北京科海培训中心系列教材

C++与 Windows 高级编程

张 剑 编著

清华大学出版社

(京)新登字 158 号

【内容提要】 本书是C语言程序员或DOS程序员进入C++与Windows领域的提高读物。书中以几个极有针对性的专题，介绍了当前程序员必须掌握的C++与Windows程序设计中的关键技术——“从DOS轻松转到Windows”，“深入C++内部”，“Windows编程导引”，“用Borland C++的类库进行编程”，“使用Visual C++的类库编程”，“在面向对象的时代管理对象”，“充分利用已有资源”；特别富有吸引力的是最后两章，在同类图书中本书首次深入介绍了当今最为迫切的Windows中的汉字技术的内幕——“在Windows中使用汉字”，“彻底汉化Windows”。

本书深入透出，实例丰富，适于广大电脑爱好者及程序设计人员阅读。

欲购书中程序磁盘，可与清华大学出版社软件部或科海培训中心门市部联系(电话：2594891，2589259)。

图书在版编目(CIP)数据

C++与Windows高级编程/张剑编著. —北京：清华大学出版社，1994.9

ISBN 7-302-01666-6

I. C… II. 张… III. ①C语言—程序设计②窗口软件—操作系统—程序设计 IV. TP①312C ②TP316

中国版本图书馆CIP数据核字(94)第13309号

书名：C++与Windows高级编程

作者：张剑

出版：清华大学出版社(北京·清华园，100084)

印刷：北京市密云县印刷厂

发行：新华书店总店北京科技发行所

开本：787×1092 1/16

印张：16

字数：417千字

版次：1994年11月第1版　　1994年11月第1次印刷

书号：ISBN 7-302-01666-6/TP.715

印数：00001～10000

定价：20.00元

目 录

第一章　从 DOS 轻松转到 Windows

本章分成三部分。

1. 什么是 EasyWin，介绍 EasyWin 的概念。从一个 EasyWin 的 DOS 风格程序介绍 DOS 和 Windows 的不同，作为对 DOS 时代的简单介绍。
2. 介绍如何用 EasyWin 作为你进步的阶梯。如何从 Windows 中获得各种句柄参数，添加菜单，处理消息。这些高级技术你可以轻松地应用于其它方面。
3. 程序的调试是比较重要的一方面，一般比较理想的方法是用两个屏幕来调试，一个用作操作，另一个用来显示各种调试信息，显示程序运行中各种变量的值。这部分介绍的如何使用 EasyWin。从而可以只用一个屏幕达到同样的效果。

1.1　在 Windows 下运行 DOS 程序

1.1.1　Windows 与 DOS 应用程序的比较

一个程序员刚转到 Windows 下时，会对 Windows 的消息驱动风格感到难以掌握。在 DOS 下编程序时，一个程序设计者完全决定了什么时候接收用户的输入，当等待用户输入时，显示一串信息，然后等待，用户输完后，程序接着向下执行。当程序要显示信息时，则向屏幕发送一字符串，根本不必管字符在屏幕上的显示位置。程序的执行基本上是顺序执行的。而在 Windows 下，程序接收外部的消息不再是在自己想接收的时候，程序可以在任意的时候接到任意的消息，这样，就要求程序要保留自己执行状态的信息，以便在接到下一条消息时知道自己处在程序的什么地方。

从 DOS 的简单输入输出接口、顺序执行的程序方式过渡到 Windows 下的图形用户接口、消息驱动方式，实际上也反映了计算机科学的发展和应用范围的扩大。在计算机应用的早期，计算机的主要用途是数值计算机，用户接口用于输入用户要计算的数，输出计算后的结果，这样简单输入输出接口即可满足要求。顺序执行的程序设计方式也恰好反映了数值计算的程序设计方法。但随着计算机的发展，计算机的应用范围不断扩大，计算机的应用逐渐扩大到事务处理等一些非数值计算领域，在这些领域中，用户与计算机的操作方式是交互式操作方式，计算机在处理过程中要不断与用户打交道，根据使用者的指令去执行，在这种与用户的交互作用日益重要的情况下，必然要过渡到消息驱动方式。这也是时代发展的趋势。

