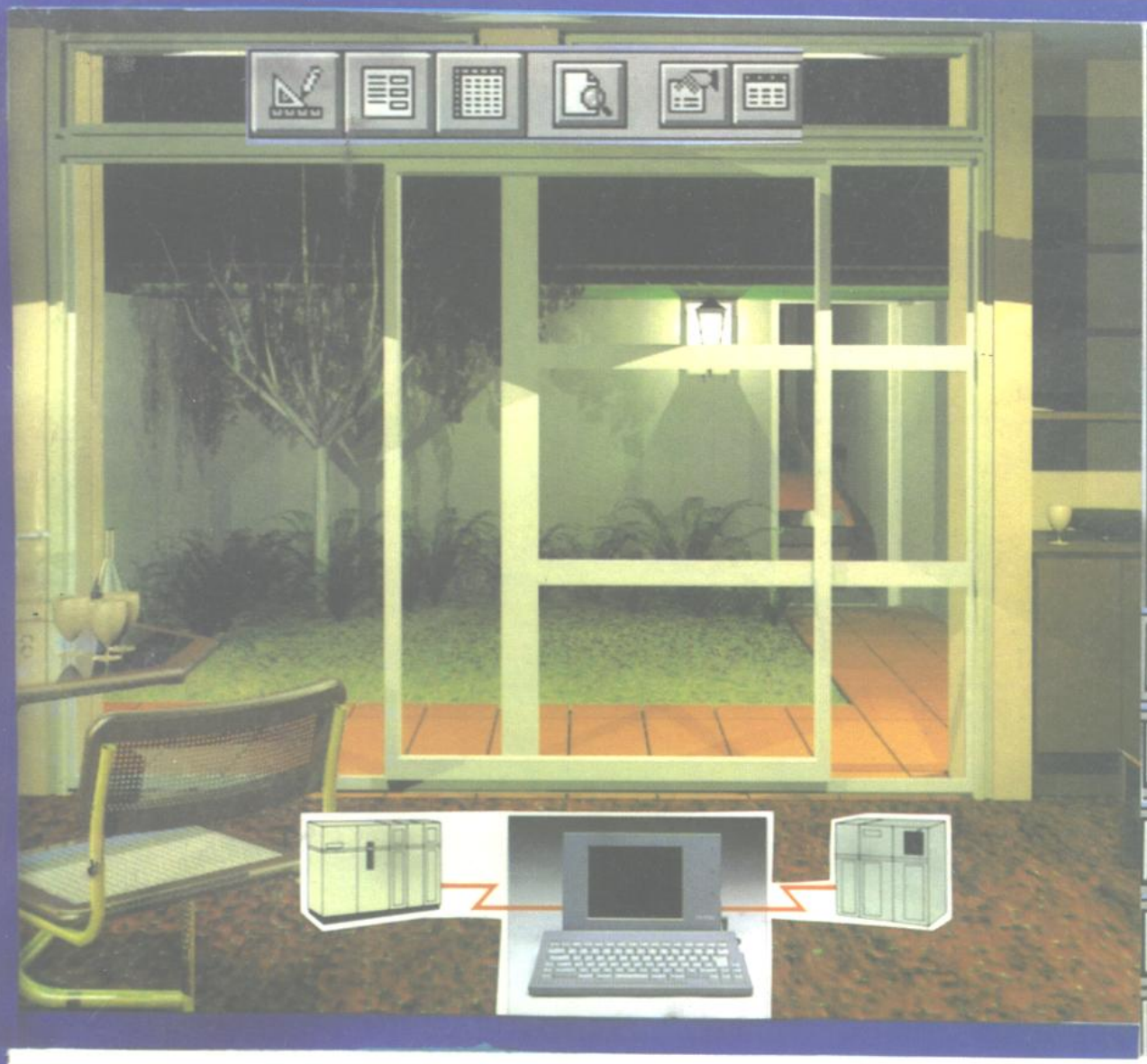


Borland C++ for Windows

蔡明志 著

Windows 程序设计·绘图篇
——使用 Borland C++ for Windows

科学出版社

北京希望电脑公司 Windows 最新技术丛书

Windows 程序设计 · 绘图篇

— 使用 Borland C++ for Windows

蔡明志　　原著
徐　蔓　　等改编
刘　伟　　等审校

科学出版社

1993 年

(京)新登字 092 号

内 容 简 介

本书在介绍 MS Windows 程序设计和 Borland C++面向对象的程序设计语言的基础上，主要针对 Windows GDI 即图形设备界面，进行进一步的研究与讨论。GDI 支持强大的绘图功能，具有彻底的设备独立性，可让程序自行控制映射模式、画笔、画刷、调色板、位图像、图元文件、字体及文字输出、打印机输出和图形的绘制等，产生完美的输出效果。GDI 绘图的功能虽然繁多，但本书由浅入深，从基本概念一步步地详述如何利用 GDI 的各项功能。另外，针对各项功能举出了许多范例，让读者更容易了解如何编写该项功能，读者可以将这些代码直接加到自己的程序中，提高软件的开发效率。因此本书作为 Windows 程序设计的进阶书籍是很有价的，适于计算机用户和软件开发人员阅读。

需要本书的用户请直接与北京 8721 信箱联系，邮码 100080，电话 2562329。

版 权 声 明

Windows 程序设计·绘图篇
—使用 Borland C++ for Windows

蔡明志　原 著
徐 蔓　等改编
刘 伟　等审校
责任编辑　马长芳
科学出版社出版
北京东黄城根北街 16 号
邮政编码：100717
双青印刷厂印刷
新华书店北京发行所发行　各地新华书店经售
★
1993 年 9 月第一版　开本：787×1092　1/16
1993 年 9 月第一次印刷　印张：39.87
印数：0—5000　字数：939 000
ISBN 7-03-003904-1/TP·303
定价：88.00 元

前　言

