绘图环境相关函数

cleardevice

功能:

这个函数用于清除画面内容。具体的，是用当前背景色清空画面。

声明:

```
void cleardevice(
    PIMAGE pimg = NULL
);
```

参数:

pimg

指定要清除的PIMAGE，可选参数。如果不填本参数，则清除屏幕。

返回值:

（无）

示例:

```
#include "graphics.h"
int main()
{
    initgraph(640, 480);
    circle(200, 200, 100);
    getch();
    cleardevice();
    getch();
    closegraph();
    return 0;
}
```

clearviewport

功能:

这个函数用于清空视图。相当于对视图区进行cleardevice。

声明:

```
void clearviewport(
    PIMAGE pimg = NULL
);
```

参数:

pimg

见setviewport

返回值:

（无）

示例:

（无）

closegraph

功能:

这个函数用于关闭图形环境。

声明:

```
void closegraph();
```

参数:

（无）

返回值:

（无）

示例:

```
#include "graphics.h"
int main()
{
    initgraph(640, 480);
    circle(200, 200, 100);
    getch();
    cleardevice();
```

```
    getch();
    closegraph();
    return 0;
}
```

gettarget

功能:

这个函数用于获取当前绘图对象。

声明:

```
PIMAGE gettarget();
```

参数: （无）

返回值: PIMAGE对象

示例: （无）

getviewport

功能:

这个函数用于获取当前视图信息。

声明:

```
void getviewport(
    int *pleft,
    int *ptop,
    int *pright,
    int *pbottom,
    int *pclip = NULL,
    PIMAGE pimg = NULL
);
```

参数:

pleft

返回当前视图的左部 x 坐标。

ptop

返回当前视图的上部 y 坐标。
pright
返回当前视图的右部 x 坐标。
pbottom
返回当前视图的下部 y 坐标。
pclip
返回当前视图的裁剪标志。
pimg
详见setviewport的说明
返回值:
（无）
示例:
参见setviewport
initgraph
功能:
这个函数用于初始化绘图环境。
声明:

```
void initgraph(
    int Width,
    int Height,
    int Flag = INIT_DEFAULT
);
void initgraph(
    int* gdriver,
    int* gmode,
    char* path
); // 兼容 Borland C++ 3.1 的重载，不建议使用。
```

参数:

Width
绘图环境的宽度。如果为-1，则使用屏幕的宽度
Height
绘图环境的高度。如果为-1，则使用屏幕的高度
Style
请留空，为保留参数
返回值:
（无）
示例:

```
#include "graphics.h"
int main()
{
    initgraph(640, 480);
    circle(200, 200, 100);
    getch();
    cleardevice();
    getch();
    closegraph();
    return 0;
}
```

is_run
功能:
这个函数用于判断窗口是否还存在。
声明:

```
void is_run();
```

参数:
（无）
返回值:

0 表示窗口被关闭了
1 表示窗口没有被关闭，程序还在运行
示例:
详见入门示例里的动画部分
setactivepage
功能:
这个函数用于设置当前绘图页。
声明:
void setactivepage(int page);
参数:
page
绘图页，范围从，范围从0-3，越界会导致程序错误。默认值为0
返回值:
（无）
示例:
（无）
setcaption
功能:
这个函数用设置窗口标题。
声明:
void setcaption(LPCSTR caption);
void setcaption(LPCWSTR caption);
参数:
caption
指定要设置的标题字符串，可用宽字符或MBCS字符串，需要使用gbk编码。
返回值:
（无）
示例:

（无）
setinitmode
功能:
这个函数用于设置初始化图形的选项和模式。
声明:
void setinitmode(int mode, int x = CW_USEDEFAULT, int y = CW_USEDEFAULT);
参数:
mode
初始化模式，是二进制组合的值。如果为INIT_DEFAULT 表示使用默认值。
可以使用的值的组合:
INIT_NOBORDER 为无边框窗口
INIT_CHILD 为子窗口（需要使用attachHWND指定要依附的父窗口，此函数不另说明）
INIT_TOPMOST 使窗口总在最前
INIT_WITHLOGO 使initgraph的时候显示开场动画logo
INIT_NOFORCEEXIT 使关闭窗口的时候不强制退出程序，但窗口会消失，需要配合is_run函数
x, y
初始化时窗口左上角在屏幕的坐标，默认为系统分配。
返回值:
（无）
说明:
本函数只能在initgraph前调用。
示例:
（无）
setrendermode
功能:
这个函数用于设置更新窗口的模式，模式有两种，自动更新和手动更新
声明:
void setrendermode(rendermode_e mode);

参数:
mode
值可能为RENDER_AUTO或者RENDER_MANUAL，前者自动（默认值），后者手动
自动模式用于简单绘图，手动模式用于制作动画或者游戏
所谓手动模式，即需要调用delay_fps/delay_ms等带有等待特性的函数时才会更新窗口
返回值:
（无）
示例:
（无）
settarget
功能:
这个函数用于设置当前绘图对象。
声明:
void settarget(PIMAGE pbuf = NULL);
参数:
pbuf
绘图对象，为PIMAGE类型，你要绘画到的PIMAGE，如果不填，则还原为窗口
返回值:
（无）
示例:
（无）
setvisualpage
功能:
这个函数用于设置当前显示页，显示页是输出到窗口的页。
声明:
void setvisualpage(int page);
参数:
page

绘图页，范围从，范围从0-3，越界会导致程序错误。默认值为0
返回值:
（无）
示例:
（无）
window_getviewport
功能:
这个函数用于获取当前窗口可见区域。
声明:
```
void window_getviewport(
    int *pleft,
    int *ptop,
    int *pright,
    int *pbottom,
);
```
参数:
pleft
返回当前视图的左部 x 坐标。
ptop
返回当前视图的上部 y 坐标。
pright
返回当前视图的右部 x 坐标。
pbottom
返回当前视图的下部 y 坐标。
返回值:
（无）
示例:
（无）

window_setviewport
功能:
这个函数用于设置当前窗口可见区域。
声明:
```
void window_setviewport(
    int left,
    int top,
    int right,
    int bottom
);
```
参数:
left
可见区域的左部 x 坐标。
top
可见区域的上部 y 坐标。(left, top) 将成为新的原点。
right
可见区域的右部 x 坐标。
bottom
可见区域的下部 y 坐标。(right-1, bottom-1) 是视图的右下角坐标。
返回值:
(无)
示例:
```
#include "graphics.h"
int main()
{
    initgraph(640, 480);
    window_setviewport(100, 100, 400, 400);
    rectangle(0, 0, 200, 200);
```

```
    getch();
    closegraph();
    return 0;
}
```

颜色表示及相关函数

颜色表示

设置绘图色有以下几种办法:

1. 用 16 进制的颜色表示，形式为:

0xrrggbb (rr=红，gg=绿，bb=蓝)

2. 用预定义颜色，如下:

常量值 值 颜色值 常量值 值 颜色 BLACK值 0值 黑值 DARKGRAY值 0x545454值 深灰 BLUE值 0x0000A8值 蓝值 LIGHTBLUE值 0x5454FC值 亮蓝 GREEN值 0x00A800值 绿值 LIGHTGREEN值 0x54FC54值 亮绿 CYAN值 0x00A8A8值 青值 LIGHTCYAN值 0x54FCFC值 亮青 RED值 0xA80000值 值 LIGHTRED值 0xFC5454值 亮红 MAGENTA值 0xA800A8值 紫值 LIGHTMAGENTA值 0xFC54FC值 亮紫 BROWN值 0xA8A800值 棕值 YELLOW值 0xFCFC54值 黄 LIGHTGRAY值 0xA8A8A8值 浅灰值 WHITE值 0xFCFCFC值 白

3. 用 EGERGB 宏合成颜色。详见 EGERGB。

4. 用 hsl2rgb、hsv2rgb 转换其他色彩模型到 RGB 颜色。详见 hsl2rgb、hsv2rgb。

示例:

以下是部分设置前景色的方法:

```
setcolor(0xff0000);
setcolor(BLUE);
setcolor(EGERGB(0, 0, 255));
setcolor(hsl2rgb(240, 1, 0.5));
```

getbkcolor

功能:

这个函数用于获取当前绘图背景色。

声明:

```
color_t getbkcolor(
    PIMAGE pimg = NULL
```

);
参数:
（无）
返回值:
返回当前绘图背景色。
示例:
（无）
EGEGET_B
功能:
EGEGET_B 宏用于返回指定颜色中的蓝色值。
声明:
color_t EGEGET_B(color_t rgb);
参数:
rgb
指定的颜色。
返回值:
指定颜色中的蓝色值，值的范围 0~255。
示例:
（无）
getcolor
功能:
这个函数用于获取当前绘图前景色。
声明:
color_t getcolor(
    PIMAGE pimg = NULL
);
参数:
（无）

返回值:
返回当前绘图前景色。
示例:
（无）
getfillcolor
功能:
这个函数用于获取当前绘图填充色。
声明:
```
color_t getfillcolor(
    PIMAGE pimg = NULL
);
```
参数:
（无）
返回值:
返回当前绘图填充色。
示例:
（无）
EGEGET_G
功能:
EGEGET_G 宏用于返回指定颜色中的绿色值。
声明:
```
color_t EGEGET_G(color_t rgb);
```
参数:
rgb
指定的颜色。
返回值:
指定颜色中的绿色值，值的范围 0~255。
示例:

（无）
EGEGET_R
功能:
EGEGET_R 宏用于返回指定颜色中的红色值。
声明:
color_t EGEGET_R(color_t rgb);
参数:
rgb
指定的颜色。
返回值:
指定颜色中的红色值，值的范围 0~255。
示例:
（无）
hsl2rgb
功能:
该函数用于转换 HSL 颜色为 RGB 颜色。
声明:
```
color_t hsl2rgb(
    float H,
    float S,
    float L
);
```
参数:
H
原 HSL 颜色模型的 Hue(色相) 分量，0 ≤ H < 360。
S
原 HSL 颜色模型的 Saturation(饱和度) 分量，0 ≤ S ≤ 1。
L

原 HSL 颜色模型的 Lightness(亮度) 分量，0 ⩽ L ⩽ 1。
返回值:
对应的 RGB 颜色。
说明:
HSL 又称 HLS。
HSL 的颜色模型如图所示:
H 是英文 Hue 的首字母，表示色相，即组成可见光谱的单色。红色在 0 度，绿色在 120 度，蓝色在 240 度，以此方向过渡。
S 是英文 Saturation 的首字母，表示饱和度，等于 0 时为灰色。在最大饱和度 1 时，具有最纯的色光。
L 是英文 Lightness 的首字母，表示亮度，等于 0 时为黑色，等于 0.5 时是色彩最鲜明的状态，等于 1 时为白色。
示例:
请参见示例程序19中的“彩虹”。
hsv2rgb
功能:
该函数用于转换 HSV 颜色为 RGB 颜色。
声明:

```
color_t hsv2rgb(
    float H,
    float S,
    float V
);
```

