Lập trình C trên Windows

Các khái niệm cơ bản

Nguyễn Tri Tuấn
Khoa CNTT – ĐH.KHTN.Tp.HCM
Email: nttuan@ fit.hcmuns.edu.vn

Nội dung

- Vài điểm khác biệt giữa lập trình Windows và DOS
- Các thư viện lập trình của Windows
- Các loại ứng dụng
- Các khái niệm cơ bản
- Lập trình sự kiện (Even driven programming)
- Các thành phần giao diện đồ họa (GUI)
- Tài nguyên của ứng dụng (Resources)
- Một chương trình tổng quát xây dựng trên Win32 API
- Các kiểu dữ liệu cơ bản

Vài điểm khác biệt giữa lập trình Windows và DOS

Windows	**DOS**
Lập trình sự kiện, dựa vào thông điệp (message)	Thực hiện tuần tự theo chỉ định
Multi-tasking	Single task
Multi-CPU	Single CPU
Tích hợp sẵn Multimedia	Phải dùng các thư viện Multimedia riêng
Hỗ trợ 32 bits hay hơn nữa	Ứng dụng 16 bits
Hỗ trợ nhiều công nghệ DLL, OLE, DDE, COM, OpenGL, DirectX, …	Không có

Các thư viện lập trình của Windows

- SDK - Software Development Kit
  - Là bộ thư viện lập trình nền tảng của HĐH Windows
  - Cung cấp tất cả các công cụ cần thiết để xây dựng 1 ứng dụng trên Windows
  - Được sử dụng như là thư viện cơ sở để tạo ra những thư viện cấp cao hơn trong những ngôn ngữ lập trình. VD. OWL của BorlandC, MFC của Visual C++,…

Các thư viện lập trình của Windows …(tt)

- Một số thành phần tiêu biểu của SDK:
  - Win32 API
  - GDI/GDI+
  - Windows MultiMedia
  - OpenGL
  - DirectX
  - COM/COM+
  - ADO (ActiveX Data Object)
  - OLE DB
  - …

  Xem thêm *MSDN/Platform SDK Documentation/Getting started/Content of Platform SDK*

Các thư viện lập trình của Windows …(tt)

- OWL - Object Windows Library
  - Là bộ thư viện hướng đối tượng của BorlandC++
- MFC - Microsoft Foundation Classes
  - Là bộ thư viện hướng đối tượng của Visual C++
- Một ứng dụng trên Windows có thể được viết bằng
  - Thư viện SDK
  - Một thư viện khác (OWL, MFC,…) phối hợp với SDK

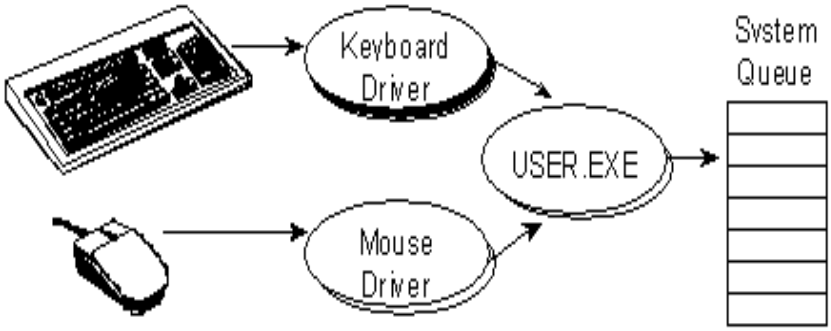
Các loại ứng dụng

- Win32 Console: ứng dụng 32 bits, với giao diện dạng DOS command line
- Win32 (SDK): ứng dụng 32 bits, chỉ sử dụng thư viện SDK
- Win32 DLL: ứng dụng 32 bits, dạng thư viện liên kết động (Dynamic-linked library), sử dụng SDK
- Win32 LIB: ứng dụng 32 bits, dạng thư viện liên kết tĩnh (Static-linked library)
- MFC EXE: ứng dụng 32 bits, sử dụng thư viện Microsoft Foundation Class
- MFC DLL: ứng dụng 32 bits, dạng thư viện liên kết động (Dynamic-linked library), sử dụng MFC
- …