1.1.2　使用 EasyWin

即使在 Windows 下，有时也要编一些相对简单的程序，使用 DOS 编程的一般方法即可。如希望它能以 Windows 程序的方式运行，换句话说，就是能尽量简便地将 DOS 程序移植到 Windows 下，使用 EasyWin 可以简单地做到这一点。

EasyWin 是 Borland 公司推出的 Borland C++中用来将 DOS 程序移植成 Windows 程序的一个工具。编本书时用的是 Borland C++ 3.1。在 BC++3.1 的 Borland C++ for

Windows 程序开发环境下(简称 BCW)只要打开一文件,按 DOS 下 C++程序的一般编写方法去编写,然后选菜单中的 Run 命令即可。BCW 在链接程序时,如果没有找到 Windows 应用程序特有的入口点 WinMain()函数,便会寻找 main()函数并连接 EasyWin 库。这样,几乎一行专用于 Windows 的代码都不必编写,就可以得到一个可在 Windows 下运行的程序。这样,在编一些普通的 Winodws 程序时,可以很容易地用 EasyWin 达到目的。

值得注意的是,EasyWin 只能移植使用字符接口的程序。如果你的 DOS 程序涉及直接操作硬件,使用 DOS 的图形模式,那么,你的程序是不可移植的。因为在 Windows 这些部分是必须由操作系统控制的部分,你的程序与它们直接打交道将和操作系统相冲突,导致程序不可再运行。所以,如果你的程序中有上面所说的成分,如果想移植到 Windows 下,那么,就应考虑编写一正规的 Windows 函数代替 DOS 下图形库中的函数。

1.2 实例分析

下面,我们仔细看一个标准的 EasyWin 程序,作为对 DOS 的一个回顾,了解 EasyWin 能干什么。

程序的名称是 DOSSTYLE.CPP,在 Borland C++的 BCW 环境下,选择 New,创建一新文件,将程序输入,然后选择 Run 命令,即可以 Windows 应用程序的方式运行。在运行完毕后,按标准 Windows 程序关闭方法打开系统菜单,然后按 Close 命令关闭。

应用程序代码

```
//filename: dosstyle.cpp, demo dos io.
// author TSINGHUA ZHANGJIAN

#include <iostream.h>
#include <conio.h>
main()
{
    cout<<"This program accept two number"<<endl;
    cout<<"Please Input first number:";
    float m,n;
    cin>>m;
    cout<<"\nPlease Input sceond number:";
    cin>>n;

    clrscr();

    cout<<m<<" add "<<n<<" Get "<<m+n<<endl;
    cout<<m<<" multiple "<<n<<" Get "<<m*n<<endl;

    cout<<"now will delete the first line,press any key start."<<endl;

    getch();

    gotoxy(1,1);
    clreol();

    cout<<"now will clean screen,press any key start."<<endl;
    getch();

    clrscr();
```

```
    cout<<"This is a demo of Dos like screen control style."<<endl;
    gotoxy(8,2);
    cout<<"Author TSINGHUA ZHANGJIAN,1/16,1994"<<endl;
    return 0;
}
//file dosstyle is end;
```

程序的前两行是预处理包含文件命令，包含了 iostream. h 和 conio. h 两个标准头文件。包含 iostream. h 是因为我们使用了 C++中的流，利用流的插入算符"<<"和提取算符">>"来进行输入和输出。conio. h 用来进行控制屏幕输出的一些特性，对屏幕输出进行简单的控制。

流的介绍

在这里值得说一说的是 C++的流，使用 C++中的流进行输入和输出比用标准 C 语言库中的输入输出函数 printf()和 scanf()要方便得多。这是因为在 C++中进行了函数重载，C++语言能够自动根据参数的不同类型而去调用不同的函数。对一个通常的算符调用，比如：

```
int i,j ;
cout<<i<<j<<endL;
```

C++语言实际上将调用语句转换为如下的函数调用(假设"<<"算符是 C++流的成员函数)((cout. <<(i)).<<(j))；由于"<<"的返回值仍然是输出流，因此可以接着进行输出调用，如果"<<"算符是输出流的友元，那么表达式可以被转换成如下形式：

```
<<(<<(cout,i)),j)
```