参数:
H
原 HSL 颜色模型的 Hue(色相) 分量，0 ⩽ H < 360。
S
原 HSL 颜色模型的 Saturation(饱和度) 分量，0 ⩽ S ⩽ 1。
V
原 HSL 颜色模型的 Value(明度) 分量，0 ⩽ L ⩽ 1。
返回值:

对应的 RGB 颜色。
说明:
HSV 又称 HSB。
HSV 的颜色模型如图所示:
H 是英文 Hue 的首字母，表示色相，即组成可见光谱的单色。红色在 0 度，绿色在 120 度，蓝色在 240 度，以此方向过渡。
S 是英文 Saturation 的首字母，表示饱和度，等于 0 时为灰色。在最大饱和度 1 时，每一色相具有最纯的色光。
V 是英文 Value 的首字母，表示明度，等于 0 时为黑色，在最大明度 1 时，是色彩最鲜明的状态。
示例:
HSV 颜色模型类似于 HSL，示例程序19中的“彩虹”是 HSL 模型的操作范例，可以参考。
EGERGB
功能:
EGERGB 宏用于通过红、绿、蓝颜色分量合成颜色。
声明:

```
color_t EGERGB(
    BYTE byRed,     // 颜色的红色部分
    BYTE byGreen,   // 颜色的绿色部分
    BYTE byBlue     // 颜色的蓝色部分
);
```

参数:
byRed
颜色的红色部分，取值范围: 0~255。
byGreen
颜色的绿色部分，取值范围: 0~255。
byBlue
颜色的蓝色部分，取值范围: 0~255。
返回值:
返回合成的颜色。
示例:

（无）
rgb2gray
功能:
该函数用于返回与指定颜色对应的灰度值颜色。
声明:
```
color_t rgb2gray(
    color_t rgb
);
```
参数:
rgb
原 RGB 颜色。
返回值:
对应的灰度颜色。
示例:
（无）
rgb2hsl
功能:
该函数用于转换 RGB 颜色为 HSL 颜色。
声明:
```
void rgb2hsl(
    color_t rgb,
    float *H,
    float *S,
    float *L
);
```
参数:
rgb
原 RGB 颜色。

H
原 HSL 颜色模型的 Hue(色相) 分量，0 ⩽ H < 360。
S
原 HSL 颜色模型的 Saturation(饱和度) 分量，0 ⩽ S ⩽ 1。
L
原 HSL 颜色模型的 Lightness(亮度) 分量，0 ⩽ L ⩽ 1。
返回值:
（无）
说明:
HSL 详见 hsl2rgb。
示例:
（无）
rgb2hsv
功能:
该函数用于转换 RGB 颜色为 HSV 颜色。
声明:
```
void rgb2hsv(
    color_t rgb,
    float *H,
    float *S,
    float *V
);
```
参数:
rgb
原 RGB 颜色。
H
原 HSL 颜色模型的 Hue(色相) 分量，0 ⩽ H < 360。
S

原 HSL 颜色模型的 Saturation(饱和度) 分量，0 ⩽ S ⩽ 1。
V
原 HSL 颜色模型的 Value(明度) 分量，0 ⩽ L ⩽ 1。
返回值:
（无）
说明:
HSV 详见 hsv2rgb。
示例:
（无）
setbkcolor
功能:
这个函数用于设置当前背景色。并且会把当前图片上是原背景色的像素，转变为新的背景色。
声明:
```
void setbkcolor(
    color_t color,
    PIMAGE pimg = NULL
);
```
参数:
color
指定要设置的背景颜色。注意，该设置会同时影响文字背景色。
返回值:
（无）
示例:
（无）
setbkcolor_f
功能:
这个函数用于设置当前背景色。即仅设置cleardevice时所使用的颜色，不立即生效，需要等cleardevice调用。
声明:

```
void setbkcolor(
    color_t color,
    PIMAGE pimg = NULL
);
```

参数:

color

指定要设置的背景颜色。注意，该设置会同时影响文字背景色。

返回值:

（无）

示例:

（无）

setbkmode

功能:

这个函数用于设置输出文字时的背景模式。

声明:

```
void setbkmode(
    int iBkMode,
    PIMAGE pimg = NULL
);
```

参数:

iBkMode

指定输出文字时的背景模式，可以是以下值:

值 描述 OPAQUE值 背景用当前背景色填充（默认）。 TRANSPARENT 背景是透明的。

返回值:

（无）

示例:

（无）

setcolor

功能:
这个函数用于设置绘图前景色。
声明:
```
void setcolor(
    color_t color,
    PIMAGE pimg = NULL
);
```
参数:
color
要设置的前景颜色。
返回值:
(无)
示例:
(无)
setcolor
功能:
这个函数用于设置绘图填充色。
声明:
```
void setfillcolor(
    color_t color,
    PIMAGE pimg = NULL
);
```
参数:
color
要设置的填充颜色。
返回值:
(无)
示例:

（无）
setfontbkcolor
功能:
这个函数用于设置文字背景色。
声明:
```
void setfontbkcolor(
    color_t color,
    PIMAGE pimg = NULL
);
```
参数:
color
要设置的文字背景颜色。
返回值:
（无）
示例:
（无）
绘制图形相关函数
arc
功能:
这个函数用于画圆弧。边线颜色由setcolor函数决定
声明:
```
void arc(
    int x,
    int y,
    int stangle,
    int endangle,
    int radius,
    PIMAGE pimg = NULL
```

);
参数:
x
圆弧的圆心 x 坐标。
y
圆弧的圆心 y 坐标。
stangle
圆弧的起始角的角度。
endangle
圆弧的终止角的角度。
radius
圆弧的半径。
返回值:
（无）
示例:
（无）
bar
功能:
这个函数用于画无边框填充矩形。其中，填充颜色由setfillstyle函数决定
声明:
```
void bar(
    int left,
    int top,
    int right,
    int bottom,
    PIMAGE pimg = NULL
);
```
参数:

left
矩形左部 x 坐标。
top
矩形上部 y 坐标。
right
矩形右部 x 坐标（该点取不到，实际右边界为right-1）。
bottom
矩形下部 y 坐标（该点取不到，实际下边界为bottom-1）。
返回值:
（无）
示例:
（无）
bar3d
功能:
这个函数用于画有边框三维填充矩形。其中，填充颜色由setfillstyle函数决定
声明:

```
void bar3d(
    int left,
    int top,
    int right,
    int bottom,
    int depth,
    bool topflag,
    PIMAGE pimg = NULL
);
```

参数:
left
矩形左部 x 坐标。

top
矩形上部 y 坐标。
right
矩形右部 x 坐标（该点取不到，实际右边界为right-1）。
bottom
矩形下部 y 坐标（该点取不到，实际下边界为bottom-1）。
depth
矩形深度。
topflag
为 false 时，将不画矩形的三维顶部。该选项可用来画堆叠的三维矩形。
返回值:
（无）
示例:

```
#include "graphics.h"
int main()
{
    initgraph(600, 400);
    setfillstyle(RED);
    bar3d(100, 100, 150, 150, 20, 1);
    getch();
    return 0;
}
```

示例效果:
circle
功能:
这个函数用于画圆。此圆是空心的，不填充，而边线颜色由setcolor函数决定
声明:
void circle(

```
    int x,
    int y,
    int radius,
    PIMAGE pimg = NULL
);
```
参数:
x
圆的圆心 x 坐标。
y
圆的圆心 y 坐标。
radius
圆的半径。
返回值:
（无）
示例:
（无）
drawbezier
功能:
这个函数用于画bezier曲线。边线颜色由setcolor函数决定
声明:
```
void drawbezier(
    int numpoints,
    const int *polypoints,
    PIMAGE pimg = NULL
);
```
参数:
numpoints
多边形点的个数，需要是被3除余1的数，如果不是，则忽略最后面若干个点。

polypoints
每个点的坐标（依次两个分别为x,y），数组元素个数为 numpoints * 2。
每一条bezier曲线由两个端点和两个控制点组成，相邻两条则共用端点。
返回值:
（无）
示例:
（无）
drawlines
功能:
这个函数用于画多条线段。边线颜色由setcolor函数决定
声明:

```
void drawlines(
    int numliness,
    const int *polypoints,
    PIMAGE pimg = NULL
);
```

参数:
numlines
线段数目。
polypoints
每个点的坐标（依次两个分别为x,y），数组元素个数为 numlines * 4。
每两个点画一线段。
返回值:
（无）
示例:
（无）
drawpoly
功能:

这个函数用于画多边形。边线颜色由setcolor函数决定
声明:
```
void drawpoly(
    int numliness,
    const int *polypoints,
    PIMAGE pimg = NULL
);
```
参数:
numpoints
多边形点的个数。
polypoints
每个点的坐标（依次两个分别为x,y），数组元素个数为 numpoints * 2。
该函数并不会自动连接多边形首尾。如果需要画封闭的多边形，请将最后一个点设置为与第一点相同。
返回值:
（无）
示例:
（无）
ellipse
功能:
这个函数用于画椭圆弧线。边线颜色由setcolor函数决定
声明:
```
void ellipse(
    int x,
    int y,
    int stangle,
    int endangle,
    int xradius,
    int yradius,
```

```
    PIMAGE pimg = NULL
);
```
参数:
x
椭圆弧线的圆心 x 坐标。
y
椭圆弧线的圆心 y 坐标。
stangle
椭圆弧线的起始角的角度。
endangle
椭圆弧线的终止角的角度。
xradius
椭圆弧线的 x 轴半径。
yradius
椭圆弧线的 y 轴半径。
返回值:
（无）
示例:
（无）
fillellipse
功能:
这个函数用于画填充的椭圆。边线颜色由setcolor函数决定，填充颜色由setfillstyle函数决定。
声明:
```
void fillellipse(
    int x,
    int y,
    int xradius,
    int yradius,
```

```
    PIMAGE pimg = NULL
);
```
参数:
x
椭圆的圆心 x 坐标。
y
椭圆的圆心 y 坐标。
xradius
椭圆的 x 轴半径。
yradius
椭圆的 y 轴半径。
返回值:
（无）
示例:
（无）
fillpoly
功能:
这个函数用于画填充的多边形。边线颜色由setcolor函数决定，填充颜色由setfillstyle函数决定
声明:
```
void fillpoly(
    int numpoints,
    const int *polypoints,
    PIMAGE pimg = NULL
);
```
参数:
numpoints
多边形点的个数。
polypoints

每个点的坐标，数组元素个数为 numpoints * 2。
该函数会自动连接多边形首尾。
返回值:
（无）
示例:
（无）
说明:
如果这个多边形发生自相交，那么自交次数为奇数的区域则不填充，偶数次的填充，不自交就是偶数次。不过这样说明相信非常难理解，以下给个例子:

```
#include "graphics.h"
int main()
{
    initgraph(600, 400);
    setfillstyle(RED);
    int pt[] = {
        0,   0,
        100, 0,
        100, 100,
        10,  10,
        90,  10,
        0,   100,
    };
    fillpoly(6, pt);
    getch();
    return 0;
}
```

