1. 窗口创建步骤

1.1 定义程序入口

WinMain是一个函数，该函数的功能是被系统调用，作为一个**32**位应用程序的入口点。**WinMain**函数应初始化应用程序，显示主窗口，进入一个消息接收一发送循环，这个循环是应用程序执行的其余部分的顶级控制结构。

```
int APIENTRY WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow);

WinMain函数的参数是系统传递的,不用我们操心.
```

1.2 设计窗口类

```
typedef struct tagWNDCLASS {
    UINT        style;          // 窗口类风格
    WNDPROC     lpfnWndProc;    // 窗口过程函数指针
    int         cbClsExtra;     // 窗口类附加数据缓冲区字节数
    int         cbWndExtra;     // 窗口附加数据缓冲区字节数
    HINSTANCE   hInstance;      // 当前应用程序实例句柄
    HICON       hIcon;          // 图标句柄
    HCURSOR     hCursor;        // 光标句柄
    HBRUSH      hbrBackground;  // 刷子句柄
    LPCSTR      lpszMenuName;   // 菜单资源名
    LPCSTR      lpszClassName;  // 窗口类名
} WNDCLASS, *PWNDCLASS, NEAR *NPWNDCLASS, FAR *LPWNDCLASS;

注意: WNDCLASSEX中多了一个 UINT cbSize;  // 结构体字节数*
```

1.3 注册窗口类

```
ATOM RegisterClass (CONST WNDCLASS* lpWndClass);
ATOM即unsigned short，成功返回所注册窗口类的唯一标识码(非0)， 失败返回0。
```

1.4 创建窗口

```
HWND CreateWindow (
    LPCTSTR    lpClassName,  // 窗口类名
    LPCTSTR    lpWindowName, // 窗口标题栏信息
    DWORD      dwStyle,      // 窗口风格
    int          x,          // 窗口左上角水平坐标
    int          y,          // 窗口左上角垂直坐标
    int          nWidth,     // 窗口宽度
    int          nHeight,    // 窗口高度
    HWND       hWndParent,   // 父窗口句柄
    HMENU      hMenu,        // 菜单句柄
    HINSTANCE  hInstance,    // 当前应用程序实例句柄
    LPVOID     lpParam       // 附加数据
 )
```

1.5 显示更新窗口

```
BOOL ShowWindow ( HWND hWnd, int  nCmdShow);
BOOL UpdateWindow( HWND hWnd);
```

1.6 消息循环

```
BOOL GetMessage (   //收到WM_QUIT返回FALSE，收到其它消息返回TRUE
    LPMSG lpMsg,         // 消息结构
    HWND  hWnd,          // 窗口句柄
    UINT  wMsgFilterMin, // 起始消息
    UINT  wMsgFilterMax  // 终止消息
);

将虚键消息翻译为字符消息。
若消息被翻译则返回TRUE，否则返回FALSE。
BOOL TranslateMessage (const MSG* lpMsg // 消息结构 );

将消息派发到窗口过程函数。
返回窗口过程函数的返回值
LRESULT DispatchMessage (const MSG* lpmsg // 消息结构 );
```

1.7 返回

```
return 0;
return (int)msg.wParam;

返回0,表示没有错误,最好是返回消息结构体内的wParam;
```

1.8 完整代码

```
#include <Windows.h>
#include <assert.h>


LRESULT CALLBACK MyWndProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam);

int WINAPI WinMain(_In_ HINSTANCE hInstance, _In_opt_ HINSTANCE hPrevInstance, _In_ LPSTR lpCmdLine, _In_ int
nShowCmd)
{
    //0.Design window class
    WNDCLASS wc;
    wc.cbClsExtra = 0;
    wc.cbWndExtra = 0;
    wc.hbrBackground = (HBRUSH)GetStockObject(LTGRAY_BRUSH);
    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
    wc.hInstance = hInstance;
    wc.lpfnWndProc = (WNDPROC)MyWndProc;
    wc.lpszClassName = L"CN_CLASSNAME";

    wc.lpszMenuName = 0;
```

```
    wc.style = 0;

    //1.Register window class
    bool bRet = GetClassInfoEx(hInstance, wc.lpszClassName, &wc);
    if (!bRet)
    {
        bRet = RegisterClass(&wc);
        assert(bRet);
    }

    //2.create window
    HWND hWnd = CreateWindow(L"CN_CLASSNAME", L"WN_WINDOWNAEM", WS_OVERLAPPEDWINDOW, 720, 480, CW_USEDEFAULT,
CW_USEDEFAULT, 0, 0, hInstance, 0);
    assert(INVALID_HANDLE_VALUE != hWnd);

    //3.show update
    ShowWindow(hWnd, nShowCmd);
    UpdateWindow(hWnd);

    //4.Msg loop
    MSG msg;
    while (GetMessage(&msg, 0, 0, 0))
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }

    //5.return
    return msg.wParam;
}

LRESULT CALLBACK MyWndProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
    PAINTSTRUCT ps;
    HDC hdc;

    switch (uMsg)
    {
    case WM_PAINT:
        hdc = BeginPaint(hWnd, &ps);
        Ellipse(hdc, 0, 0, 200, 100);
        EndPaint(hWnd, &ps);
        return 0;
    case WM_DESTROY:
        PostQuitMessage(0);
        return 0;
    }

    return DefWindowProc(hWnd, uMsg, wParam, lParam);
}
```