自定义流操作符

使用 C++的流的优点是不必去记忆繁杂的操纵控制符，在程序设计中，不必去翻看语言手册，即可直接写出输出表达式。

流的另一个特点是你可以写出你自己的类的输出表达式，而仅需同样调用 C++中的输出算符，下面看一个例子。

```
Class TMyClass {
        public:int i,j;
              TMyClass(int l,int m) { i=l; j=m}
              friend class ostream;}
Ostream perator <<(ostream &os,TMyClass cl)
    {
        Os<<"\n,now out class TMyClass,"<<cl. i<<" "<<cl. j<<endl;
            return os;
    }
  TMyClass AClass(3,5);
Cout<<AClass;
```

那么，运行上面程序的结果将是如下：

```
now outClass TMyClass 3 5
```

流的优点

从上面和程序中可以明显地看到流的优点。对于一个你自定义的类，你可以用形式统一的流输出算符来操纵它，只要你已重载了运算符。使用形式统一的流输出算符来进行输入输出的好处是巨大的。首先形式方便，流输出都有统一的格式。另外，当你希望改变输出形式时，只要修改重载函数的定义即可。

利用 conio.h 中提供的函数可以对屏幕输出进行简单的控制。可以进行清屏，这将导致光标移到屏幕左上角。在字符型终端上进行输入输出时，有一个光标的概念。在字符型应用程序中，当你敲入要输入的内容时，你输入的内容也同时在屏幕上显示出来，称之为回显。所谓光标就是决定下一个待显示字符位置的一种闪烁标记。你可以通过控制光标的位置来达到控制下一个字符输出位置的功能。在正宗的 Windows 程序中，由于鼠标的指示已被称为光标，所以对于字符型终端的这种概念在 Windows 中称为插入符号。

程序外观

在 DOSSTYLE.CPP 的 main()函数中，开始显示一提示信息，请你输入两个数。然后程序利用 C++中的流操作符来接收你输入的两个数。这里，你再次看到了 C++流的方便与易用性。

为了显示普通 DOS 类程序的特点，在取得了两个数后，程序首先应用 ClrScr()函数来清屏。调用此函数将清除屏幕上的显示内容，将光标定位于屏幕左上角。这是说，这时，下一个显示的字符，除非你特别控制，将显示在屏幕的左上角。

这时，程序计算出你输入的两个数的和与积，分别显示出来。

这个程序的主要目的是为了演示 DOS 风格程序的特点，所以，这时程序请你按下任意一键以便清除屏幕显示的第一行。利用 getch()来等待用户按下任意一键以便进行下一个操作是 DOS 程序设计中常用的方法。getch()用来取出一个用户输入的字符，该函数将等待，直至用户按下一键时才返回，所以只有当用户按下一键后该函数才会返回。

光标与屏幕坐标系

你按下一键后，程序接着往下执行。程序首先调用 gotoxy()来将光标移到第一行第一个字符的位置，这使该位置成为当前显示位置。然后调用 clreol()从当前位置开始，清掉该行的内容。在这里，值得说一说的是字符显示中的坐标。计算机显示中，屏幕显示坐标的规定和人看书时的习惯是一致的。即从左至右、从上到下。所以，屏幕上 x 坐标增长方向是从左至右，而 y 轴的方向是从上到下。而在普通的迪卡尔坐标系中，y 轴方向是从下至上增长的。这是值得注意的地方。在通常用的屏幕坐标中，屏幕的原点在屏幕的左上角。一般将该点定为(1,1)，即 x 方向和 y 方向的坐标都为 1 的地方。一般屏幕是 40 列，25 行，即 x 的变化范围是从 1 至 40，y 轴是 1 到 25。

程序在清掉第一行后，在该行上显示一提示信息，告诉你，按下任意一键后将清屏，你按下任意一键后，屏幕清屏后显示一行提示信息，然后在第二行的中间显示版权信息。

这个程序在运行完毕返回后，程序的窗口将加上一个前缀“inActive”，意思是该程序已从 main()主函数返回。在此之前，EasyWin 程序的窗口边框不可以被缩放。变为不活动后，则可改变大小。