运行结果:
第二个例子:

```
#include "graphics.h"
int main()
{
    initgraph(600, 400);
    setfillstyle(RED);
    int pt[] = {
        0,   0,
        100, 0,
        100, 100,
        0,   100,
        0,   0,
        100, 0,
        100, 120,
        0,   100,
    };
    fillpoly(8, pt);
    getch();
    return 0;
}
```

运行结果:

floodfill

功能:

这个函数使用setfillstyle设置的填充方式对区域进行填充。填充颜色由setfillstyle函数决定。

声明:

```
void floodfill(
    int x,
    int y,
    int border,
```

```
    PIMAGE pimg = NULL
);
```
参数:
x
填充的起始点 x 坐标。
y
填充的起始点 y 坐标。
border
填充的边界颜色。填充动作在该颜色围成的区域内填充。如果该颜色围成的区域不封闭，那么将使全屏幕都填充上。
返回值:
（无）
示例:
（无）
getfillcolor
功能:
这个函数用于设置当前填充颜色。
声明:
```
COLORREF getfillcolor(
    IMAGE* pimg = NULL
);
```
参数:
（无）
返回值:
当前的填充颜色
示例:
（无）
getheight
功能:

这个函数用于获取图片高度。
声明:
```
int getheight(
    PIMAGE pimg = NULL
);
```
参数:
（无）
返回值:
返回图片高度。
示例:
（无）
getlinestyle
功能:
这个函数用于获取当前线形。
声明:
```
void getlinestyle(
    int  *plinestyle,
    WORD *pupattern = NULL,
    int  *pthickness = NULL,
    PIMAGE pimg = NULL
);
```
参数:
plinestyle
返回当前线型。详见 setlinestyle。
pupattern
返回当前自定义线形数据。
pthickness
返回当前线形宽度。

返回值:
（无）
示例:
（无）
getpixel
功能:
这个函数用于获取像素点的颜色。
声明:

```
COLORREF getpixel(
    int x,
    int y
    PIMAGE pimg = NULL
);
```

参数:
x
要获取颜色的 x 坐标。
y
要获取颜色的 y 坐标。
返回值:
指定点的颜色。
示例:
（无）
其它说明: 另有高速版的getpixel_f函数，参数一样，作用一样，但不进行相对坐标变换和边界检查（如果越界绘图，要么画错地方，要么程序结果莫名其妙，甚至直接崩溃），并且必须在批量绘图模式下才能使用，否则将发生不可预知的结果。
getwidth
功能:
这个函数用于获取图片宽度。
声明:

```
int getwidth(
    PIMAGE pimg = NULL
);
```
参数:
（无）
返回值:
返回图片宽度。
示例:
（无）
getx
功能:
这个函数用于获取当前 x 坐标。
声明:
```
int getx(
    PIMAGE pimg = NULL
);
```
参数:
（无）
返回值:
返回当前 x 坐标。
示例:
（无）
gety
功能:
这个函数用于获取当前 y 坐标。
声明:
```
int gety(
    PIMAGE pimg = NULL
```

);
参数:
（无）
返回值:
返回当前 y 坐标。
示例:
（无）
line
功能:
这个函数用于画线。
声明:
```
void line(
    int x1,
    int y1,
    int x2,
    int y2,
    PIMAGE pimg = NULL
);
```
参数:
x1
线的起始点的 x 坐标。
y1
线的起始点的 y 坐标。
x2
线的终止点的 x 坐标（该点本身画不到）。
y2
线的终止点的 y 坐标（该点本身画不到）。
返回值:

（无）
示例:
（无）
其它说明: 另有高速版的line_f函数，参数一样，作用一样，但不进行相对坐标变换和边界检查（如果越界绘图，要么画错地方，要么程序结果莫名其妙，甚至直接崩溃），并且必须在窗口锁定绘图模式下才能使用，否则将发生不可预知的结果。
linerel
功能:
这个函数用于画线。
声明:
```
void linerel(
    int dx,
    int dy,
    PIMAGE pimg = NULL
);
```
参数:
dx
从“当前点”cx开始画线，沿 x 轴偏移 dx，终点为cx+dx（终点本身画不到）。
dy
从“当前点”cy开始画线，沿 y 轴偏移 dy，终点为cy+dy（终点本身画不到）。
返回值:
（无）
示例:
（无）
其它说明: 另有高速版的linerel_f函数，参数一样，作用一样，但不进行相对坐标变换和边界检查（如果越界绘图，要么画错地方，要么程序结果莫名其妙，甚至直接崩溃），并且必须在窗口锁定绘图模式下才能使用，否则将发生不可预知的结果。
lineto
功能:
这个函数用于画线。

声明:

```
void lineto(
    int x,
    int y,
    PIMAGE pimg = NULL
);
```

参数:

x

从“当前点”开始画线，终点横坐标为 x （终点本身画不到）。

y

从“当前点”开始画线，终点纵坐标为 y （终点本身画不到）。

返回值:

（无）

示例:

（无）

其它说明: 另有高速版的lineto_f函数，参数一样，作用一样，但不进行相对坐标变换和边界检查（如果越界绘图，要么画错地方，要么程序结果莫名其妙，甚至直接崩溃），并且必须在窗口锁定绘图模式下才能使用，否则将发生不可预知的结果。

moverel

功能:

这个函数用于移动当前点。有些绘图操作会从“当前点”开始，这个函数可以设置该点。

声明:

```
void moverel(
    int dx,
    int dy,
    PIMAGE pimg = NULL
);
```

参数:

dx

将当前点沿 x 轴移动 dx。
dy
将当前点沿 y 轴移动 dy。
返回值:
（无）
示例:
（无）
moveto
功能:
这个函数用于移动当前点。有些绘图操作会从“当前点”开始，这个函数可以设置该点。
声明:
```
void moveto(
    int x,
    int y,
    PIMAGE pimg = NULL
);
```
参数:
x
新的当前点 x 坐标。
y
新的当前点 y 坐标。
返回值:
（无）
示例:
（无）
pieslice
功能:
这个函数用于画填充圆扇形。

声明:

```
void pieslice(
    int x,
    int y,
    int stangle,
    int endangle,
    int radius,
    PIMAGE pimg = NULL
);
```

参数:

x

圆扇形的圆心 x 坐标。

y

圆扇形的圆心 y 坐标。

stangle

圆扇形的起始角的角度。

endangle

圆扇形的终止角的角度。

radius

圆扇形的半径。

返回值:

（无）

示例:

（无）

putpixel

功能:

这个函数用于画点。

声明:

```
void putpixel(
    int x,
    int y,
    COLORREF color,
    PIMAGE pimg = NULL
);
```
参数:
x
点的 x 坐标。
y
点的 y 坐标。
color
点的颜色。
返回值:
（无）
示例:
（无）
其它说明: 另有高速版的putpixel_f函数，参数一样，作用一样，但不进行相对坐标变换和边界检查（如果越界绘图，要么画错地方，要么程序结果莫名其妙，甚至直接崩溃），并且必须在窗口锁定绘图模式下才能使用，否则将发生不可预知的结果。
putpixels
功能:
这个函数用于画多个点。
声明:
```
void putpixels(
    int nPoint,
    int* pPoints,
    PIMAGE pimg = NULL
);
```

参数:
nPoint
点的数目。
pPoints
指向点的描述的指针，一个int型的数组，依次每三个int描述一个点：第一个为x坐标，第二个为y坐标，第三个为颜色值。
返回值:
（无）
示例:
（无）
其它说明：另有高速版的putpixels_f函数，参数一样，作用一样，但不进行相对坐标变换和边界检查（如果越界绘图，要么画错地方，要么程序结果莫名其妙，甚至直接崩溃），并且必须在窗口锁定绘图模式下才能使用，否则将发生不可预知的结果。
rectangle
功能:
这个函数用于画空心矩形。
声明:

```
void rectangle(
    int left,
    int top,
    int right,
    int bottom,
    PIMAGE pimg = NULL
);
```

参数:
left
矩形左部 x 坐标。
top
矩形上部 y 坐标。
right

矩形右部 x 坐标。
bottom
矩形下部 y 坐标。
返回值:
（无）
示例:
（无）
sector
功能:
这个函数用于画填充椭圆扇形。
声明:

```
void sector(
    int x,
    int y,
    int stangle,
    int endangle,
    int xradius,
    int yradius,
    PIMAGE pimg = NULL
);
```

参数:
x
椭圆扇形的圆心 x 坐标。
y
椭圆扇形的圆心 y 坐标。
stangle
椭圆扇形的起始角的角度。
endangle

椭圆扇形的终止角的角度。
xradius
椭圆扇形的 x 轴半径。
yradius
椭圆扇形的 y 轴半径。
返回值:
（无）
示例:
（无）
setfillcolor
功能:
这个函数用于设置当前填充颜色。
声明:
```
void setfillcolor(
    COLORREF color,
    PIMAGE pimg = NULL
);
```
参数:
color
填充颜色。
返回值:
（无）
示例:
设置蓝色固实填充:
```
setfillcolor(BLUE);
```
setfillstyle
功能:
这个函数用于设置当前填充类型。该函数的自定义填充部分尚不支持。

声明:

```
void setfillstyle(
    int pattern,
    COLORREF color,
    PIMAGE pimg = NULL
);
```

参数:

pattern

填充类型，可以是以下宏或值:

宏 值 含义 NULL_FILL 1 不填充 SOLID_FILL 2 固实填充

pupattern

color

填充颜色。

指定图案填充时的样式，目前无作用。

返回值:

（无）

示例:

设置蓝色固实填充:

```
setfillstyle(SOLID_FILL, BLUE);
```

setlinestyle

功能:

这个函数用于设置当前线形。

声明:

```
void setlinestyle(
    int linestyle,
    WORD upattern = NULL,
    int thickness = 1,
    PIMAGE pimg = NULL
```

);
参数:
linestyle
线型，可以是以下值:
值 含义 PS_SOLID 线形为实线。 PS_DASH 线形为: －－－－－－－－－－－－ PS_DOT 线形为: ●●●●●●●●●●●●● PS_DASHDOT 线形为: －●－●－●－●－●－● PS_DASHDOTDOT 线形为: －●●－●●－●●－●● PS_NULL 线形为不可见。 PS_USERSTYLE 线形样式是自定义的，依赖于 upattern 参数。
upattern
自定义线形数据。
自定义规则: 该数据为 WORD 类型，共 16 个二进制位，每位为 1 表示画线，为 0 表示空白。从低位到高位表示从起始到终止的方向。
仅当线型为 PS_USERSTYLE 时该参数有效。
thickness
线形宽度。
返回值:
（无）
示例:
设置线形为点划线: setlinestyle(PS_DASHDOT);
设置线形为宽度 3 像素的虚线: setlinestyle(PS_DASH, NULL, 3);
setwritemode
功能:
这个函数用于设置绘图位操作模式。
声明:
```
void setwritemode(
    int mode.
    PIMAGE pimg = NULL
);
```
参数:
mode