Windows应用程序设计是很复杂的，它可供我们使用的资源相当的多。函数库数目之多及其复杂程度确实令人咋舌，更重要的一点是，它与在DOS下的程序设计有相当大的差别，程序的流程不再是由头到尾一行一行地执行，而是以消息驱动为模式；另外，由于在Windows环境下是多任务的环境，各个应用程序只有遵循一定的规则来利用系统的资源，才能与各个应用程序互相配合，共同在Windows之下工作，这是程序设计师最难适应的一点。

本书在介绍MS Windows程序设计和Borland C++面向对象的程序设计语言的基础上，主要针对Windows GDI (Graphic Device Interface) 即图形设备界面，进行更进一步的研究与讨论。GDI支持相当强大的绘图功能，可让程序自行控制映射模式、画笔、画刷的样式、调色板、位图像等等；更具代表性的一点，就是它的设备独立性，无论输出设备提供了几个颜色，屏幕之纵横比为何，Windows的GDI绘图函数都会利用输出设备的相关信息产生出最完美的输出效果。

GDI绘图的功能虽然繁多，但本书由浅入深，从基本观念一步步地详述如何利用GDI的各项功能，另外，针对各项功能本书举出了许多范例，让读者更容易了解如何编写各项功能，读者可以将这些代码直接加到自己的程序中，达到软件重用的目的，从而提高软件的开发效率。

本书简体字中文版由徐蔓主持改编，徐蔓、刘伟、曲明、杨漉、朱秋竣、田澄等编写，刘伟、徐蔓、李爱乐等审校。编审者愿与广大读者一起研究和学习，共同提高。

目 录

第一章 基础知识

Windows 应用程序的设计是件不太容易做的事．首先，它包含了许许多多的功能，程序设计师往往不能只学会某些种，因为它们之间可能都是互相关连的；其次，Windows 不会放任程序设计师无限制地去使用任何资源．因为在 Windows 环境之下，程序的执行是多任务的(Multi-task)，有许多的程序可能都在使用系统的资源，所以，不但 Windows 不希望我们浪费系统太多的资源，身为程序设计师的我们自己也应当注意这个问题，否则，恶性循环的后果可能也会影响到自己的身上．由此可见，写程序时必须遵循 Windows 给我们的限制，善加利用 Windows 所提供的工具．