如果这个程序在 Borland C++的 DOS 环境下编译，则生成一个 DOS 程序。建议你这样做，以便比较 DOS 环境和 EasyWin 环境的异同。

通过上面这个程序，你应该对 DOS 下屏幕控制有较多的了解。在 DOS 下，你可以在屏幕的任一点开始显示，这使用函数 gotoxy()将光标移到那一点即可。也可以将屏幕上的显示内容清掉，清掉一行或整屏都可以。使用 ClrScr()和 Clreol()这两个函数即可达到目的。

1.3　Windows 编程分析

1.3.1　Windows 的尴尬

当你进入 Windows 进行编程时，感到惊叹的不仅是近七百个函数(普通 C 语言库只有一百多个函数)，而且对调用这些函数时所需的参数个数感到惊奇，调一个函数，经常要传递四、五个参数，有的甚至十几个。令人难以记忆，难以掌握。所以，几乎所有的 Windows 程序员都要在桌面上摆一本程序员参考手册，当在计算机上编制程序时，还要随时查看联机帮助文件，以便寻找某个函数的具体用法。有时候，当你对某个 Windows 函数的特性不十分了解时，希望先编一个小程序测试它一下。如果是在 DOS 下，只要包含有这个函数定义的头文件，然后在 main()函数中直接调用该函数即可。代码一般不超过十行。这是掌握函数用法，提高自己水平的很有效的方法。但是，在 Windows 下，即使编一个什么功能都不实现，只具有一个可以关闭的简单窗口的程序，其代码恐怕也已超出了四五十行。有谁会愿意为试验一个函数而去对着屏幕敲四五十行源代码呢？这恐怕只会使人对 Windows 编程兴趣索然。

1.3.2　Windows 的事件驱动与函数调用特点

那么，为什么会造成这种状况呢？这是因为，Windows 是一个消息驱动式的程序设计环境，你编的程序必须具有 Windows 程序所共有的那些部分。同时，很多 Windows 函数都要有一个句柄参数，以便操作系统对此函数的行为进行控制。比如，对窗口操作的函数都要有一个窗口句柄作为头一个参数，以便操作系统知道是对哪个窗口进行操作，避免对其它窗口的操作，以便实现多任务。几乎所有的 GDI 函数都有一个 HDC 句柄参数，以便 Windows 利用此句柄对该函数的行为加以控制，比如，剪裁，以免绘制操作影响到别的区域等。在 Windows 中，窗口句柄、实例句柄都是在程序创建过程中得到的，似乎只有按正规的 Windows 程序来设计，才能获得这些句柄，并把这些句柄参数用在各种函数调用中。

那么，难道就没有什么像 DOS 下实验函数那样方便的方法吗？希望首先获得各种句柄参数，在数十行代码内，然后用这些句柄来实验函数。

这是能做到的，答案就是利用 EasyWin。

EasyWin 中已为你做好了消息处理机构，Borland C++在连接你的程序时，如果没有找到 WinMain()入口，则会寻找 main()入口并自动连接 EasyWin 库。那么，在 EasyWin 中如何获得各项句柄呢？

1.3.3　获得 Windows 中的句柄

Windows 是一个非占先式的操作系统，意思是说，只有应用程序主动让出执行权，程序才有可能切换到另一个程序。而应用程序是以窗口为基础的。一个应用程序拥有一个主窗

口，用户和此窗口进行交互操作，操作系统向此窗口发送各种消息。在操作系统中，保存有一个窗口句柄表，其中包含有各个窗口句柄、每个窗口句柄实际上是操作系统数据段中的地址。窗口的数据包括实例句柄，程序窗口函数处理地址等各种数据。可见，只要有了窗口句柄，其它各种参数就都能获得。Windows 保存当前活动窗口的窗口句柄，这样，我们就可以获得活动窗口句柄。