二元光栅操作码（即位操作模式），支持全部的 16 种二元光栅操作码，罗列如下:
位操作模式 描述 R2_BLACK 绘制出的像素颜色 = 黑色 R2_COPYPEN 绘制出的像素颜色 = 当前颜色（默认） R2_MASKNOTPEN 绘制出的像素颜色 = 屏幕颜色 AND (NOT 当前颜色) R2_MASKPEN 绘制出的像素颜色 = 屏幕颜色 AND 当前颜色 R2_MASKPENNOT 绘制出的像素颜色 = (NOT 屏幕颜色) AND 当前颜色 R2_MERGENOTPEN 绘制出的像素颜色 = 屏幕颜色 OR (NOT 当前颜色) R2_MERGEPEN 绘制出的像素颜色 = 屏幕颜色 OR 当前颜色 R2_MERGEPENNOT 绘制出的像素颜色 = (NOT 屏幕颜色) OR 当前颜色 R2_NOP 绘制出的像素颜色 = 屏幕颜色 R2_NOT 绘制出的像素颜色 = NOT 屏幕颜色 R2_NOTCOPYPEN 绘制出的像素颜色 = NOT 当前颜色 R2_NOTMASKPEN 绘制出的像素颜色 = NOT (屏幕颜色 AND 当前颜色) R2_NOTMERGEPEN 绘制出的像素颜色 = NOT (屏幕颜色 OR 当前颜色) R2_NOTXORPEN 绘制出的像素颜色 = NOT (屏幕颜色 XOR 当前颜色) R2_WHITE 绘制出的像素颜色 = 白色 R2_XORPEN 绘制出的像素颜色 = 屏幕颜色 XOR 当前颜色
注: 1. AND / OR / NOT / XOR 为布尔位运算。2. "屏幕颜色"指绘制所经过的屏幕像素点的颜色。3. "当前颜色"是指通过 setcolor 设置的用于当前绘制的颜色。
返回值:
（无）
示例:
（无）
文字输出相关函数
getfont
功能:
这个函数用于获取当前字体样式
声明:

```
void getfont(
    LOGFONT *font
    PIMAGE pimg = NULL
);
```

参数:
font
指向 LOGFONT 结构体的指针。
返回值:
（无）

示例:
（无）
LOGFONT 结构体
功能:
这个结构体定义了字体的属性。
声明:

```
struct LOGFONT {
    LONG lfHeight;
    LONG lfWidth
    LONG lfEscapement;
    LONG lfOrientation;
    LONG lfWeight;
    BYTE lfItalic;
    BYTE lfUnderline;
    BYTE lfStrikeOut;
    BYTE lfCharSet;
    BYTE lfOutPrecision;
    BYTE lfClipPrecision;
    BYTE lfQuality;
    BYTE lfPitchAndFamily;
    TCHAR lfFaceName[LF_FACESIZE];
};
```

成员
lfHeight
指定高度（逻辑单位）。
如果为正，表示指定的高度是字体的完整高度；如果为负，表示指定的高度不包含tmInternalLeading的高度。也就是说相同绝对值下，负的比正的会稍高一些；而实际输出的字体高度，正数时精确匹配，负数时，和实际略有偏差，绝对值比实际的略小。
lfWidth

指定字符的平均宽度（逻辑单位）。如果为 0，则比例自适应。
lfEscapement
字符串的书写角度，单位 0.1 度，默认为 0。
lfOrientation
每个字符的书写角度，单位 0.1 度，默认为 0。
lfWeight
字符的笔画粗细，范围 0¯1000，0 表示默认粗细，使用数字或下表中定义的宏均可。
宏 粗细值 FW_DONTCARE 0 FW_THIN 100 FW_EXTRALIGHT 200 FW_ULTRALIGHT 200 FW_LIGHT 300 FW_NORMAL 400 FW_REGULAR 400 FW_MEDIUM 500 FW_SEMIBOLD 600 FW_DEMIBOLD 600 FW_BOLD 700 FW_EXTRABOLD 800 FW_ULTRABOLD 800 FW_HEAVY 900 FW_BLACK 900
lfItalic
指定字体是否是斜体。
lfUnderline
指定字体是否有下划线。
lfStrikeOut
指定字体是否有删除线。
lfCharSet
指定字符集。以下是预定义的值:
ANSI_CHARSET
BALTIC_CHARSET
CHINESEBIG5_CHARSET
DEFAULT_CHARSET
EASTEUROPE_CHARSET
GB2312_CHARSET
GREEK_CHARSET
HANGUL_CHARSET
MAC_CHARSET
OEM_CHARSET
RUSSIAN_CHARSET

SHIFTJIS_CHARSET
SYMBOL_CHARSET
TURKISH_CHARSET
其中，OEM_CHARSET 表示字符集依赖本地操作系统。
DEFAULT_CHARSET 表示字符集基于本地操作系统。例如，系统位置是 English (United States),字符集将设置为 ANSI_CHARSET。
lfOutPrecision
指定文字的输出精度。输出精度定义输出与所请求的字体高度、宽度、字符方向、行距、间距和字体类型相匹配必须达到的匹配程度。可以是以下值:
值 含义 OUT_DEFAULT_PRECIS 指定默认的映射行为。 OUT_DEVICE_PRECIS 当系统包含多个名称相同的字体时，指定设备字体。
OUT_OUTLINE_PRECIS 指定字体映射选择 TrueType 和其它的 outline-based 字体。 OUT_RASTER_PRECIS 当系统包含多个名称相同的字体时，指定光栅字体（即点阵字体）。 OUT_STRING_PRECIS 这个值并不能用于指定字体映射，只是指定点阵字体枚举数据。 OUT_STROKE_PRECIS 这个值并不能用于指定字体映射，只是指定 TrueType 和其他的 outline-based 字体，以及矢量字体的枚举数据。 OUT_TT_ONLY_PRECIS 指定字体映射只选择 TrueType 字体。如果系统中没有安装 TrueType 字体，将选择默认操作。 OUT_TT_PRECIS 当系统包含多个名称相同的字体时，指定 TrueType 字体。
lfClipPrecision
指定文字的剪辑精度。剪辑精度定义如何剪辑字符的一部分位于剪辑区域之外的字符。可以是以下值:
值 含义 CLIP_DEFAULT_PRECIS 指定默认的剪辑行为。 CLIP_STROKE_PRECIS 这个值并不能用于指定字体映射，只是指定光栅（即点阵）、矢量或 TrueType 字体的枚举数据。 CLIP_EMBEDDED 当使用内嵌的只读字体时，必须指定这个标志。 CLIP_LH_ANGLES 如果指定了该值，所有字体的旋转都依赖于坐标系统的方向是逆时针或顺时针。如果没有指定该值，设备字体始终逆时针旋转，但是其它字体的旋转依赖于坐标系统的方向。该设置影响 lfOrientation 参数的效果。
lfQuality
指定文字的输出质量。输出质量定义图形设备界面 (GDI) 必须尝试将逻辑字体属性与实际物理字体的字体属性进行匹配的仔细程度。可以是以下值:
值 含义 ANTIALIASED_QUALITY 指定输出质量是抗锯齿的（如果字体支持）。 DEFAULT_QUALITY 指定输出质量不重要。 DRAFT_QUALITY 草稿质量。字体的显示质量是不重要的。对于光栅字体（即点阵字体），缩放是有效的，这就意味着可以使用更多的尺寸，但是显示质量并不高。如果需要，粗体、斜体、下划线和删除线字体会被合成。 NONANTIALIASED_QUALITY 指定输出质量不是抗锯齿的。 PROOF_QUALITY 正稿质量。指定字体质量比匹配字体属性更重要。对于光栅字体（即点阵字体），缩放是无效的，会选用其最接近的字体大小。虽然选中 PROOF_QUALITY 时字体大小不能精确地映射，但是输出质量很高，并且不会有畸变现象。如果需要，粗体、斜体、下划线和删除线字体会被合成。

如果 ANTIALIASED_QUALITY 和 NONANTIALIASED_QUALITY 都未被选择，抗锯齿效果将依赖于控制面板中字体抗锯齿的设置。
lfPitchAndFamily
指定以常规方式描述字体的字体系列。字体系列描述大致的字体外观。字体系列用于在所需精确字体不可用时指定字体。
1-2 位指定字体间距，可以是以下值:
值 含义 DEFAULT_PITCH 指定默认间距。 FIXED_PITCH 指定固定间距。 VARIABLE_PITCH 指定可变间距。
4-7 位指定字体系列，可以是以下值:
值 含义 FF_DECORATIVE 指定特殊字体。例如 Old English。 FF_DONTCARE 指定字体系列不重要。 FF_MODERN 指定具有或不具有衬线的等宽字体。例如，Pica、Elite 和 Courier New 都是等宽字体。 FF_ROMAN 指定具有衬线的等比字体。例如 MS Serif。 FF_SCRIPT 指定设计为类似手写体的字体。例如 Script 和 Cursive。 FF_SWISS 指定不具有衬线的等比字体。例如 MS Sans Serif。
字体间距和字体系列可以用布尔运算符 OR 连接(即符号 |)。
lfFaceName
字体名称，名称不得超过 31 个字符。如果是空字符串，系统将使用第一个满足其它属性的字体。
＞outtext
功能:
这个函数用于在当前位置输出字符串。
声明:

```
void outtext(
    LPCSTR textstring,
    PIMAGE pimg = NULL
);
void outtext(
    CHAR c,
    PIMAGE pimg = NULL
);
void outtext(
    LPCWSTR textstring,
    PIMAGE pimg = NULL
);
```

```
void outtext(
    WCHAR c,
    PIMAGE pimg = NULL
);
```
参数:
textstring
要输出的字符串的指针。
c
要输出的字符。
返回值:
（无）
示例:
```
// 输出字符串
char s[] = "Hello World";
outtext(s);
// 输出字符
char c = 'A';
outtext(c);
// 输出数值，先将数字格式化输出为字符串
char s[5];
sprintf(s, "%d", 1024);
outtext(s);
```
outtextrect
功能:
这个函数用于在指定矩形范围内输出字符串。
声明:
```
void outtextrect(
    int x,
```

```
    int y,
    int w,
    int h,
    LPCSTR textstring,
    PIMAGE pimg = NULL
);
void outtextrect(
    int x,
    int y,
    int w,
    int h,
    LPCWSTR textstring,
    PIMAGE pimg = NULL
);
```

参数:

x, y, w, h

要输出字符串所在的矩形区域

textstring

要输出的字符串的指针。

返回值:

（无）

outtextxy

功能:

这个函数用于在指定位置输出字符串。

声明:

```
void outtextxy(
    int x,
    int y,
```

```
    LPCSTR textstring,
    PIMAGE pimg = NULL
);
void outtextxy(
    int x,
    int y,
    CHAR c,
    PIMAGE pimg = NULL
);
void outtextxy(
    int x,
    int y,
    LPCWSTR textstring,
    PIMAGE pimg = NULL
);
void outtextxy(
    int x,
    int y,
    WCHAR c,
    PIMAGE pimg = NULL
);
```

参数:

x

字符串输出时头字母的 x 轴的坐标值

y

字符串输出时头字母的 y 轴的坐标值。

textstring

要输出的字符串的指针。

c
要输出的字符。
返回值:
（无）
示例:
```
// 输出字符串
char s[] = "Hello World";
outtextxy(10, 20, s);
// 输出字符
char c = 'A';
outtextxy(10, 40, c);
// 输出数值，先将数字格式化输出为字符串
char s[5];
sprintf(s, "%d", 1024);
outtextxy(10, 60, s);
```
xyprintf
功能:
这个函数用于在指定位置格式化输出字符串。
声明:
```
void xyprintf(
    int x,
    int y,
    int w,
    int h,
    LPCSTR textstring,
    ...
);
void xyprintf(
```

```
    int x,
    int y,
    int w,
    int h,
    LPCWSTR textstring,
    ...
);
```

参数:

x, y, w, h

要输出字符串所在的矩形区域

textstring

要输出的字符串的指针。

...

