第一课　编程基础

一、课程介绍

1. Windows编程基础

1) 不要背函数，猜函数是一项技能。理解+猜。
   Windows源代码不公开，根据自己掌握的知识推理。

2) 常识性的东西。

2. Windows消息和消息机制

Windows是基于消息的操作系统。

3. Windows绘图

1) 视窗系统时时处处都有绘图。一切都是用代码画出来的。
2) Windows界面编程十分强大，客户端、工业控制类编程(上位机)。

4. Windows控件

Windows的基本控件：按钮、文本编辑框、组合框，等等。

5. Windows资源管理

资源：图标、光标、快捷键、菜单、工具栏，等等。

6. Windows文件处理

1) 文件处理是操作系统的必要组成部分。

2) CreateFile/ReadFile/WriteFile/CloseHandle对应
   fopen/fread/fwrite/fclose。

7. Windows的进程和线程

创建子进程和子线程。每个程序的入口函数都是运行在主线程中的。

8. Windows内存管理

内存管理是操作系统的必要组成部分。

二、Windows编程基础

1. Windows应用程序的类型

1) 控制台(console)程序：DOS程序，本身没有窗口，
通过Windows DOS窗口执行。

控制台窗口是Windows系统提供的，不是程序自己创建的。

写一个控制台程序：

建一个工作区Day01，.dsw文件是工作区文件；
添加新工程，选控制台程序，指定工程名WinConsole；
选择A "Hello, World!" application；
看一下WinConsole.cpp；
编译，提示错误，VC的BUG，忽略之；
编译链接，没有错误；
运行一下；
关闭所有窗口。

范例：WinConsole

2) 窗口程序：拥有自己的窗口，可以与用户交互。

程序自己通过代码创建窗口。

写一个窗口程序：

添加新工程，选Win32 Application，指定工程名WinCreate；
选择A typical "Hello World!" application；
看一下WinCreate.cpp，WinMain是入口函数；
编译链接，运行一下，"Hello World!"字符串不在代码中，在资源中；
关闭所有窗口。

范例：WinCreate

3) 库程序：存放代码、数据的程序，
执行文件可以从中取出代码执行和获取数据。

A. 静态库程序：扩展名LIB，在编译链接程序时，
将代码嵌入到执行文件中。

写一个静态库程序：

添加新工程，选Win32 Static Library，指定工程名WinLib；
选择Pre-Compiled header；
看一下预编译头文件StdAfx.h和源文件StdAfx.cpp；
编译链接，生成WinLib.lib，其中封装了二进制形式的函数代码，不可运行；
关闭所有窗口。

范例：WinLib

B. 动态库程序：扩展名DLL，在执行文件执行时从中获取代码。

写一个动态库程序：

添加新工程，选Win32 Dynamic-Link Library，指定工程名WinDll；
选择A simple DLL project；
看一下WinDll.cpp，DllMain即是动态库的入口函数，可运行，
但不会自己运行，需要依附于其它可执行程序运行；
动态库有自己的进程空间，其中存放动态库内部的函数代码、
全局变量、静态变量、常量，等等；
编译链接，生成WinDll.dll，不可独立运行；
关闭所有窗口。

范例：WinDll

4) 四类Windows程序对比

	控制台程序	窗口程序	静态库	动态库
入口函数	main	WinMain	没有	DllMain
生成文件	.exe	.exe	.lib	.dll
运行方式	在DOS窗口中运行	在自己的窗口中运行	不可运行 代码嵌入到可执行程序或动态库中	不能独立运行 代码被可执行程序或其它动态库调用

2. Windows开发工具和库

1) 集成开发环境(IDE)

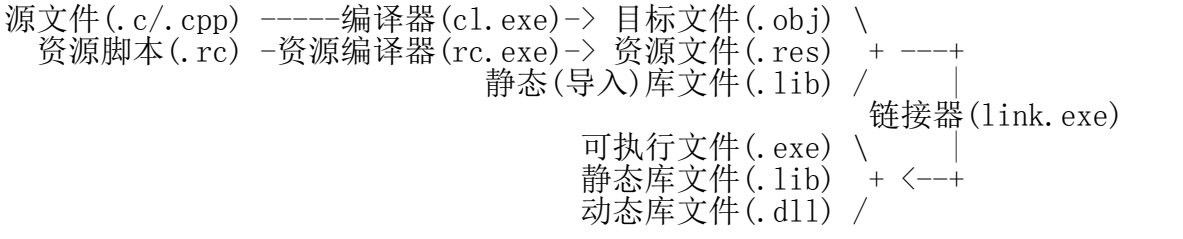
C/C++、VC、Visual Studio
VC1.5 - Win16
VC2.0 - Win32
VC5.0 - Visual Studio 97
VC6.0 - Visual Studio 98
VC7.0/8.0/9.0/10.0 - Visual Studio 2003/2005/2008/2010