当一个 EasyWin 程序开始执行时，它所创建的窗口成为当前活动窗口，因为这个窗口接收键盘输入等消息。这时，如果用 Windows 函数 GetActiveWindow()将获得当前活动窗口句柄，这个窗口活动句柄就是应用程序产生的 EasyWin 窗口，这样，我们所有的问题就都迎刃而解。

1.4 Windows 与 DOS 混合的应用程序

我们来研究下面这个 demogdi.cpp 程序。这个程序有一个菜单，你可以在窗口上产生一个钟表，一个沿窗口四个边沿碰撞的小球，以及用随机颜色在窗口的随机区域位置上画矩形方块。你也许感到奇怪，菜单是怎么加上的呢？Windows 是基于消息的操作系统，你自己的消息处理函数是怎么加上的呢？别急，让我们一步一步来看。这个程序虽然很不完善，但它展示了 Windows 编程中的各种技巧，值得研究。

程序代码

```
//filename: get.h,get the EasyWin handle;
// author TSINGHUA ZHANGJIAN
#include <windows.h>
class TGet{
    public:
        TGet();
        HWND hWnd;
        HINSTANCE hInst;
};
inline TGet::TGet()
{
    hWnd=GetActiveWindow();
    if(IsWindow(hWnd))
        hInst=GetWindowWord(hWnd,GWW_HINSTANCE);
    else hInst=0;
}
//define class TDC,get the window DC,
class TDC{
    public:
        TDC(HWND hWnd){
            this->hWnd=hWnd;
            hDC=GetDC(hWnd);
            }
        TDC(TGet AGet){
            hWnd=AGet.hWnd;
            hDC=GetDC(hWnd);
```

```
        }
    ~TDC(){
        ReleaseDC(hWnd,hDC);
    }
    HWND hWnd;
    HDC hDC;
};
//file get.h is end;
//////////////////////
//file name:menu.h;use menu in EasyWin;
//author TSINGHUA ZHANGJIAN
#include <windows.h>
class TMenu{
  public:
    TMenu(){
        hMenu=CreateMenu();
        hPopup=CreatePopupMenu();
        AppendMenu(hMenu,MF_POPUP|MF_STRING,hPopup,"&Menu");
        }
    Add(int nID,char * string){
        AppendMenu(hPopup,MF_STRING,nID,string);
        AppendMenu(hPopup,MF_SEPARATOR,0,0);          }
    Check(int nID,BOOL MF){
        CheckMenuItem(hPopup,nID,MF? MF_CHECKED:MF_UNCHECKED
);
        }
    HMENU hMenu,hPopup;
};
//file menu is end;
//filename demogdi.cpp,used demo gdi and message control in EasyWin;
//author TSINGHUA ZHANGJIAN
#include <windows.h>
#include <iostream.h>
#include <stdlib.h>
#include <math.h>
#include "get.h"
#include "menu.h"
#define IDM_RANDOM   10
#define IDM_CLOCK    20
#define IDM_BALL     30
#define IDM_CLEAN    32
//timer ID define
#define IDT_RANDOM   35
#define IDT_CLOCK    38
#define IDT_BALL     40
#define TIME         1000
#define TIMERANDOM   1000
#define TIMECLOCK    1000
```

```
#define TIMEBALL        100

FARPROC lpfnOldProc;
void BeginRandom(HWND hWnd);
void EndRandom(HWND hWnd);
void BeginClock(HWND hWnd);
void EndClock(HWND hWnd);
void BeginBall(HWND hWnd);
void EndBall(HWND hWnd);
void DoRandom(HWND hWnd);
void DoClock(HWND hWnd);
void DoBall(HWND hWnd);

TMenu Menu;

LRESULT WINAPI_export DealCommand(HWND hWnd,WORD Message,WPARAM
wP,LPARAM lP)
{
    switch(Message){
        case WM_COMMAND:{

        switch(wP){

        case IDM_RANDOM:
            if(LOBYTE(GetMenuState(Menu.hMenu,IDM_RANDOM,MF_BYCOMMAND))
              &MF_CHECKED)
                {
                Menu.Check(IDM_RANDOM,FALSE);
                EndRandom( hWnd);
                }
             else {
                Menu.Check(IDM_RANDOM,TRUE);