要输出的内容的参数，类似printf。

返回值:

（无）

示例:

无

setfont

功能:

这个函数用于设置当前字体样式。

声明:

```
void setfont(
    int nHeight,
    int nWidth,
    LPCSTR lpszFace,
    PIMAGE pimg = NULL
);
```

```
void setfont(
    int nHeight,
    int nWidth,
    LPCSTR lpszFace,
    int nEscapement,
    int nOrientation,
    int nWeight,
    bool bItalic,
    bool bUnderline,
    bool bStrikeOut,
    PIMAGE pimg = NULL
);
void setfont(
    int nHeight,
    int nWidth,
    LPCSTR lpszFace,
    int nEscapement,
    int nOrientation,
    int nWeight,
    bool bItalic,
    bool bUnderline,
    bool bStrikeOut,
    BYTE fbCharSet,
    BYTE fbOutPrecision,
    BYTE fbClipPrecision,
    BYTE fbQuality,
    BYTE fbPitchAndFamily,
    PIMAGE pimg = NULL
```

```
);
void setfont(
    const LOGFONT *font,
    PIMAGE pimg = NULL
);
```

参数:

nHeight

指定高度（逻辑单位）。如果为正，表示指定的高度包括字体的默认行距; 如果为负，表示指定的高度只是字符的高度。

nWidth

字符的平均宽度（逻辑单位）。如果为 0，则比例自适应。

lpszFace

字体名称。对于此参数均有LPCSTR和LPCWSTR两个版本，以上函数声明仅列出一种。提供两个接口是为了方便能同时使用两种不同的字符集。

nEscapement

字符串的书写角度，单位 0.1 度。

nOrientation

每个字符的书写角度，单位 0.1 度。

nWeight

字符的笔画粗细，范围 0¯1000。0 表示默认粗细。使用数字或下表中定义的宏均可:

宏 粗细值 FW_DONTCARE 0 FW_THIN 100 FW_EXTRALIGHT 200 FW_ULTRALIGHT 200 FW_LIGHT 300 FW_NORMAL 400 FW_REGULAR 400 FW_MEDIUM 500 FW_SEMIBOLD 600 FW_DEMIBOLD 600 FW_BOLD 700 FW_EXTRABOLD 800 FW_ULTRABOLD 800 FW_HEAVY 900 FW_BLACK 900

bItalic

是否斜体，true / false。

bUnderline

是否有下划线，true / false。

bStrikeOut

是否有删除线，true / false。

fbCharSet

指定字符集(详见 LOGFONT 结构体)。

fbOutPrecision
指定文字的输出精度(详见 LOGFONT 结构体)。
fbClipPrecision
指定文字的剪辑精度(详见 LOGFONT 结构体)。
fbQuality
指定文字的输出质量(详见 LOGFONT 结构体)。
fbPitchAndFamily
指定以常规方式描述字体的字体系列(详见 LOGFONT 结构体)。
font
指向 LOGFONT 结构体的指针。
返回值:
(无)
示例:

```
// 设置当前字体为高 16 像素的“宋体”(忽略行距)。
setfont(-16, 0,"宋体");
outtextxy(0, 0,"测试");
// 设置输出效果为抗锯齿(LOGFONTA是MBCS版本, LOGFONTW是UTF16版本)
LOGFONTA f;
getfont(&f);                          // 获取当前字体设置
f.lfHeight = 48;                      // 设置字体高度为 48(包含行距)
strcpy(f.lfFaceName, "黑体");         // 设置字体为“黑体”
f.lfQuality = ANTIALIASED_QUALITY;    // 设置输出效果为抗锯齿
setfont(&f);                          // 设置字体样式
outtextxy(0,50,"抗锯齿效果");
```

settextjustify
功能:
这个函数用于设置文字对齐方式。
声明:

```
void settextjustify(
    int horiz,
    int vert,
    PIMAGE pimg = NULL
);
```
参数:
horiz
横向对齐方式，可选值LEFT_TEXT (默认), CENTER_TEXT, RIGHT_TEXT
vert
纵向对齐方式，可选值TOP_TEXT (默认), CENTER_TEXT, BOTTOM_TEXT
textstring
要输出的字符串的指针。
返回值:
（无）
示例:
（无）
textheight
功能:
这个函数用于获取输出字符串的高（像素为单位）
声明:
```
int textheight(
    LPCTSTR textstring,
    PIMAGE pimg = NULL
);
```
参数:
textstring
指定的字符串指针。
返回值:

该字符串实际占用的像素高度。
示例:
（无）
textwidth
功能:
这个函数用于获取字符串的宽（像素为单位）
声明:
```
int textwidth(
    LPCTSTR textstring,
    PIMAGE pimg = NULL
);
```
参数:
textstring
指定的字符串指针。
返回值:
该字符串实际占用的像素宽度。
示例:
（无）
xyprintf
功能:
这个函数用于在指定位置格式化输出字符串。
声明:
```
void xyprintf(
    int x,
    int y,
    LPCSTR textstring,
    ...
);
```

```
void xyprintf(
    int x,
    int y,
    LPCWSTR textstring,
    ...
);
```
参数:
x
字符串输出时头字母的 x 轴的坐标值
y
字符串输出时头字母的 y 轴的坐标值。
textstring
要输出的字符串的指针。
...
要输出的内容的参数，类似printf。
返回值:
（无）
示例:
无
图像处理相关函数
getimage
功能:
这个函数的四个重载分别用于从屏幕 / 文件 / 资源 / IMAGE 对象中获取图像
声明:
```
// 从屏幕获取图像
void getimage(
    PIMAGE pDstImg,    // 保存图像的 IMAGE 对象指针
    int srcX,          // 要获取图像的区域左上角 x 坐标
```

```
    int srcY,          // 要获取图像的区域左上角 y 坐标
    int srcWidth,      // 要获取图像的区域宽度
    int srcHeight      // 要获取图像的区域高度
);
// 从图片文件获取图像(bmp/jpg/gif/emf/wmf/ico)
void getimage(
    PIMAGE pDstImg,        // 保存图像的 IMAGE 对象指针
    LPCTSTR pImgFile,      // 图片文件名
    int zoomWidth = 0,     // 设定图像缩放至的宽度(0 表示默认宽度, 不缩放)
    int zoomHeight = 0     // 设定图像缩放至的高度(0 表示默认高度, 不缩放)
);
// 从资源文件获取图像(bmp/jpg/gif/emf/wmf/ico)
void getimage(
    PIMAGE pDstImg,        // 保存图像的 IMAGE 对象指针
    LPCTSTR pResType,      // 资源类型
    LPCTSTR pResName,      // 资源名称
    int zoomWidth = 0,     // 设定图像缩放至的宽度(0 表示默认宽度, 不缩放)
    int zoomHeight = 0     // 设定图像缩放至的高度(0 表示默认高度, 不缩放)
);
// 从另一个 IMAGE 对象中获取图像
void getimage(
    PIMAGE pDstImg,        // 保存图像的 IMAGE 对象指针
    const IMAGE *pSrcImg, // 源图像 IMAGE 对象
    int srcX,              // 要获取图像的区域左上角 x 坐标
    int srcY,              // 要获取图像的区域左上角 y 坐标
    int srcWidth,          // 要获取图像的区域宽度
    int srcHeight          // 要获取图像的区域高度
);
```

参数:
pimg
（详见各重载函数原型内的注释）
返回值:
（无）
示例:
请参考 putimage 函数示例。
PIMAGE
功能:
保存图像的对象
示例:
以下语句可以创建一个名为 pimg 的 PIMAGE 对象，尺寸为100x50:
PIMAGE pimg = newimage(100, 50);
注意，不需要时要 delimage(pimg); 来释放掉，切记。
更多示例请参考 putimage 函数示例。
imagefilter_blurring
功能:
这个函数用于对一图片区域进行模糊滤镜操作
声明:

```
int imagefilter_blurring (
    PIMAGE imgdest,
    int intensity,
    int alpha,
    int nXOriginDest = 0,
    int nYOriginDest = 0,
    int nWidthDest = 0,
    int nHeightDest = 0
);
```