Các khái niệm cơ bản

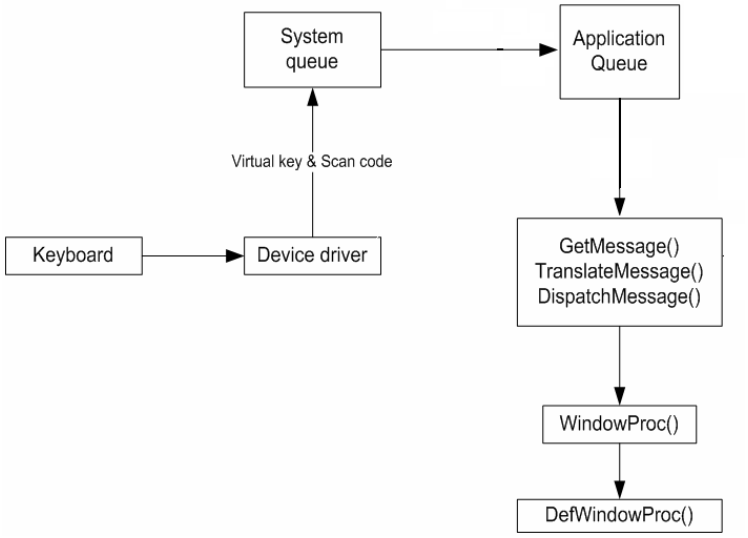
- Handle:
  - Một giá trị 32 bits không dấu (unsigned) do HĐH tạo ra để làm định danh cho 1 đối tượng (cửa sổ, file, vùng nhớ, menu,…)
- ID (Identifier):
  - Một giá trị nguyên do ứng dụng tạo ra để làm định danh cho 1 đối tượng (menu item, control)
- Instance:
  - Một giá trị nguyên do HĐH tạo ra để định danh 1 thể hiện đang thực thi của ứng dụng
- Callback:
  - Thuộc tính của 1 hàm/thủ tục sẽ được gọi bởi HĐH, không phải bởi ứng dụng

Lập trình sự kiện (Even driven programming)



Phát sinh các sự kiện và thông điệp

Lập trình sự kiện (Even driven programming)…(tt)



Qui trình xử lý thông điệp

Lập trình sự kiện (Even driven programming)…(tt)

```
// Vòng lặp xử lý thông điệp trong 1 ứng dụng –
// Message loop
MSG         msg;

// lấy thông điệp ra khỏi hàng đợi của ứng dụng
while (GetMessage(&msg, NULL, 0, 0))
{
      // chuyển đổi các phím ảo (virtual key) thành các thông điệp ký tự
      TranslateMessage(&msg);
      // chuyển message đến hàm xử lý thông điệp của cửa
      // sổ tương ứng
      DispatchMessage(&msg);
}
return  msg.wParam;
```

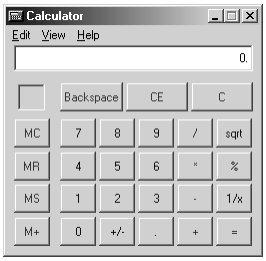

Các thành phần giao diện đồ họa (GUI)

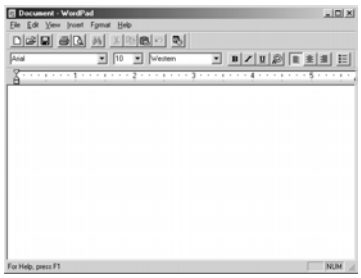
- Các dạng GUI cơ bản
- Cửa sổ
  - Định nghĩa
  - Nguyên tắc quản lý
  - Phân loại
  - Lớp cửa sổ (window class)
  - Hàm xử lý thông điệp (window procedure)
  - Các thuộc tính

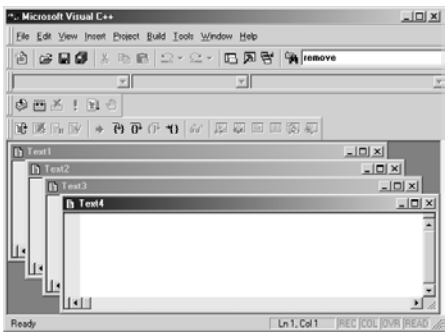
Các thành phần giao diện đồ họa (GUI)…(tt)