Windows 程序设计最重要的概念是"消息驱动(message-driven)"，因为 Windows 必须利用消息的传送，才能使得许多应用程序同时显示在用户面前，一起为用户服务，所以作为程序设计师，我们也必须依照这种逻辑来设计程序．当然由于它和以往由上而下、一行一行执行的程序大不相同，所以这对 Windows 程序设计的初学者来说也是最难适应的部分．但所谓熟能生巧，只要您进入了 Windows 程序设计的领域，一天从早到晚都会见到消息的身影．只要您知道消息的概念，以及该消息的来源、意义、配合上参考书籍，相信您会越来越得心应手的．

本书在介绍 Windows 程序设计和 Borland C++面向对象的程序设计语言的基础上，着重讨论绘图设备界面，即 GDI(Graphics Device Interface)，算是 Windows 程序设计的进阶书籍，研究用 Borland C++程序设计语言在 Windows 环境下开发用户界面和绘图设备界面。

1-1 Windows 绘图简介

在设计 Windows 应用程序时，必须牢记几件事:

- ☐ 它不再像DOS环境下执行程序那样，有本文模式（例如，80行X25列）及绘图模式两种，而是纯粹只有唯一的绘图模式；而且 C 语言的绘图函数也都不能再使用了，只能使用 Windows 提供的绘图函数．
- ☐ 它不再像DOS环境下执行程序那样，程序的执行可占用整个屏幕，而是每个程序只能在各个窗口的工作区域(Client Area)内作图．虽然也可以在工作区域之外的地方作图，但那可能会破坏掉其他窗口的内容．
- ☐ 虽然可以在工作区域之内尽情地绘图，但是Windows不会保证显示在工作区域内的图形数据不会消失掉．举例来说，当别的窗口的肖像(icon)或对话框(dialog box)覆盖在我们的窗口上移走后，Windows 会试图为我们挽回被覆盖掉的区域内的图形，但有时它并不能做到如此．甚至当我们改变窗口的大小时，工作区域的内容都有可能会改变，而我们自己必须负责将它还原．

针对以上的说明，我们来看看一个简单的例子:

范例 demo1__1

计划文件 demo1__1.prj 包括:

demo1__1.c
demo1__1.rc
demo1__1.def

程序 DEMO1__1.H

```
1 / / Constants defined for Command menu
2
3 #define IDM__DRAW          100
4 #define IDM__CLEAR         110
5 #define IDM__EXIT          120
6
7 / / Constants defined for Shape menu
8
9 #define IDM__LINE          200
10 #define IDM__RECT         210
11 #define IDM__ELLIP        220
12
```

程序 DEMO1__1.RC

```
1 #include "demo1__1.h"
2
3 Demo1__1 MENU
4 BEGIN
5     POPUP       "&Command"
6         BEGIN
7             MENUITEM      "&Draw"        IDM__DRAW
8             MENUITEM      "&Clear"       IDM__CLEAR
9             MENUITEM      "&Exit"        IDM__EXIT
10        END
11    POPUP       "&Shape"
12        BEGIN
```

```
            MENUITEM      "&Line",        IDM_LINE, CHECKED
            MENUITEM      "&Rectangle",   IDM_RECT
            MENUITEM      "&Ellipse",     IDM_ELLIP
        END
END
```

程序 **DEMO1_1.DEF**

```
NAME            Demo1_1

DESCRIPTION     'The simple example'

EXETYPE         WINDOWS

STUB            'WINSTUB.EXE'

CODE            PRELOAD MOVEABLE
DATA            PRELOAD MOVEABLE MULTIPLE

HEAPSIZE        1024
STACKSIZE       8192

EXPORTS
            MainWndProc     @1
```

程序 **DEMO1_1.C**

```
/* ************************************************************* */
/*            Demo1_1  ---  The simple example                    */
/* ************************************************************* */

#include <windows.h>
#include "demo1_1.h"

int  PASCAL  WinMain(HANDLE, HANDLE, LPSTR, int);
long FAR PASCAL MainWndProc(HWND, unsigned, WORD, LONG);