参数:
imgdest
要进行模糊操作的图片，如果为NULL则表示操作窗口上的图片
intensity
模糊度，值越大越模糊。当值在 0x0 - 0x7F之间时，为四向模糊；当值在 0x80 - 0xFF之间时，为八向模糊，运算量会大一倍
alpha
图像亮度。取值为0x100表示亮度不变，取值为0x0表示图像变成纯黑
nXOriginDest, nYOriginDest, nWidthDest, nHeightDest
描述要进行此操作的矩形区域。如果nWidthDest和nHeightDest 为0，表示操作整张图片。
返回值:
成功返回0，否则返回非0，若imgdest传入错误，会引发运行时异常。
示例:
（无）。
putimage
功能:
这个函数的几个重载用于在屏幕或另一个图像上绘制指定图像。
声明:

```
// 绘制图像到屏幕
void putimage(
    int dstX,              // 绘制位置的 x 坐标
    int dstY,              // 绘制位置的 y 坐标
    PIMAGE pSrcImg,        // 要绘制的 IMAGE 对象指针
    DWORD dwRop = SRCCOPY  // 三元光栅操作码（详见备注）
);
// 绘制图像到屏幕(指定宽高)
void putimage(
    int dstX,              // 绘制位置的 x 坐标
    int dstY,              // 绘制位置的 y 坐标
```

```
    int dstWidth,           // 绘制的宽度
    int dstHeight,          // 绘制的高度
    PIMAGE pSrcImg,         // 要绘制的 IMAGE 对象指针
    int srcX,               // 绘制内容在 IMAGE 对象中的左上角 x 坐标
    int srcY,               // 绘制内容在 IMAGE 对象中的左上角 y 坐标
    DWORD dwRop = SRCCOPY   // 三元光栅操作码（详见备注）
);
// 绘制图像到屏幕(拉伸)
void putimage(
    int dstX,               // 绘制位置的 x 坐标
    int dstY,               // 绘制位置的 y 坐标
    int dstWidth,           // 绘制的宽度
    int dstHeight,          // 绘制的高度
    PIMAGE pSrcImg,         // 要绘制的 IMAGE 对象指针
    int srcX,               // 绘制内容在 IMAGE 对象中的左上角 x 坐标
    int srcY,               // 绘制内容在 IMAGE 对象中的左上角 y 坐标
    int srcWidth,           // 绘制内容在源 IMAGE 对象中的宽度
    int srcHeight,          // 绘制内容在源 IMAGE 对象中的高度
    DWORD dwRop = SRCCOPY   // 三元光栅操作码（详见备注）
);
// 绘制图像到另一图像
void putimage(
    PIMAGE pDstImg,         // 目标 IMAGE 对象指针
    int dstX,               // 绘制位置的 x 坐标
    int dstY,               // 绘制位置的 y 坐标
    PIMAGE pSrcImg,         // 源 IMAGE 对象指针
    DWORD dwRop = SRCCOPY   // 三元光栅操作码（详见备注）
);
```

```
// 绘制图像到另一图像(指定宽高)
void putimage(
    PIMAGE pDstImg,         // 目标 IMAGE 对象指针
    int dstX,               // 绘制位置的 x 坐标
    int dstY,               // 绘制位置的 y 坐标
    int dstWidth,           // 绘制的宽度
    int dstHeight,          // 绘制的高度
    PIMAGE pSrcImg,         // 源 IMAGE 对象指针
    int srcX,               // 绘制内容在源 IMAGE 对象中的左上角 x 坐标
    int srcY,               // 绘制内容在源 IMAGE 对象中的左上角 y 坐标
    DWORD dwRop = SRCCOPY   // 三元光栅操作码（详见备注）
);
// 绘制图像到另一图像(拉伸)
void putimage(
    PIMAGE pDstImg,         // 目标 IMAGE 对象指针
    int dstX,               // 绘制位置的 x 坐标
    int dstY,               // 绘制位置的 y 坐标
    int dstWidth,           // 绘制的宽度
    int dstHeight,          // 绘制的高度
    PIMAGE pSrcImg,         // 源 IMAGE 对象指针
    int srcX,               // 绘制内容在源 IMAGE 对象中的左上角 x 坐标
    int srcY,               // 绘制内容在源 IMAGE 对象中的左上角 y 坐标
    int srcWidth,           // 绘制内容在源 IMAGE 对象中的宽度
    int srcHeight,          // 绘制内容在源 IMAGE 对象中的高度
    DWORD dwRop = SRCCOPY   // 三元光栅操作码（详见备注）
);
```

参数:

（详见各重载函数原型内的注释）

备注:
三元光栅操作码（即位操作模式），支持全部的 256 种三元光栅操作码，常用的几种如下:
值 含义 DSTINVERT 绘制出的像素颜色 = NOT 屏幕颜色 MERGECOPY 绘制出的像素颜色 = 图像颜色 AND 当前填充颜色 MERGEPAINT 绘制出的像素颜色 = 屏幕颜色 OR (NOT 图像颜色) NOTSRCCOPY 绘制出的像素颜色 = NOT 图像颜色 NOTSRCERASE 绘制出的像素颜色 = NOT (屏幕颜色 OR 图像颜色) PATCOPY 绘制出的像素颜色 = 当前填充颜色 PATINVERT 绘制出的像素颜色 = 屏幕颜色 XOR 当前填充颜色 PATPAINT 绘制出的像素颜色 = 屏幕颜色 OR ((NOT 图像颜色) OR 当前填充颜色) SRCAND 绘制出的像素颜色 = 屏幕颜色 AND 图像颜色 SRCCOPY 绘制出的像素颜色 = 图像颜色 SRCERASE 绘制出的像素颜色 = (NOT 屏幕颜色) AND 图像颜色 SRCINVERT 绘制出的像素颜色 = 屏幕颜色 XOR 图像颜色 SRCPAINT 绘制出的像素颜色 = 屏幕颜色 OR 图像颜色
注: 1. AND / OR / NOT / XOR 为布尔位运算。2. "屏幕颜色"指绘制所经过的屏幕像素点的颜色。3. "图像颜色"是指通过 IMAGE 对象中的图像的颜色。4. "当前填充颜色"是指通过 setfillstyle 设置的用于当前填充的颜色。5. 查看全部的三元光栅操作码请详见: 三元光栅操作码。
返回值:
（无）
示例:
以下局部代码读取 c:\test.jpg 绘制在屏幕左上角:

```
PIMAGE img = newimage();
getimage(img, "c:\\test.jpg");
putimage(0, 0, img);
delimage(img);
```

以下局部代码将屏幕 (0,0) 起始的长宽各 100 像素的图像拷贝至 (200,200) 位置:

```
PIMAGE img = newimage();
getimage(img, 0, 0, 100, 100);
putimage(200, 200, img);
delimage(img);
```

putimage_alphablend
功能:
这个函数用于对两张图片进行半透明混合，并把混合结果写入目标图片。
声明:

```
int putimage_alphablend(
```

```
    PIMAGE imgdest,         // handle to dest
    PIMAGE imgsrc,          // handle to source
    int nXOriginDest,       // x-coord of destination upper-left corner
    int nYOriginDest,       // y-coord of destination upper-left corner
    unsigned char alpha,    // alpha
    int nXOriginSrc = 0,    // x-coord of source upper-left corner
    int nYOriginSrc = 0,    // y-coord of source upper-left corner
    int nWidthSrc = 0,      // width of source rectangle
    int nHeightSrc = 0      // height of source rectangle
);
```

参数:

imgdest

要进行半透明混合的目标图片，如果为NULL则表示操作窗口上的图片

imgsrc

要进行半透明混合的源图片，该操作不会改变源图片

nXOriginDest, nYOriginDest

要开始进行混合的目标图片坐标，该坐标是混合区域的左上角

alpha

透明度值，如果为0x0，表示源图片完全透明，如果为0xFF，表示源图片完全不透明。

nXOriginDest, nYOriginDest, nWidthDest, nHeightDest

描述要进行此操作的源图矩形区域。如果nWidthDest和nHeightDest 为0，表示操作整张图片。

返回值:

成功返回0，否则返回非0，若imgdest或imgsrc传入错误，会引发运行时异常。

示例:

（无）。

putimage_transparent

功能:

这个函数用于对两张图片进行透明混合，并把混合结果写入目标图片。

声明:

```
int putimage_transparent(
    PIMAGE imgdest,          // handle to dest
    PIMAGE imgsrc,           // handle to source
    int nXOriginDest,        // x-coord of destination upper-left corner
    int nYOriginDest,        // y-coord of destination upper-left corner
    COLORREF crTransparent, // color to make transparent
    int nXOriginSrc = 0,     // x-coord of source upper-left corner
    int nYOriginSrc = 0,     // y-coord of source upper-left corner
    int nWidthSrc = 0,       // width of source rectangle
    int nHeightSrc = 0       // height of source rectangle
);
```

参数:

imgdest

要进行透明混合的目标图片，如果为NULL则表示操作窗口上的图片

imgsrc

要进行透明混合的源图片，该操作不会改变源图片

nXOriginDest, nYOriginDest

要开始进行混合的目标图片坐标，该坐标是混合区域的左上角

crTransparent

关键色。源图片上为该颜色值的像素，将忽略，不会改写目标图片上相应位置的像素。

nXOriginDest, nYOriginDest, nWidthDest, nHeightDest

描述要进行此操作的源图矩形区域。如果nWidthDest和nHeightDest 为0，表示操作整张图片。

返回值:

成功返回0，否则返回非0，若imgdest或imgsrc传入错误，会引发运行时异常。

示例:

（无）。

putimage_alphatransparent

功能:
这个函数用于对两张图片进行透明/半透明混合，并把混合结果写入目标图片。
声明:

```
int putimage_alphatransparent(
    PIMAGE imgdest,          // handle to dest
    PIMAGE imgsrc,           // handle to source
    int nXOriginDest,        // x-coord of destination upper-left corner
    int nYOriginDest,        // y-coord of destination upper-left corner
    COLORREF crTransparent, // color to make transparent
    unsigned char alpha,     // alpha
    int nXOriginSrc = 0,     // x-coord of source upper-left corner
    int nYOriginSrc = 0,     // y-coord of source upper-left corner
    int nWidthSrc = 0,       // width of source rectangle
    int nHeightSrc = 0       // height of source rectangle
);
```

参数:
imgdest
要进行半透明混合的目标图片，如果为NULL则表示操作窗口上的图片
imgsrc
要进行半透明混合的源图片，该操作不会改变源图片
nXOriginDest, nYOriginDest
要开始进行混合的目标图片坐标，该坐标是混合区域的左上角
crTransparent
关键色。源图片上为该颜色值的像素，将忽略，不会改写目标图片上相应位置的像素。
alpha
透明度值，如果为0x0，表示源图片完全透明，如果为0xFF，表示源图片完全不透明。
nXOriginDest, nYOriginDest, nWidthDest, nHeightDest
描述要进行此操作的源图矩形区域。如果nWidthDest和nHeightDest 为0，表示操作整张图片。

返回值:
成功返回0，否则返回非0，若imgdest或imgsrc传入错误，会引发运行时异常。
示例:
（无）。
三元光栅操作码
这篇补充文档列出了 putimage 函数支持的所有三元光栅操作码。
三元光栅操作码定义了源图像与屏幕图像的位合并形式，这个合并形式是以下三个操作数对应像素的布尔运算:
操作数 含义 D 屏幕图像 P 当前填充颜色 S 源图像
布尔运算符包括以下几种:
操作 含义 a 位的 AND 运算(双目运算) n 位的 NOT 运算(单目运算) o 位的 OR 运算(双目运算) x 位的 XOR 运算(双目运算)
所有的布尔操作都采用逆波兰表示法，例如，“当前填充颜色 or 源图像”可表示为: PSo。（当然 SPo 也是等价的，这里只列举出了其中一种等价格式）
三元光栅操作码是 32位int 类型，其高位字是布尔操作索引，低位字是操作码。布尔操作索引的 16 个位中，高 8 位用 0 填充，低 8 位是当前填充颜色、源图像和屏幕的布尔操作结果。例如，PSo 和 DPSoo 的操作索引如下:
P S D PSo DPSoo 0 0 0 0 0 0 0 1 0 1 0 1 0 1 1 0 1 1 1 1 1 0 0 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 操作索引: 00FCh 00FEh
上例中，PSo 的操作索引是 00FC (从下往上读)，DPSoo 的是 00FE。这些值定义了相应的三元光栅操作码在“三元光栅操作码”表格中的位置，PSo 在 252 (00FCh) 行，DPSoo 在 254 (00FEh) 行。常用的三元光栅操作码已经定义了常量名，程序中可以直接使用。
三元光栅操作码
布尔功能 (16 进制) 光栅操作 (16 进制) 布尔功能的逆波兰表示法 常量名 00 00000042 0 BLACKNESS 01 00010289 DPSoon 02 00020C89 DPSona 03 000300AA PSon 04 00040C88 SDPona 05 000500A9 DPon 06 00060865 PDSxnon 07 000702C5 PDSaon 08 00080F08 SDPnaa 09 00090245 PDSxon 0A 000A0329 DPna 0B 000B0B2A PSDnaon 0C 000C0324 SPna 0D 000D0B25 PDSnaon 0E 000E08A5 PDSonon 0F 000F0001 Pn 10 00100C85 PDSona 11 001100A6 DSon NOTSRCERASE 12 00120868 SDPxnon 13 001302C8 SDPaon 14 00140869 DPSxnon 15 001502C9 DPSaon 16 00165CCA PSDPSanaxx 17 00171D54 SSPxDSxaxn 18 00180D59 SPxPDxa 19 00191CC8 SDPSanaxn 1A 001A06C5 PDSPaox 1B 001B0768 SDPSxaxn 1C 001C06CA PSDPaox 1D 001D0766 DSPDxaxn 1E 001E01A5 PDSox 1F 001F0385 PDSoan 20 00200F09 DPSnaa 21 00210248 SDPxon 22 00220326 DSna 23 00230B24 SPDnaon 24 00240D55 SPxDSxa 25 00251CC5 PDSPanaxn 26 002606C8 SDPSaox 27 00271868 SDPSxnox 28 00280369 DPSxa 29 002916CA PSDPSaoxxn 2A 002A0CC9 DPSana 2B 002B1D58 SSPxPDxaxn 2C 002C0784 SPDSoax 2D 002D060A PSDnox 2E 002E064A PSDPxox 2F 002F0E2A PSDnoan 30 0030032A PSna 31 00310B28 SDPnaon 32 00320688 SDPSoox 33 00330008 Sn NOTSRCCOPY 34 003406C4 SPDSaox 35 00351864 SPDSxnox 36 003601A8 SDPox 37 00370388 SDPoan 38 0038078A PSDPoax 39