- **GUI: Graphics User Interface**
- **Các dạng GUI cơ bản:**
  - **SDI - Single Document Interface**
    - Một cửa sổ làm việc
    - Cho phép thay đổi kích thước cửa sổ (Resizeable)
    - Không có các cửa sổ con
    - Ví dụ: NotePad, Paint,…
  - **MDI - Multiple Document Interface**
    - Một cửa sổ làm việc chính (Frame window) và nhiều cửa sổ con (Child window)
    - Cho phép thay đổi kích thước cửa sổ (Resizeable)
    - Cho phép Maximize/Minimize/Close các cửa sổ con
    - Ví dụ: Word, Excel, VC++,…
  - **Dialog**
    - Một cửa sổ làm việc
    - Thường kích thước cố định
    - Thường không có menu bar
    - Thường có các button, edit box, list-box,…
    - Ví dụ: Calculator, CD Player,…

Các thành phần giao diện đồ họa (GUI)…(tt)







Dialog, SDI, MDI

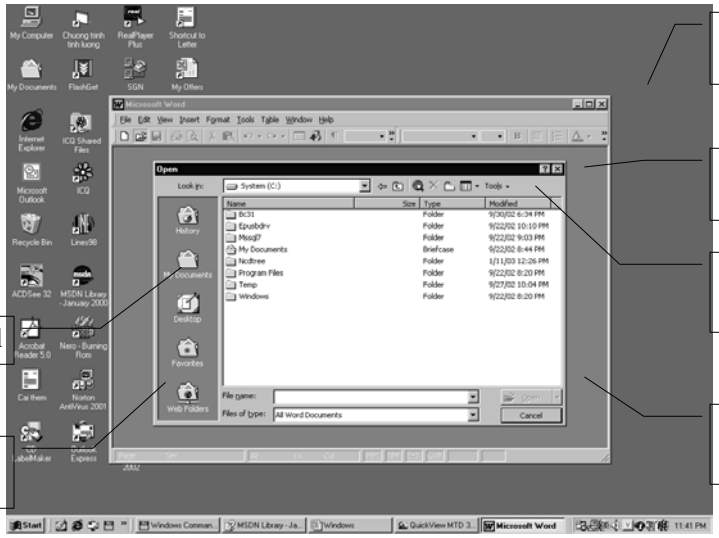
GUI – Window …(tt)

- Định nghĩa:
  - là 1 vùng chữ nhật trên màn hình,
  - dùng để hiển thị kết quả output,
  - và nhận các input từ người dùng
  - ➢ Công việc đầu tiên của 1 ứng dụng GUI là tạo 1 cửa sổ làm việc
- Nguyên tắc quản lý:
  - Mô hình phân cấp: mỗi cửa sổ đều có 1 cửa sổ cha (parent window), ngoại trừ cửa sổ nền Desktop
  - Tại mỗi thời điểm, chỉ có 1 cửa sổ nhận input từ user (Active/Focused window)

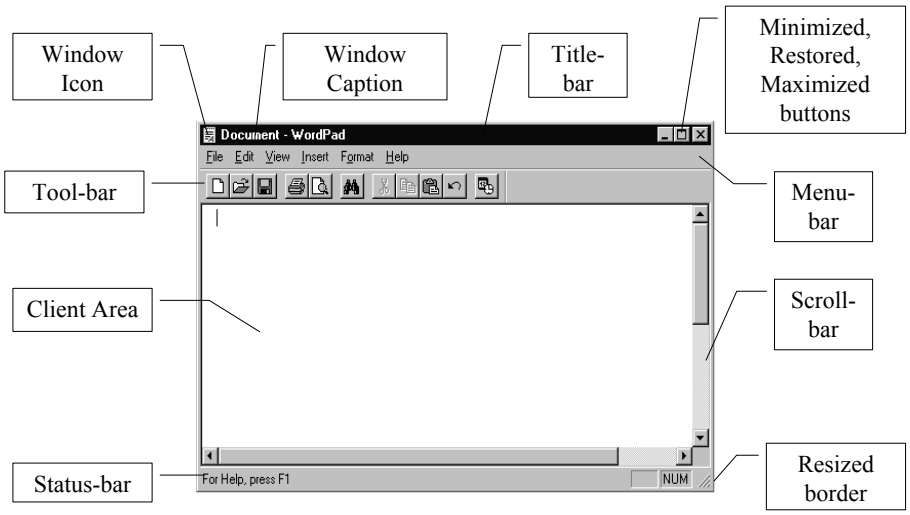
GUI – Window …(tt)