2) 工具链

编译器：C:\Program Files\Microsoft Visual Studio\VC98\Bin\cl.exe，
将源代码编译成目标代码。

链接器：C:\Program Files\Microsoft Visual Studio\VC98\Bin\link.exe，
将目标代码、库和资源链接生成最终文件。

资源编译器：C:\Program Files\Microsoft Visual Studio\Common\MSDev98\Bin\rc.exe，
编译资源，最终通过链接器存入最终文件。



3) Win32 SDK (Platform SDK) = 库 + 头文件 + 文档

A. 库：封装了Windows APIs - 非常庞大的一套函数接口。

C:\WINDOWS\system32目录下：

```
  user32.dll - 窗口、消息、资源等APIs
   gdi32.dll - 绘图相关APIs
kernel32.dll - 进程、线程、内存管理等内核APIs
```

C:\Program Files\Microsoft Visual Studio\VC98\Lib目录下：

```
  user32.lib - user32.dll的导入库
   gdi32.lib - gdi32.dll的导入库
kernel32.lib - kernel32.dll的导入库
```

B. 头文件：库函数、外部变量的声明，类型定义，宏定义等。

C:\Program Files\Microsoft Visual Studio\VC98\Include目录下：

```
windows.h - 所有windows头文件的集合，已包含下面的头文件
 windef.h - windows数据类型
winuser.h - user32.dll的APIs
 wingdi.h - gdi32.dll的APIs
winbase.h - kernel32.dll的APIs
  winnt.h - UNICODE字符集支持
```

C. 文档：MSDN，介绍MSDN的使用。

3. 不使用IDE编写窗口程序

1) 入口函数

```
int WINAPI WinMain (
    HINSTANCE hInstance,      // 当前程序的实例句柄
    HINSTANCE hPrevInstance, // 当前程序的前一个实例句柄，
                              // Win32下已废弃，总为NULL
    LPSTR     lpCmdLine,      // 命令行参数字符串
    int       nCmdShow        // 窗口显示方式
                              // 最大化、最小化、非最大最小化
) {
    函数体代码;
}
```

若该函数执行成功，并在收到WM_QUIT消息后终止，
则应返回包含在该消息wParam参数中的退出码。
若该函数在进入消息循环之前终止，则应返回0。

A. 调用约定

windef.h中定义了如下调用约定宏：

```
#define WINAPI   __stdcall \
#define APIENTRY __stdcall  \ PASCAL调用(标准调用)，
#define CALLBACK __stdcall  / 被调用者负责清理堆栈。
```

```
#define PASCAL   __stdcall /
#define WINAPIV  __cdecl   -- C调用(缺省)，调用者负责清理堆栈
```

a) 相对于PASCAL调用，C调用在多次调用同一个函数的情况下，
内容相同的清栈代码会重复出现，编译生成的二进制模块会比较大。

b) 带有变长参数表的函数，如printf/scanf，只能使用C调用约定，
因为参数个数不确定，编译器无法在被调用者中生成清栈代码。

c) 无论PASCAL调用，还是C调用，参数入栈的顺序都是从右到左。
对于C++的类成员函数，最后一个入栈的是this指针，
相当于第一个参数。

B. 句柄

内存对象的唯一标识。并非指针。

应用程序实例句柄就是进程内存空间的句柄。

2) 消息框函数

```
int MessageBox (
    HWND    hWnd,       // 父窗口句柄
    LPCTSTR lpText,     // 显示在消息框中的文字
    LPCTSTR lpCaption, // 显示在标题栏中的文字
    UINT    uType       // 消息框的按钮、图标类型。
                        // 组内排斥，组间位或
);
```

失败返回0，成功返回用户点击的按钮标识(非0)。

3) 按如下步骤操作：

A. 用记事本编辑WinBox.c，保存在WinBox目录下。
B. 验证cl是否可用：cl /?。若不能使用，
则将C:\Program Files\Microsoft Visual Studio\VC98\Bin\VCVARS32.BAT
批处理文件，拷贝到WinBox目录下，执行之。
C. 执行：cl /c WinBox.c。编译生成WinBox.obj目标文件。
D. 执行：link WinBox.obj user32.lib。链接生成WinBox.exe可执行文件。
E. 运行WinBox.exe程序。