```

```
11
12
13 /*  * * * * * * * * * * * * * *          * * * * * * * * * * * * *  */
14 /*                        WinMain                                   */
15 /*  * * * * * * * * * * * * * * * * * *  * * * * * * * * * * * * *  */
16
17 int PASCAL WinMain(HANDLE hInstance, HANDLE hPrevInstance,
18                     LPSTR lpszCmdLine, int nCmdShow)
19 {
20     WNDCLASS     wclass;
21     MSG          msg;
22     HWND         hWnd;
23     char         szName[] = "Demo1__1";
24
25     if (! hPrevInstance)
26      {
27          wclass.style          = CS__HREDRAW | CS__VREDRAW;
28          wclass.lpfnWndProc    = MainWndProc;
29          wclass.cbClsExtra     = 0;
30          wclass.cbWndExtra     = 0;
31          wclass.hInstance      = hInstance;
32          wclass.hIcon          = LoadIcon(NULL, IDI__APPLICATION);
33          wclass.hCursor        = LoadCursor(NULL, IDC__ARROW);
34          wclass.hbrBackground  = GetStockObject(WHITE__BRUSH);
35          wclass.lpszMenuName   = szName;
36          wclass.lpszClassName  = szName;
37
38          if (! RegisterClass (&wclass))
39              return (FALSE);
40      }
41
42      hWnd = CreateWindow(
43                  szName,
44                  "The simple example",
45                  WS__OVERLAPPEDWINDOW,
46                  CW__USEDEFAULT,
47                  CW__USEDEFAULT,
48                  CW__USEDEFAULT,
49                  CW__USEDEFAULT,
```

```
                    NULL,
                    NULL,
                    hInstance,
                    NULL );

    if (! hWnd)
        return (FALSE);

    ShowWindow(hWnd, nCmdShow);
    UpdateWindow(hWnd);

    while (GetMessage(&msg, NULL, NULL,NULL))
        {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }
    return (msg.wParam);
}



/* ********************************* */
/*               MainWndProc           */
/* ********************************* */

long FAR PASCAL MainWndProc(HWND hWnd, unsigned message,
                            WORD wParam, LONG lParam)
{
   HDC          hDC;
   HMENU        hMenu;
   static int   ShapeID = IDM_LINE;
   RECT         Rect;
   int          x1, x2, y1, y2;
   static int   CX, CY;

   switch (message)
    {
      case WM_CREATE :
                 srand(LOWORD(GetCurrentTime()));
```