00390604 SPDnox 3A 003A0644 SPDSxox 3B 003B0E24 SPDnoan 3C 003C004A PSx 3D 003D18A4 SPDSonox 3E 003E1B24 SPDSnaox 3F 003F00EA PSan 40 00400F0A PSDnaa 41 00410249 DPSxon 42 00420D5D SDxPDxa 43 00431CC4 SPDSanaxn 44 00440328 SDna SRCERASE 45 00450B29 DPSnaon 46 004606C6 DSPDaox 47 0047076A PSDPxaxn 48 00480368 SDPxa 49 004916C5 PDSPDaoxxn 4A 004A0789 DPSDoax 4B 004B0605 PDSnox 4C 004C0CC8 SDPana 4D 004D1954 SSPxDSxoxn 4E 004E0645 PDSPxox 4F 004F0E25 PDSnoan 50 00500325 PDna 51 00510B26 DSPnaon 52 005206C9 DPSDaox 53 00530764 SPDSxaxn 54 005408A9 DPSonon 55 00550009 Dn DSTINVERT 56 005601A9 DPSox 57 00570389 DPSoan 58 00580785 PDSPoax 59 00590609 DPSnox 5A 005A0049 DPx PATINVERT 5B 005B18A9 DPSDonox 5C 005C0649 DPSDxox 5D 005D0E29 DPSnoan 5E 005E1B29 DPSDnaox 5F 005F00E9 DPan 60 00600365 PDSxa 61 006116C6 DSPDSaoxxn 62 00620786 DSPDoax 63 00630608 SDPnox 64 00640788 SDPSoax 65 00650606 DSPnox 66 00660046 DSx SRCINVERT 67 006718A8 SDPSonox 68 006858A6 DSPDSonoxxn 69 00690145 PDSxxn 6A 006A01E9 DPSax 6B 006B178A PSDPSoaxxn 6C 006C01E8 SDPax 6D 006D1785 PDSPDoaxxn 6E 006E1E28 SDPSnoax 6F 006F0C65 PDSxnan 70 00700CC5 PDSana 71 00711D5C SSDxPDxaxn 72 00720648 SDPSxox 73 00730E28 SDPnoan 74 00740646 DSPDxox 75 00750E26 DSPnoan 76 00761B28 SDPSnaox 77 007700E6 DSan 78 007801E5 PDSax 79 00791786 DSPDSoaxxn 7A 007A1E29 DPSDnoax 7B 007B0C68 SDPxnan 7C 007C1E24 SPDSnoax 7D 007D0C69 DPSxnan 7E 007E0955 SPxDSxo 7F 007F03C9 DPSaan 80 008003E9 DPSaa 81 00810975 SPxDSxon 82 00820C49 DPSxna 83 00831E04 SPDSnoaxn 84 00840C48 SDPxna 85 00851E05 PDSPnoaxn 86 008617A6 DSPDSoaxx 87 008701C5 PDSaxn 88 008800C6 DSa SRCAND 89 00891B08 SDPSnaoxn 8A 008A0E06 DSPnoa 8B 008B0666 DSPDxoxn 8C 008C0E08 SDPnoa 8D 008D0668 SDPSxoxn 8E 008E1D7C SSDxPDxax 8F 008F0CE5 PDSanan 90 00900C45 PDSxna 91 00911E08 SDPSnoaxn 92 009217A9 DPSDPoaxx 93 009301C4 SPDaxn 94 009417AA PSDPSoaxx 95 009501C9 DPSaxn 96 00960169 DPSxx 97 0097588A PSDPSonoxx 98 00981888 SDPSonoxn 99 00990066 DSxn 9A 009A0709 DPSnax 9B 009B07A8 SDPSoaxn 9C 009C0704 SPDnax 9D 009D07A6 DSPDoaxn 9E 009E16E6 DSPDSaoxx 9F 009F0345 PDSxan A0 00A000C9 DPa A1 00A11B05 PDSPnaoxn A2 00A20E09 DPSnoa A3 00A30669 DPSDxoxn A4 00A41885 PDSPonoxn A5 00A50065 PDxn A6 00A60706 DSPnax A7 00A707A5 PDSPoaxn A8 00A803A9 DPSoa A9 00A90189 DPSoxn AA 00AA0029 D AB 00AB0889 DPSono AC 00AC0744 SPDSxax AD 00AD06E9 DPSDaoxn AE 00AE0B06 DSPnao AF 00AF0229 DPno B0 00B00E05 PDSnoa B1 00B10665 PDSPxoxn B2 00B21974 SSPxDSxox B3 00B30CE8 SDPanan B4 00B4070A PSDnax B5 00B507A9 DPSDoaxn B6 00B616E9 DPSDPaoxx B7 00B70348 SDPxan B8 00B8074A PSDPxax B9 00B906E6 DSPDaoxn BA 00BA0B09 DPSnao BB 00BB0226 DSno MERGEPAINT BC 00BC1CE4 SPDSanax BD 00BD0D7D SDxPDxan BE 00BE0269 DPSxo BF 00BF08C9 DPSano C0 00C000CA PSa MERGECOPY C1 00C11B04 SPDSnaoxn C2 00C21884 SPDSonoxn C3 00C3006A PSxn C4 00C40E04 SPDnoa C5 00C50664 SPDSxoxn C6 00C60708 SDPnax C7 00C707AA PSDPoaxn C8 00C803A8 SDPoa C9 00C90184 SPDoxn CA 00CA0749 DPSDxax CB 00CB06E4 SPDSaoxn CC 00CC0020 S SRCCOPY CD 00CD0888 SDPono CE 00CE0B08 SDPnao CF 00CF0224 SPno D0 00D00E0A PSDnoa D1 00D1066A PSDPxoxn D2 00D20705 PDSnax D3 00D307A4 SPDSoaxn D4 00D41D78 SSPxPDxax D5 00D50CE9 DPSanan D6 00D616EA PSDPSaoxx D7 00D70349 DPSxan D8 00D80745 PDSPxax D9 00D906E8 SDPSaoxn DA 00DA1CE9 DPSDanax DB 00DB0D75 SPxDSxan DC 00DC0B04 SPDnao DD 00DD0228 SDno DE 00DE0268 SDPxo DF 00DF08C8 SDPano E0

00E003A5 PDSoa E1 00E10185 PDSoxn E2 00E20746 DSPDxax E3 00E306EA PSDPaoxn E4 00E40748 SDPSxax E5 00E506E5 PDSPaoxn E6 00E61CE8 SDPSanax E7 00E70D79 SPxPDxan E8 00E81D74 SSPxDSxax E9 00E95CE6 DSPDSanaxxn EA 00EA02E9 DPSao EB 00EB0849 DPSxno EC 00EC02E8 SDPao ED 00ED0848 SDPxno EE 00EE0086 DSo SRCPAINT EF 00EF0A08 SDPnoo F0 00F00021 P PATCOPY F1 00F10885 PDSono F2 00F20B05 PDSnao F3 00F3022A PSno F4 00F40B0A PSDnao F5 00F50225 PDno F6 00F60265 PDSxo F7 00F708C5 PDSano F8 00F802E5 PDSao F9 00F90845 PDSxno FA 00FA0089 DPo FB 00FB0A09 DPSnoo PATPAINT FC 00FC008A PSo FD 00FD0A0A PSDnoo FE 00FE02A9 DPSoo FF 00FF0062 1 WHITENESS

键盘鼠标输入函数

FlushMouseMsgBuffer

功能:

这个函数用于清空鼠标消息缓冲区。

声明:

void FlushMouseMsgBuffer();

参数:

（无）

返回值:

（无）

示例:

（无）

getch

功能:

这个函数用于获取键盘字符输入，如果当前没有输入，则等待。

声明:

int getch();

参数:

（无）

返回值:

如果存在键盘字符输入，返回 1；否则返回 0。

示例:

（无）
getkey
功能:
这个函数用于获取键盘消息，如果当前没有消息，则等待。
声明:
key_msg getkey();
参数:
（无）
返回值:
返回 key_msg 结构体
示例:
（无）
getmouse
功能:
这个函数用于获取一个鼠标消息。如果当前鼠标消息队列中没有，就一直等待。
声明:
mouse_msg getmouse();
参数:
（无）
返回值:
（无）
示例:
（无）
GetMouseMsg
功能:
这个函数用于获取一个鼠标消息。如果当前鼠标消息队列中没有，就一直等待。
声明:
MOUSEMSG GetMouseMsg();