- Phân loại:
  - Cửa sổ Desktop
  - Cửa sổ tiêu chuẩn
  - Cửa sổ hộp thoại (Dialog box)
  - Các control

GUI – Window …(tt)

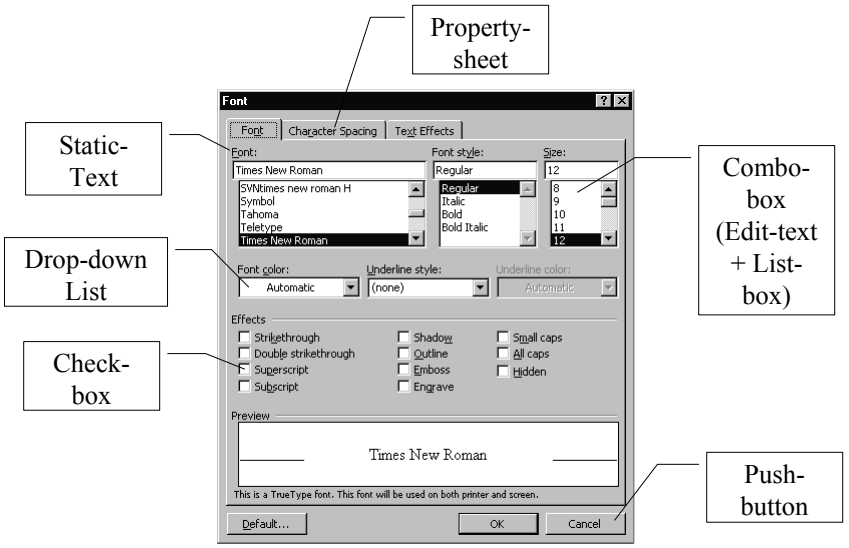


GUI – Window …(tt)



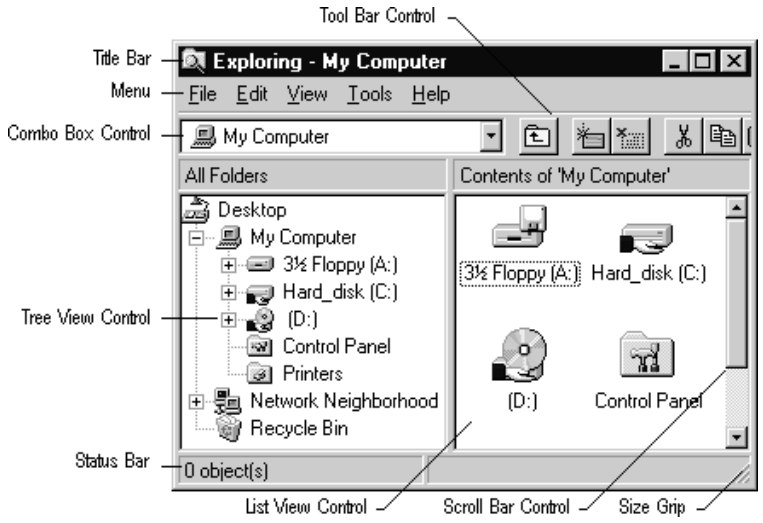
Các thành phần của 1 cửa sổ tiêu chuẩn

GUI – Window …(tt)



Các dạng control

GUI – Window …(tt)



Các dạng control

GUI – Window …(tt)

- Lớp cửa sổ (window class):
  - Là một tập các thuộc tính mà HĐH Windows sử dụng làm khuôn mẫu (template) khi tạo lập 1 cửa sổ
  - Mỗi lớp cửa sổ được đặc trưng bằng 1 tên (class-name) dạng chuỗi
  - Phân loại class:
    - Lớp cửa sổ của hệ thống (System class)
      - Được định nghĩa trước bởi HĐH Windows
      - Các ứng dụng không thể hủy bỏ
    - Lớp cửa sổ do ứng dụng định nghĩa:
      - Được đăng ký bởi ứng dụng
      - Có thể hủy bỏ khi không còn sử dụng nữa
      - Lớp toàn cục của ứng dụng (Application global class)
      - Lớp cục bộ của ứng dụng (Application local class)