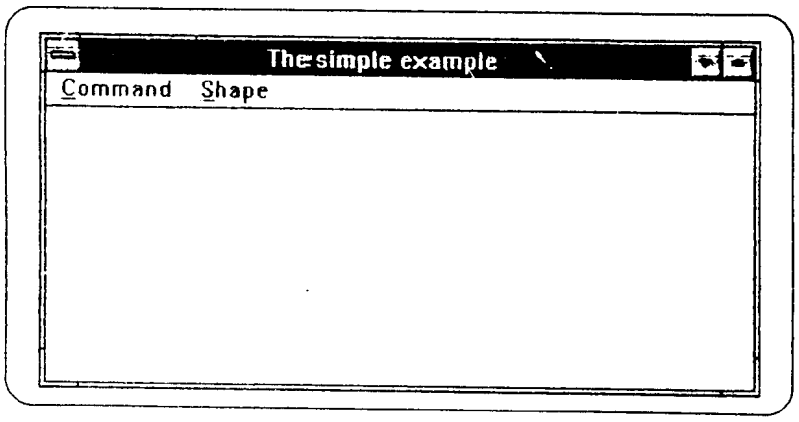
```
89                  break;
90
91          case WM_COMMAND :
92                  switch(wParam)
93                    {
94                      case IDM_CLEAR :
95                           InvalidateRect(hWnd, NULL, TRUE);
96                           UpdateWindow(hWnd);
97                           break;
98
99                      case IDM_EXIT :
100                          DestroyWindow (hWnd);
101                          break;
102
103                     case IDM_LINE :
104                          if (ShapeID ! = IDM_LINE)
105                            {
106                              hMenu = GetMenu(hWnd);
107                              CheckMenuItem(hMenu, ShapeID,
108                                                   MF_UNCHECKED);
109                              ShapeID = IDM_LINE;
110                              CheckMenuItem(hMenu, ShapeID,
111                                                   MF_CHECKED);
112                            }
113                          break;
114
115                     case IDM_RECT :
116                          if (ShapeID ! = IDM_RECT)
117                            {
118                              hMenu = GetMenu(hWnd);
119                              CheckMenuItem(hMenu, ShapeID,
120                                                   MF_UNCHECKED);
121                              ShapeID = IDM_RECT;
122                              CheckMenuItem(hMenu, ShapeID,
123                                                   MF_CHECKED);
124                            }
125                          break;
126
127                     case IDM_ELLIP :
```

```
128                     if (ShapeID ! = IDM__ELLIP)
129                       {
130                         hMenu = GetMenu(hWnd);
131                         CheckMenuItem(hMenu, ShapeID,
132                                       MF__UNCHECKED);
133                         ShapeID = IDM__ELLIP;
134                         CheckMenuItem(hMenu, ShapeID,
135                                             MF__CHECKED);
136                       }
137                     break;
138
139                 case IDM__DRAW :
140                     hDC = GetDC(hWnd);
141
142                     x1 = rand( ) % CX;
143                     y1 = rand( ) % CY;
144                     x2 = rand( ) % CX;
145                     y2 = rand( ) % CY;
146
147                     switch (ShapeID)
148                       {
149                         case IDM__LINE :
150                              MoveTo(hDC, x1, y1);
151                              LineTo(hDC, x2, y2);
152                              break;
153
154                         case IDM__RECT :
155                              Rectangle(hDC, x1, y1, x2, y2);
156                              break;
157
158                         case IDM__ELLIP :
159                              Ellipse(hDC, x1, y1, x2, y2);
160                              break;
161                       }
162
163                     ReleaseDC(hWnd, hDC);
164                     break;
165               }
166           break;
```

```
167
168         case WM_SIZE :
169                     GetClientRect(hWnd, &Rect);
170                     CX = Rect.right  - Rect.left;
171                     CY = Rect.bottom - Rect.top;
172                     break;
173
174         case WM_DESTROY :
175                     PostQuitMessage(0);
176                     break ;
177
178         default :
179          return (DefWindowProc(hWnd,message,wParam,lParam));
180       }
181     return (NULL);
182 }
```

程序虽然不算短，但就 Windows 应用程序来说，已算是很简单的了，如果读者有了 Windows 程序设计的基本常识及概念，那么应该很容易看懂。现在让我们看看这个程序。

demo1_1.h 内定义了 6 个在 demo1_1.rc 中用来作为菜单选项(menu item)的识别码(identifier code)的常数。demo1_1.rc 中只定义了一个简单的菜单，主菜单上有两个弹出式(Pop-up)菜单，Command 弹出式菜单下有 Draw，Clear 及 Exit 选项，Shape 弹出式菜单下则有 Line，Rectangle 及 Ellipse 选项，屏幕画面如下所示：



Shape 弹出式菜单是用来选择欲绘制的图形的形状，有线条(Line)，矩形(Rectangle)及椭圆形(Ellipse)可供选择。选择好后，再选择 Draw 选项，则程序就会以随机数产生不定大小的选择图形，绘制在以随机数产生的位置。若欲一直绘同一种图形，则只要一直选

择 Draw 选项即可。

WinMain( )函数的内容没什么特殊之处，在此不作说明，若有不清楚之处，也都是 Windows 程序设计入门必须学会的部分。

窗口函数 MainWndProc( )是用来接收 Windows 传送给程序的消息。在此处处理的消息包括:

□WM__CREATE:

在窗口建立之初调用 srand( )函数来起始一个随机数序列，并以 GetCurrentTime( )函数取得当前的时间来作为 sr and( )的种子(seed)值（开关 srand( )函数的说明请参见 C 语言函数库参考手册)。GetCurrentTime( )函数的语法如下所示:

```
DWORD GetCurrentTime( );
```

它会返回系统自启动至当前为止所经过的时间（以千分之一秒为单位)。

□WM__SIZE:

当窗口大小被改变时，窗口函数就会接收到一个 WM__SIZE 消息，此时我们调用 GetClientRect( )函数，其语法如下:

```
void GetClientRect(HWND hWnd,LPRECT lpRect);
```

它会取得工作区域的大小，并将矩形坐标存入 Rect 结构内。接着在第 170 列及 171 列设计出此工作区域的宽及高，并存入 CX 及 CY 变量内。这样在以随机数绘制图形时，才能将图形限制在工作区域之内。

事实上，当消息函数接收到 WM__size 消息时，工作区域的宽及高就已经存放在 lParam 参数内了，所以第 169 列至第 171 列的程序码可改为下面这两列:

```
CX = LOWORD(lParam);
CY = HIWORD(lParam);
```

□WM__DESTORY:

直接调用 PostQuitMeassge(0)结束程序。

□WM__COMMAND:

接收由用户选择的菜单选项之后传来的识别码。由于在 dem01__1.rc 中我们已见到定义了 6 个选项，所以在此分别处理这 6 个选项所应做的事:

1. IDM__CLEAR:

当用户选择了 Clear 选项，表示要清除工作区域内的图形，所以先调用 Invalid__ateRect( )函数将整个工作区域设置为失效矩形，然后调用 UpdateWindow( )函数送出 WM__PAINT 消息。关于失效矩形(invalid rectangle)及 WM__PAINT 消息的详细意义，在下两节中我们会详细地讨论到。

2. IDM__EXIT:

当用户选择了 Exit 选项，表示要结束程序，所以调用 DestroyWindow( )函数送出 WM__DESTROY 消息以结束程序。

3. IDM__LINE:

4. IDM__RECT:

5. IDM__ELLIP:

当用户选择了这三个选项时，只是表示先将绘制的图形种类设置好，事实上并未真正地画出，所以在第 80 列声明了一静态(Static)变量 ShapeID，用来记录选择好的选项识别码。由于这三个选项的处理方式类似，我们现在就以 IDM__LINE 为例来说明。

104 列：判断 ShapeID 是否已经是 IDM__LINE 了，若是，则不做任何处理。

106 列：取得菜单的识别码。

107 列：将 ShapeID 所代表的选项（即上一次所选择的选项）上的检查记号(CheckMark)取消。

109 列：将 ShpaeID 设置为 IDM__LINE。

110 列：在 ShapeID 所代表的选项（即 Line 选项）上加上检查记号。

6. IDM__DRAW:

在这儿我们初次见到了 GDI 的影子，第 140 列我们调用了：

```
hDC = GetDC(hWnd)
```

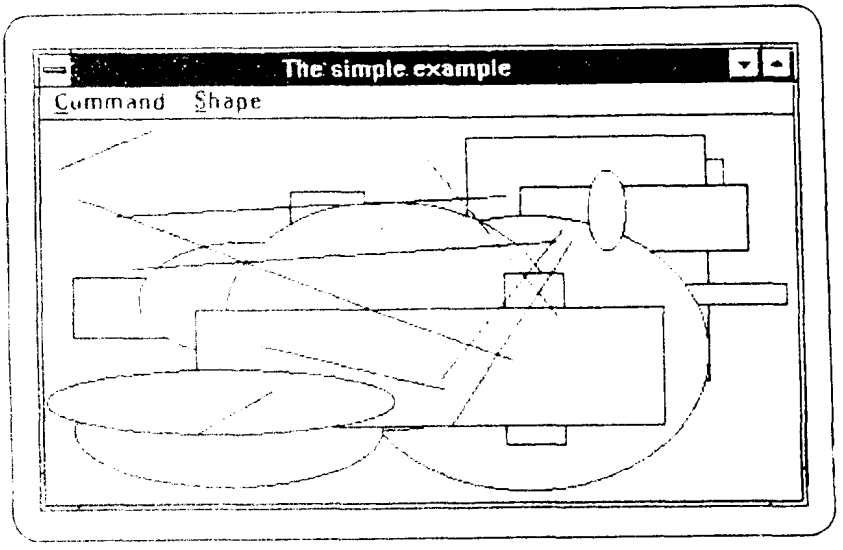
GetDC 函数会返回一个与 hWnd 窗口句柄相关的设备背景句柄(Device Context Handle)，设备背景对应一个实体的输出设备及其驱动程序。若要在工作区域内作画或显示文字，一定要先取得和窗口相关的设备背景句柄，因为几乎所有的绘图函数（包括显示文字的函数）都要以一设备背景句柄作为函数调用的第一个参数，这样该函数才知道要将图形输出在何处。