                BeginRandom( hWnd);
                }
             DrawMenuBar(hWnd);
             break;
        case IDM_CLOCK:
            if(LOBYTE(GetMenuState(Menu.hMenu,IDM_CLOCK,MF_BYCOMMAND))
              &MF_CHECKED)
                {
                Menu.Check(IDM_CLOCK,FALSE);
                EndClock( hWnd);
                }
             else {
                Menu.Check(IDM_CLOCK,TRUE);
                BeginClock( hWnd);
                }
             DrawMenuBar(hWnd);
             break;
        case IDM_BALL:
                if(LOBYTE(GetMenuState(Menu.hMenu,IDM_BALL,MF_BYCOMMAND))
                  &MF_CHECKED)
                {
                Menu.Check(IDM_BALL,FALSE);
```

```
                EndBall( hWnd);
                }
            else {
                Menu.Check(IDM_BALL,TRUE);
                BeginBall( hWnd);
                }
            DrawMenuBar(hWnd);
            break;
        case IDM_CLEAN:{
            if(IsWindow(hWnd)){
              TDC DC(hWnd);
              RECT rt;
              GetClientRect(hWnd,&rt);
FillRect(DC.hDC,&rt,GetClassWord(hWnd,GCW_HBRBACKGROUND));
              }
              }
              break;
        default: return CallWindowProc(lpfnOldProc,hWnd,Message,wP,lP);
        }
      }
        break;
    case WM_TIMER:
    if(wP==IDT_RANDOM){
if(LOBYTE(GetMenuState(Menu.hMenu,IDM_RANDOM,MF_BYCOMMAND))
  &MF_CHECKED)
            DoRandom( hWnd);
    }
    else if(wP==IDT_CLOCK){
    if(LOBYTE(GetMenuState(Menu.hMenu,IDM_CLOCK,MF_BYCOMMAND))
     &MF_CHECKED)
            DoClock( hWnd);
            }
        else if(wP==IDT_BALL){
       if(LOBYTE(GetMenuState(Menu.hMenu,IDM_BALL,MF_BYCOMMAND))
         &MF_CHECKED)
           DoBall( hWnd);
       }
    break;
    case WM_DESTROY:
        if(lpfnOldProc)
            SetWindowLong(hWnd,GWL_WNDPROC,(long)lpfnOldProc);
    default: return CallWindowProc(lpfnOldProc,hWnd,Message,wP,lP);
    }
}
main()
{    cout<<"This is EasyWin gdi demo"<<endl;
     TGet AGet;
     Menu.Add(IDM_RANDOM,"&Random");
```

```
        Menu.Add(IDM_CLOCK,"&Clock");
        Menu.Add(IDM_BALL,"&Ball");
        Menu.Add(IDM_CLEAN,"C&lean Screen");
        SetMenu(AGet.hWnd,Menu.hMenu);
        lpfnOldProc=(FARPROC)SetWindowLong(AGet.hWnd,GWL_WNDPROC,(long)Deal
Command);
        return 0;
}
void BeginRandom(HWND hWnd)
{
        if(IsWindow(hWnd)){
        SetTimer(hWnd,IDT_RANDOM ,TIMERANDOM,NULL);
        randomize();
        }
}
void EndRandom(HWND hWnd)
{       if(IsWindow(hWnd)){
        KillTimer(hWnd,IDT_RANDOM);
        }
}
void BeginClock(HWND hWnd)
{
        if(IsWindow(hWnd)){
        SetTimer(hWnd,IDT_CLOCK ,TIMECLOCK,NULL);
        randomize();
        }
}
void EndClock(HWND hWnd)
{       if(IsWindow(hWnd)){
        KillTimer(hWnd,IDT_CLOCK);
        }
}
void BeginBall(HWND hWnd)
{
        if(IsWindow(hWnd)){