2. 回调函数

回调函数

回调函数就是一个通过函数指针调用的函数。如果你把函数的指针（地址）作为参数传递给另一个函数，当这个指针被用来调用其所指向的函数时，我们就说这是回调函数。回调函数不是由该函数的实现方直接调用，而是在特定的事件或条件发生时由另外的一方调用的，用于对该事件或条件进行响应。

机制

- 定义一个回调函数
- 提供函数实现的一方在初始化的时候,将回调函数的函数指针注册给调用者
- 当特定的事件或条件发生的时候,调用者使用函数指针调用回调函数对事件进行处理

例子 设计窗口类有成员变量WNDPROC WNDCLASS::lpfnWndProc,它就是一个回调函数的指针;我们需要写一个同类型函数(消息处理函数),赋值给lpfnWndProc;当有应用消息时,系统会调用这个函数;

消息处理结果

一个消息一定要有结果,一般是0(表示处理了).

3. 基本概念汇总

3.1 句柄

在windows应用程序中，窗口都是通过句柄进行标识的， 当系统创建这些资源的时候会分配内存，并返回资源标志号–句柄

3.2 加载图标

```
HANDLE LoadImage (
    HINSTANCE   hinst,      //实例句柄
    LPCTSTR     lpszName,   //文件名
    UINT        uType,      //图片类型
    int         cxDesired,  //指定图标或光标的宽度
    int         cyDesired,  //高度
    UINT        fuLoad,     //加载方式
);

uType:
IMAGE_BITMAP: 装载位图;
IMAGE_CURSOR: 装载光标;
IMAGE_ICON: 装载图标。

fuLoad:
LR_DEFAULTSIZE ——默认大小
LR_DEFAULTCOLOR: 缺省标志；它不作任何事情
LR_LOADFROMFILE——从文件中加载
```

3.3 得到 & 设置 窗口类信息

```
ULONG_PTR WINAPI GetClassLongPtr(
    _In_ HWND hWnd,     //窗口的句柄，间接指向窗口所属的类。
    _In_ int  nIndex
);
返回值:
如果函数成功，返回值就是请求的值。
如果函数失败，返回值为零。


ULONG_PTR WINAPI SetClassLongPtr(
    _In_ HWND     hWnd,             //窗口的句柄，间接指向窗口所属的类。
    _In_ int      nIndex,           //要设置的信息,详见MSDN.
    _In_ LONG_PTR dwNewLong         //把信息设置为dwNewLong.
);
返回值:
如果函数成功，则返回值是先前的信息。如果以前没有设置，则返回值为零。
如果函数失败，返回值为零。

我们可能要设置信息为0,或有时得到的信息就是0,因为返回0是失败;这给这个二个函数带来了更多的思考,用时需注意;这个我能说是微软的锅么,怎么也不改下.
```

3.4 得到 & 设置 窗口信息

```
LONG_PTR WINAPI GetWindowLongPtr(
    _In_ HWND hWnd,         //窗口的句柄，间接指向窗口所属的类。
    _In_ int  nIndex        //要设置的信息,详见MSDN.
);
返回值:
如果函数成功，返回值就是请求的值。
如果函数失败，返回值为零。

LONG_PTR WINAPI SetWindowLongPtr(
                            窗口的句柄，间接指向窗口所属的类
```

```
    _In_ HWND     hWnd,     //窗口的句柄，间接指向窗口所属的类。
    _In_ int      nIndex,   //要设置的信息,详见MSDN.
    _In_ LONG_PTR dwNewLong //把信息设置为dwNewLong.
);
返回值:
如果函数成功，则返回值是先前的信息。如果以前没有设置，则返回值为零。
如果函数失败，返回值为零。

我们可能要设置信息为0,或有时得到的信息就是0,因为返回0是失败;这给这个二个函数带来了更多的思考,用时需注意;这个我能说是微软的锅么,怎么也不改下.
```

3.5 移动窗口

```
BOOL MoveWindow(
    HWND hWnd,      // 窗口句柄
    int X,          // 水平坐标
    int Y,          // 垂直坐标
    int nWidth,     // 窗口的宽
    int nHeight,    // 窗口的高
    BOOL bRepaint   // 是否擦除原来的
);
```

4. 资料

上面有很多的函数,我要在哪里找资料?

- 百度百科
- MSDN (有中文版,或翻译为中文)

5. 作业

1. 创建一个窗口,修改光标,图标(要求通过资源方式加载),当光标在窗口左边的时候是光标1,窗口右边的时候是光标2.
2. 修改窗口背景. (选做)