参数:
（无）
返回值:
（无）
示例:
（无）
kbhit
功能:
这个函数用于检测当前是否有键盘字符输入。
声明:
int kbhit();
参数:
（无）
返回值:
如果存在键盘字符输入，返回 1；否则返回 0。
示例:
（无）
kbmsg
功能:
这个函数用于检测当前是否有键盘消息。
声明:
int kbmsg();
参数:
（无）
返回值:
如果存在键盘消息，返回 1；否则返回 0。
示例:
（无）

key_msg 结构体
功能:
这个结构体用于保存键盘消息
声明:
```
typedef struct key_msg {
    UINT msg;
    UINT key;
    UINT flags;
}key_msg;
```
成员:
msg
指定鼠标消息类型，可为以下值:
值 含义 key_msg_down 键盘按下消息。 key_msg_up 键盘弹起消息。 key_msg_char 键盘字符输入消息。
key
如果是按下和弹起的消息，则表示按键虚拟键码，否则为gbk编码字符消息
flags
按键参数，可能为以下值的组合:
值 含义 key_flag_shift 同时按下了shift key_flag_ctrl 同时按下了control
示例:
（无）
keystate
功能:
这个函数用于判断某按键是否被按下。
```
int keystate(int key);
```
参数:
key
虚拟键码，一个Windows定义的，能与键盘按键一一对应的码表。如果是字母键或者数字键，则其虚拟键码与ASCII字符值相同，比如'A'键，它的虚拟键码也是'A'，但不能是'a'。小键盘上的数字则用类似VK_NUMPAD3的宏表示，详细可以查看VK_XXX这一系列宏的定义。

返回值:
返回非0表示这个按键按下了，返回0表示没有按下。该函数全局有效，即使窗口没有得到输入焦点，一样照样取得键盘的实际状态。
示例:

```
if (keystate(VK_ESCAPE))
{
    // ESC键按下了
}
```

mousemsg
功能:
这个函数用于检测当前是否有鼠标消息。
声明:

```
int mousemsg();
```

参数:
（无）
返回值:
如果存在鼠标消息，返回 1；否则返回 0。
示例:
（无）
mouse_msg 结构体
功能:
这个结构体用于保存鼠标消息
声明:

```
typedef struct mouse_msg {
    UINT msg;
    INT  x;
    INT  y;
    UINT flags;
    INT  wheel;
```

}mouse_msg;
成员:
msg
指定鼠标消息类型，可为以下值:
值 含义 mouse_msg_move 鼠标移动消息。 mouse_msg_wheel 滚轮拨动消息。 mouse_msg_down 鼠标按键按下消息。 mouse_msg_up 鼠标按键弹起消息。
x
当前鼠标 x 坐标
y
当前鼠标 y 坐标
flags
按键参数，可能为以下值的组合:
值 含义 mouse_flag_left 鼠标左键 mouse_flag_right 鼠标右键 mouse_flag_mid 鼠标中键 mouse_flag_shift 同时按下了shift
mouse_flag_ctrl 同时按下了control
wheel
鼠标滚轮滚动值，一般情况下为 120 的倍数或者约数。
示例:
（无）
MOUSEMSG 结构体
功能:
这个结构体用于保存鼠标消息
声明:

```
struct MOUSEMSG
{
    UINT uMsg;       // 当前鼠标消息
    int mkCtrl;      // Ctrl 键是否按下
    int mkShift;     // Shift 键是否按下
    int mkLButton;   // 鼠标左键是否按下
```

```
    int mkMButton;  // 鼠标中键是否按下
    int mkRButton;  // 鼠标右键是否按下
    int x;          // 当前鼠标 x 坐标
    int y;          // 当前鼠标 y 坐标
    int wheel;      // 鼠标滚轮滚动值
};
```

成员:

uMsg:

指定鼠标消息类型，可为以下值:

值 含义 WM_MOUSEMOVE 鼠标移动消息。 WM_MOUSEWHEEL 鼠标滚轮拨动消息。 WM_LBUTTONDOWN 左键按下消息。 WM_LBUTTONUP 左键弹起消息。 WM_LBUTTONDBLCLK 左键双击消息。 WM_MBUTTONDOWN 中键按下消息。 WM_MBUTTONUP 中键弹起消息。 WM_MBUTTONDBLCLK 中键双击消息。 WM_RBUTTONDOWN 右键按下消息。 WM_RBUTTONUP 右键弹起消息。 WM_RBUTTONDBLCLK 右键双击消息。

mkCtrl

Ctrl 键是否按下

mkShift

Shift 键是否按下

mkLButton

鼠标左键是否按下

mkMButton

鼠标中键是否按下

mkRButton

鼠标右键是否按下

x

当前鼠标 x 坐标

y

当前鼠标 y 坐标

wheel

鼠标滚轮滚动值，一般情况下为 120 的倍数或者约数。

示例:
（无）
mousepos
功能:
这个函数用于获取当前鼠标坐标。
声明:
int mousepos(int *x, int *y);
参数:
x
用来接收横坐标
y
用来接收纵坐标
返回值:
（无）
示例:
（无）
showmouse
功能:
这个函数用于检测设置鼠标隐藏。
声明:
int showmouse(int bShow);
参数:
bShow
为0则不显示，非0为显示。默认显示。
返回值:
返回上一次调用时设置的值，第一次调用的话返回1。
示例:
（无）

时间函数
api_sleep
功能:
与Sleep函数完全相同，单纯延迟指定时间（精确程度由系统API决定），其它事情什么都不干。
声明:
VOID api_sleep(long dwMilliseconds);
参数:
dwMilliseconds
要延迟的时间，以毫秒为单位，如果为0则不产生延时的作用（相当于无意义调用）。不会附带刷新窗口的作用。
返回值:
（无）
示例:
（无）
delay
功能:
至少延迟以毫秒为单位的时间。
声明:
void delay(long Milliseconds);
参数:
Milliseconds
要延迟的时间，以毫秒为单位
返回值:
（无）
示例:
（无）
delay_ms
功能:
平均延迟以毫秒为单位的时间。

声明:
void delay_ms(long Milliseconds);
参数:
Milliseconds
要延迟的时间，以毫秒为单位
返回值:
（无）
示例:
（无）
delay_fps
功能:
延迟以FPS为准的时间，以实现稳定帧率。
声明:
void delay_fps(long fps);
void delay_fps(double fps);
参数:
fps
要得到的帧率，平均延迟1000/fps毫秒,并更新FPS计数值。这个函数一秒最多能调用fps次。
返回值:
（无）
示例:
（无）
delay_jfps
功能:
延迟以FPS为准的时间，以实现稳定帧率（带跳帧）。
声明:
void delay_jfps(long fps);
void delay_fps(double fps);

参数:
fps
要得到的帧率，平均延迟1000/fps毫秒,并更新FPS计数值。这个函数一秒最多能调用fps次。注意的是，即使这帧跳过了，仍然会更新FPS计数值。
返回值:
（无）
示例:
（无）
fclock
功能:
获取当前程序从初始化起经过的时间，以秒为单位
声明:
double fclock();
参数:
（无）
返回值:
返回一个以秒为单位的浮点数，精度比API的GetTickCount稍高。程序中使用一般用于求时间差，一般不要直接使用这个值。
示例:
（无）
数学函数
函数或数据 说明 rotate_point3d_x 把一个3d点绕x轴旋转 rotate_point3d_y 把一个3d点绕y轴旋转 rotate_point3d_z 把一个3d点绕z轴旋转 VECTOR3D 库提供的3d向量类，以下为类的成员简介 VECTOR3D::operator = 向量复制 VECTOR3D::operator + 3d向量加法 VECTOR3D::operator - 3d向量减法 VECTOR3D::operator * 与浮点数相乘时为向量缩放，与向量相乘时为点乘 VECTOR3D::operator & 向量叉乘 VECTOR3D::GetAngel 计算两个3d向量的夹角 VECTOR3D::GetModule 计算3d向量的模 VECTOR3D::GetSqrModule 计算3d向量模的平方 VECTOR3D::Rotate 3d向量绕另一任意向量旋转，或者按指定旋转角旋转 VECTOR3D::SetModule 在保持方向不变的情况下把3d向量长度改为设定值
随机函数
random
功能:
这个函数用于生成某范围内的随机整数

声明:
unsigned int random(unsigned int n = 0);
参数:
n
生成0至n-1之间的整数。
如果n为0，则返回0 - 0xFFFFFFFF的整数。
返回值:
返回一个随机整数。
其它说明:
建议不要使用stdlib里的rand函数，而改用本函数。本函数使用专业的随机数生成算法，随机性远超系统的rand函数。另，千万不要自己random() % n的方式取获得一个范围内的随机数，请使用random(n)，切记。本随机序列的初始化只能调用randomize函数，不能使用srand。
randomf
功能:
这个函数用于生成0-1范围内的随机浮点数。
声明:
double randomf();
参数:
无
返回值:
返回一个0-1之间的随机浮点数，0.0可能取得到，1.0一定取不到。
其它说明:
本随机序列的初始化只能调用randomize函数，不能使用srand。
randomize
功能:
这个函数用于初始化随机数序列。如果不调用本函数，那么random返回的序列将会是确定不变的。
声明:
void randomize();
参数:

（无）
返回值:
（无）
示例:
（无）
其它函数
getfps
功能:
这个函数用于获取当前刷新率。
声明:
```
float getfps(
    int flag = 1
);
```
参数:
flag
仅能为0或者1，如果为1，查询的是逻辑帧数；如果为0，查询的是渲染帧数。两者之差可以得到无效帧数（被跳过渲染的帧数，仅在调用delay_jfps会产生）。如果没有调用过delay_jfps，那么两者无区别。
返回值:
返回当前刷新率。
说明:
FPS（Frames Per Second）：每秒传输帧数。通常，这个帧数在动画或者游戏里，至少要达到30才能基本流畅。现代液晶显示器均使用60FPS的刷新率，所以，如果你希望在你的显示器上达到最佳效果，那你需要至少60FPS。
而使内部FPS计数增加的方式是当你绘图后，调用delay族函数，如：delay，delay_ms，delay_fps，Sleep，否则你不调用这些函数时，FPS永远为0而不会变化。
示例:
参见示例程序中的“星空”
GetHWnd
功能:

这个函数用于获取绘图窗口句柄。
声明:
```
HWND GetHWnd();
```
参数:
（无）
返回值:
返回绘图窗口句柄。
说明:
在 Windows 下，句柄是一个窗口的标识，得到句柄后，可以使用 Windows SDK 中的各种命令实现对窗口的控制。
示例:
```
// 获得窗口句柄
HWND hWnd = GetHWnd();
// 使用 API 函数修改窗口名称
SetWindowText(hWnd, TEXT("Hello!"));
```
inputbox_getline
功能:
使用对话框让用户输入一个字符串
声明:
```
int inputbox_getline(LPCSTR  title, LPCSTR  text, LPSTR  buf, int len);
int inputbox_getline(LPCWSTR title, LPCWSTR text, LPWSTR buf, int len);
```
参数:
title
对话框标题
text
对话框内显示的提示文字，可以使用'\n'或者'\t'进行格式控制。
buf
用于接收输入的字符串指针，指向一个缓冲区
len

指定buf指向的缓冲区的大小，同时也会限制在对话框里输入的最大长度
返回值:
返回1表示输入有效，buf中的内容为用户所输入的数据，返回0表示输入无效，同时buf清空。
示例:

```
#include "graphics.h"
int main()
{
    initgraph(640, 480);
    char str[100];
    inputbox_getline("这是一个对话框",
                     "请随便\n输入一串字符，输入完请回车",
                     str,
                     sizeof(str)/sizeof(*str));
    outtextxy(0, 0, str);
    getch();
    return 0;
}
```