        SetTimer(hWnd,IDT_BALL ,TIMEBALL,NULL);
        randomize();
        }
}
void EndBall(HWND hWnd)
{
        if(IsWindow(hWnd)){
      KillTimer(hWnd,IDT_BALL);
      }
}
void DoRandom(HWND hWnd)
{
      RECT rt;
          GetClientRect(hWnd,&rt);
      rt.right=random(rt.right);
```

```
    rt.bottom=random(rt.bottom);
    char r,g,b;
    r=random(255);
    g=random(255);
    b=random(255);
    COLORREF c=RGB(r,g,b);
    if(IsWindow(hWnd)){
    TDC DC(hWnd);
    HBRUSH old=SelectObject(DC.hDC,CreateSolidBrush(c));
    Rectangle(DC.hDC,rt.right-10,rt.bottom-10,rt.right+10,rt.bottom+10);
    DeleteObject(SelectObject(DC.hDC,old));
      }
}
#define direction 60
#define step          20
void DoBall(HWND hWnd)
{
    static int x,y,x0,y0;//store last position.
    static BOOL xd,yd,Init;
    static RECT rt;
    if(IsWindow(hWnd)){
        TDC DC(hWnd);
        SetROP2(DC.hDC,R2_NOTXORPEN);
        if(! Init) {
        GetClientRect(hWnd,&rt);
        x0=step * cos(M_PI * direction/180.0);
        y0=step * sin(M_PI * direction/180.0);
        xd=yd=TRUE;
          Init=TRUE;//indicate initiate have done;
              }
        if(Init){
        HBRUSH old=SelectObject(DC.hDC,GetStockObject(BLACK_BRUSH));
        Ellipse(DC.hDC,x-10,y-10,x+10,y+10);
        SelectObject(DC.hDC,old);
        }
        if(xd){
        x+=x0;
        if(x>rt.right)xd=! xd;
        }
        else{
            x-=x0;
        if(x<0)xd=! xd;
        }
        if(yd){
            y+=y0;
            if(y>rt.bottom)yd=! yd;
        }
        else {
            y-=y0;
```

```
            if(y<0)yd=! yd;
          }
        //have justified position;
        HBRUSH old=SelectObject(DC.hDC,GetStockObject(BLACK_BRUSH));
        Ellipse(DC.hDC,x-10,y-10,x+10,y+10);
        SelectObject(DC.hDC,old);
    }
}
void DoClock(HWND hWnd)
{
    static int nLastAngle,x,y;
    static int r0;
    static RECT rt;
    if((nLastAngle+=10)>=360)nLastAngle=0;
    if(IsWindow(hWnd)){
        TDC DC(hWnd);
        SetROP2(DC.hDC,R2_NOTXORPEN);
        if(! x&&! y) {
        GetClientRect(hWnd,&rt);
            r0= (rt.right<rt.bottom? rt.right:rt.bottom);
            r0/=2;
                }
            MoveTo(DC.hDC,rt.right/2,rt.bottom/2);
            if(x||y)LineTo(DC.hDC,x+rt.right/2,y+rt.bottom/2);
        x=r0*cos(M_PI*nLastAngle/180.0);
        y=r0*sin(M_PI*nLastAngle/180.0);
        MoveTo(DC.hDC,rt.right/2,rt.bottom/2);
        LineTo(DC.hDC,x+rt.right/2,y+rt.bottom/2);
    }
}
//file demogdi is end;
```

代码分析

文件的开头包含了四个系统文件，其中包含文件 math.h 是因为为了计算时钟秒钟的屏幕位置而使用了正弦和余弦函数。

接下来，文件包含了自定义的一个文件头 get.h。它是用来获得 EasyWin 的活动窗口的。在类 TGet 的构造函数中，利用 GetActiveWindow()函数获得了当前活动窗口，也就是我们程序的窗口，把它放在一个公用数据成员 hWnd，这样，当你一旦生成一个 TGet 类对象，也就同时获得了当前的窗口句柄。你应该还记得，当生成一个类的实例，即一个该类的对象时，C++编译器将隐含地为这个对象调用构造函数。从而，一些必要的初始化代码只要放在构造函数中，就可以完成初始化，使代码简洁，避免遗忘。

取得句柄

类 TGet 的另一个数据成员是该窗口的实例句柄 hInst，一个窗口的 hInst 实际上就是