在使用完设备背景后，一定要将它释放掉，以留资源给其他的窗口使用。所以在第 163 列调用了：

```
ReleaseDC(hWnd,hDC);
```

在此我们不打算继续讨论设备背景，留到 1-4 节及下一章再来详加说明，现在让我们继续看看第 142 列至第 161 列的程序：

142～145 列：以随机数取得新图形的左上角坐标(x1,y1)及右下角坐标(x2,y2)。



147～161 列：根据 ShapeID 的数值来决定要画哪些图形．在此可见到各个绘图函数都以 hDC 为第一个参数值．在第三章我们会讨论绘制各种图形的函数，所以在此就先不说明 MoveTo( )， LineTo( )， Rectangle( )及 Ellipse( )函数的意义了．不过，熟悉 Pascal 或 Tubro C 语言的读者应该很快就可看懂，因为这两种语言都具有类似的同名函数．

解说完程序之后，乍看似乎无误，下面是执行了数次之后的执行结果：

但读者现在可试着将某应用程序的肖像移至本窗口的工作区域内，再将它移走；或是将本窗口缩小，拉大，便可发现工作区域内的图形消失了或被破坏了，这是为什么呢？

1-2 WM__PAINT 消息

在 1-1 节的一开始我们曾提及，当窗口被肖像，对话框等覆盖时，Windows 会尽力为我们挽救被覆盖的部分，但若无法挽救，则会送一个 WM__PAINT 消息给我们（窗口函数），窗口函数必须自行负责将它还原．

除此之外，在 demo1__1 中我们发现当用户改变窗口大小时，工作区域内的图形消失了，这是因为在一开始定义 wclass 窗口类别时我们设置了：

```
wclass.style = CS__HREDRAW | CS__VREDRAW;
```

若设置了 CS__HREDRAW 及 CS__VREDRAW，则当窗口的宽度及高度分别被改变时，便会以背景画刷来重画工作区域的内容，背景画刷(brush)就是一开始在定义 wclass 窗口类别时所设置的：

```
wclass.hbrBackground = GetStockObject(WHITE__BRUSH);
```

此列的意义是将背景画刷设置为系统固有的白色画刷（关于画刷的详细说明，我们会在第三章再来讨论）；一但窗口大小被改变时，便会送出 WM__SIZE 及 WM__PAINT 消息给窗口函数，然后在处理 WM__PAINT 消息的一开始（利用 BeginPaint( )函数）使用背景画刷来涂满整个工作区；于是窗口函数便可在接收到 WM__ SIZE 消息时猜知更改后工作区的宽及高度，并做一些设置．接着在接收到 WM__ PAINT 消息时，再进行重画的工作．

至此相信读者应该很清楚 WM__PAINT 消息的作用了吧！无论当工作区域的内容做了什么样的变动，Windows 都会送给窗口函数一个 WM__PAINT 消息，以告知视窗函数工作区域的内容被改变了，如有必要的话可自行负责重画的工作．

事实上，在一开始建立窗口之后所调用的 UpdateWindow( )函数便已送出了第一个 WM__PAINT 消息，用以告知窗口函数工作区域已准确妥当，可以开始作画了．从此之后，当以下的各种情形发生时，Windows 都会送给窗口函数一个 WM__PAINT 消息：

- ☐ 在窗口类的style域中设置了CS__HREDRAW或CS__VREDRAW，并改变了窗口的宽或高时．
- ☐ 窗口被缩为一个肖像之后，又被放大为具有工作区域的窗口时．
- ☐ 被别的窗口或肖像覆盖掉的区域又重新显示出来时．