4) 试验命令行参数：WinMain函数的lpCmdLine参数。
拖拽一个文件到WinBox.exe上。

范例：WinBox

三、编写第一个窗口程序

1. 定义WinMain函数：程序的入口

2. 定义窗口过程函数：处理消息

1) 窗口过程函数

```
LRESULT CALLBACK WindowProc (
    HWND   hWnd,   // 窗口句柄
    UINT   uMsg,   // 消息标识
    WPARAM wParam, // 消息参数
    LPARAM lParam  // 消息参数
) {
    函数体代码;
}
```

2) 缺省窗口过程函数

```
LRESULT CALLBACK DefWindowProc (
    HWND   hWnd,   // 窗口句柄
    UINT   uMsg,   // 消息标识
    WPARAM wParam, // 消息参数
    LPARAM lParam  // 消息参数
);
```

3. 注册窗口类：先注册才能创建

1) 窗口类结构体

```
typedef struct {
    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
    LPCTSTR   lpszMenuName;  // 菜单资源名
    LPCTSTR   lpszClassName; // 窗口类名
}   WNDCLASS, *PWNDCLASS;
```

2) 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。

4. 创建窗口：在内存中创建窗口对象，并不可见

```
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
);
```

成功返回所创建窗口的句柄，失败返回NULL。

5. 显示窗口：在屏幕上显示窗口

1) 显示

```
BOOL ShowWindow (
    HWND hWnd,     // 窗口句柄
    int  nCmdShow // 显示方式
);
```

成功返回TRUE，失败返回FALSE。

2) 刷新

```
BOOL UpdateWindow(
    HWND hWnd // 窗口句柄
);
```

成功返回TRUE，失败返回FALSE。

6. 消息循环：重复执行提取消息、翻译消息、派发消息三步

1) 从窗口的消息队列中提取一条消息。

```
BOOL GetMessage (
    LPMSG lpMsg,          // 消息结构
    HWND  hWnd,           // 窗口句柄
    UINT  wMsgFilterMin, // 起始消息
    UINT  wMsgFilterMax  // 终止消息
);
```

收到WM_QUIT消息返回FALSE，收到其它消息返回TRUE。

2) 将虚键消息翻译为字符消息。

```
BOOL TranslateMessage (
    const MSG* lpMsg // 消息结构
);
```

若消息被翻译则返回TRUE，否则返回FALSE。

3) 将消息派发到窗口过程函数。

```
LRESULT DispatchMessage (
    const MSG* lpmsg // 消息结构
);
```

返回窗口过程函数的返回值。

7. 消息处理：在窗口过程函数中处理消息

DefWindowProc函数对WM_DESTROY消息的处理不会投递WM_QUIT消息，
导致消息循环不会结束，窗口虽然关闭，但进程仍然存在。

```
switch (uMsg) {
    case WM_DESTROY:
        PostQuitMessage (0);
        break;

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

范例：WinHello

四、资源脚本和资源文件
---

1. 资源脚本是扩展名为.rc的文本文件

将WinCreate\WinCreate.ico拷贝到WinHello\WinHello.ico;
在WinHello目录下编辑WinHello.rc文件，加入文本行：
100 ICON WinHello.ico。

2. 资源文件是扩展名为.res的二进制文件

用资源编译器rc.exe编译资源脚本，生成资源文件，
执行：rc WinHello.rc，生成WinHello.res。

3. 用链接器link.exe将资源文件链接到可执行程序中

执行：link WinHello.obj WinHello.res user32.lib，
观察WinHello.exe图标的变化。

范例：WinHello

五、Makefile和NMAKE
---

1. Makefile

定义编译和链接等操作的脚本文件。

编辑：WinHello.mak文件

```
all:
    cl /c WinHello.c
    rc WinHello.rc
    link WinHello.obj WinHello.res user32.lib

clean:
    del *.exe *.obj *.res
```

一个Makefile脚本中可以包含多个依赖行，
每个依赖行可以包含多条命令。

2. NMAKE
~~~~~~~

解释执行Makefile脚本，编译链接程序，生成最终文件。

先执行：nmake /f WinHello.mak clean

再执行：nmake /f WinHello.mak build

缺省执行第一个依赖行。

执行：nmake /f WinHello.mak

或者将WinHello.mak改为：

```
all: clean
    cl /c WinHello.c
    rc WinHello.rc
    link WinHello.obj WinHello.res user32.lib

clean:
    del *.exe *.obj *.res
```

