   int day;
    int month;
    int year
};
struct student
{   int num
    char name [10 ];
    char sex [2 ];
    struct date bdate;
    char addr [15 ];
};
struct student sstu;
main()
{   FILE * fp;
    int i ;
    clrscr();
    if ((fp = fopen("jsj901.dat","rb"))= =NULL)
    { printf("Cannot open file!Press any key return!\n");
        getch();
        return;
    }
    printf("\n 请输入学号：");
    scanf("%d", ＆i);
    fseek(fp,(i-1)*sizeof(struct student),0);
```

```
        fread(&sstu.sizeof(struct student),1,fp);
        printf("\n 请输入姓名：");
        scanf("%s",sstu.name);
        fseek(fp ,(i-1)*sizeof(struct student),0);
        fwrite(&sstu.sizeof(struct student),1,fp);
        fclose(fp);
    }
```

说明：数据文件 jsj091.dat 要求是按照学号顺序存储的；否则需要通过循环读出记录与修改学号比较。

4.6.6.3 添加某条记录

向数据文件添加某条记录的情况主要有两种：一是在文件中间添加记录；二是在文件尾部追加记录。后一种情况在打开文件时，文件使用方式选择“a”，即可实现追加。对于前一种情况，处理比较麻烦一些，当然有很多方法，在此提供一种解决思路。具体如下：若在文件 jsj091.dat 中第 6 条记录后插入一条新记录，首先建立一个临时文件，如 temp.dat，先将源文件中前 6 条记录复制到临时文件中，然后将插入记录写入临时文件里，最后将源文件中的第 6 条记录以后所有记录复制到临时文件中。临时文件中的数据是按照新的顺序排放的，将临时文件复制到源文件中，覆盖原有数据（或者用删除和改名语句实现），实现添加记录的操作。

4.6.6.4 删除某条记录

删除数据文件中的某条记录也是经常遇到的操作，在此也提供一种方法，即建立“临时文件”法。具体如下：将源文件中记录复制到临时文件的过程中，需要删除的记录跳过去，不写入临时文件中，最后将源文件删除，临时文件更名为源文件即可。

注意：许多初学者往往认为只要将程序中的变量修改，磁盘中的数据就会随之改变，这种观念是不对的，必须将变化的数据送回到磁盘中保存才会实现修改，但也要注意文件指针的位置。