确保先删除后生成。

NMAKE首先找到第一个或指定的依赖行，检查该依赖行是否有依赖项，
若有依赖项，则先执行依赖项的命令，然后再执行自己的命令。

范例：WinHello

六、字符编码——字符与数字的对应
----

1. ASC
~~~~~

7位二进制数表示一个字符，共128个字符。
英语字母大小写、阿拉伯数字、标点符号。

2. ASCII
~~~~~~~

American Standard Code for Information Interchange
美国信息交换标准代码

8位二进制数表示一个字符，共256个字符。
增加欧洲语言字符。
通过代码页(CodePage)标识不同的字符集。
不同字符集的前128个字符都相同，后128个字符因不同语言而异。

英语 - 437
中文 - 936
~~~~~~~

通过SetConsoleOutputCP函数设置控制台的代码页。

3. MBCS/DBCS

Multi-Byte Character Set/Double-Byte Character Set
多字节字符集/双字节字符集

用1个或2个字节表示一个字符。单双字节混合容易出现乱码。

```
 C  语    言
-- ----- -----
43 D3 EF D1 D4
----- ----- --
    〈乱码〉
```

strlen函数返回5。

4. UNICODE/UCS-2

Universal CODE/Universal Character Set with 2 bytes
通用码/2字节通用字符集

1) 统一用2个字节表示一个字符。
   某些字符中可能含有空字符，需要特殊的处理函数。

```
 C     语    言
----- ----- -----
43 00 ED 8B 00 8A
   ^^       ^^
```

strlen函数返回1。
wcslen函数返回3。

2) 宽字符字符串：每个字符占2个字节，采用UNICODE码。

A. 宽字符类型wchar_t：实际是unsigned short类型，
   取值范围从0到65536，相应的字符串字面值需要加“L”。

B. 支持wchar_t类型的字符串函数：wcslen、wprintf，等等。

如：

```
wchar_t wsText[] = L"Hello World";
wprintf (L"%s, %u, %u\n",
    wsText, wcslen (wsText), sizeof (wsText));
```

注意：字符串长度不同于字符串的字节数。
宽字符字符串同样要以空字符结尾，
宽字符的空为wchar_t类型的0，即两个0字节。

3) 同时支持多字节(MBCS/DBCS)字符串和
   宽字符(UNICODE/UCS-2)字符串的代码。

系统头文件tchar.h中包含类似下面的代码：

```
#ifdef _UNICODE
typedef wchar_t TCHAR;
#define _T(x) L##x
#define _tcslen wcslen
#define _tprintf wprintf
...
#else
typedef char TCHAR;
#define _T(x) x
#define _tcslen strlen
#define _tprintf printf
...
#endif // _UNICODE
```

同时支持多字节和宽字符的代码：

```
#include <tchar.h>

TCHAR szText[] = _T ("Hello World");
_tprintf (_T ("%s, %u, %u\n"),
    szText, _tcslen (szText), sizeof (szText));
```

Project/Settings.../C/C++/Preprocessor definitions中的预定义宏：
_MBCS表示多字节，_UNICODE和UNICODE表示宽字符。

类型别名	实际类型	指针	常指针
CHAR	char	LPSTR	LPCSTR
WCHAR	wchar_t	LPWSTR	LPCWSTR
TCHAR	char (_MBCS) wchar_t (_UNICODE)	LPTSTR	LPCTSTR

4) UNICODE编码的汉字打印

A. 汉字的UNICODE编码范围从0x4E00到0x9FA5，共20902个字符。

B. wprintf对UNICODE编码的汉字字符打印的支持不够完善。

C. 用WriteConsoleW函数打印UNICODE编码的汉字。

输出字符串到标准输出：

```
BOOL WINAPI WriteConsoleW (
    HANDLE      hConsoleOutput,          // 标准输出句柄(类似标准I/O流的
                                         // stdout指针)
    const VOID* lpBuffer,                // 输出内容缓冲区
    DWORD       nNumberOfCharsToWrite,   // 输出内容的字符数(不是字节数)
    LPDWORD     lpNumberOfCharsWritten,  // 实际输出的字符数
    LPVOID      lpReserved               // 保留，NULL
```

```
);
```

成功返回TRUE，失败返回FALSE。

获取标准句柄：

```
HANDLE WINAPI GetStdHandle (
    DWORD nStdHandle
);
```

```
STD_INPUT_HANDLE  - 标准输入句柄
STD_OUTPUT_HANDLE - 标准输出句柄
STD_ERROR_HANDLE  - 标准出错句柄
```

成功返回相应的句柄，失败返回INVALID_HANDLE_VALUE。

范例：WinChar