GUI – Window …(tt)

Class	**Description**
Button	The class for a button
ComboBox	The class for a combo box.
Edit	The class for an edit control.
ListBox	The class for a list box
MDIClient	The class for an MDI client window
ScrollBar	The class for a scroll bar
Static	The class for a static control

Các lớp cửa sổ của hệ thống

GUI – Window …(tt)

- Lớp cửa sổ (window class):
  - Mỗi cửa sổ đều thuộc một lớp xác định
  - Cần phải đăng ký lớp trước khi tạo lập 1 cửa sổ, nếu lớp chưa được đăng ký
  - Đăng ký lớp cửa sổ:
    - Cấu trúc dữ liệu: WNDCLASS / WNDCLASSEX
    - Hàm sử dụng: RegisterClass / RegisterClassEx
  - Hủy bỏ lớp cửa sổ đã đăng ký:
    - Hàm sử dụng: UnregisterClass

GUI – Window …(tt)

VD. Đăng ký 1 lớp cửa sổ

```
WNDCLASS   wc;
wc.lpszClassName = “MyClass”;       // tên class
wc.lpfnWndProc = MyWndProc;         // tên hàm xử lý message
wc.hInstance  = hInstance;
wc.hCursor = LoadCursor(hInstance, (LPSTR) IDC_CURSOR);
wc.hIcon = LoadIcon(hInstance, (LPSTR) IDI_ICON);
wc.hbrBackground = (HBRUSH) (COLOR_APPWORKSPACE + 1);
wc.lpszMenuName = (LPSTR) IDR_MENU;
wc.style = CS_HREDRAW | CS_VREDRAW | CS_DBLCLKS;
wc.cbClsExtra = 0;
wc.cbWndExtra = 0;

if (!RegisterClass(&wc)) return FALSE;
```

GUI – Window …(tt)

- Minh họa cấu trúc WNDCLASS

```
typedef struct _WNDCLASS {
    UINT        style;
    WNDPROC     lpfnWndProc;
    int         cbClsExtra;
    int         cbWndExtra;
    HINSTANCE   hInstance;
    HICON       hIcon;
    HCURSOR     hCursor;
    HBRUSH      hbrBackground;
    LPCTSTR     lpszMenuName;
    LPCTSTR     lpszClassName;
} WNDCLASS, *PWNDCLASS;
```


GUI – Window …(tt)

Thuộc tính	**Ý nghĩa**
lpszClassName	tên lớp, chuỗi kết thúc bằng 0
lpfnWndProc	hàm tiếp nhận và xử lý thông điệp
hInstance	Instance của ứng dụng thực hiện việc đăng ký class
hCursor	handle của cursor, xác định dạng cursor trong vùng client
hIcon	handle của icon, xác định biểu tượng của cửa sổ
hbrBackground	handle của brush, xác định cách tô vùng client; +1 nếu muốn sử dụng màu mặc định của hệ thống. VD. COLOR_BACKGROUND + 1
lpszMenuName	tên menu bar, chuỗi kết thúc bằng 0
Style	kiểu dáng của lớp (class style)
cbClsExtra	số byte dành riêng cho class (<= 40 bytes)
cbWndExtra	số byte dành riêng cho mỗi cửa sổ thuộc class (<= 40 bytes)

GUI – Window …(tt)

- Hàm xử lý thông điệp (window procedure)
  - Có nhiệm vụ tiếp nhận và xử lý các thông điệp được gởi đến cho cửa sổ
  - Sử dụng chung cho tất cả các cửa sổ của cùng 1 class
  - Được gọi bởi HĐH (là hàm *Callback*)

GUI – Window …(tt)

- Dạng tổng quát của 1 hàm xử lý thông điệp

```
LRESULT CALLBACK WndProc(
        HWND hwnd,              // handle của window nhận message
        UINT uMsg,              // ID của message
        WPARAM wParam,          // tham số thứ 1 của message (WORD)
        LPARAM lParam)          // tham số thứ 2 của message (LONG)
{
        switch (uMsg) {
            case WM_CREATE:    // khởi tạo window.
                return 0;
            case WM_PAINT:     // vẽ lại (cập nhật) vùng client của window
                return 0;
            case WM_SIZE:      // thay đổi kích thước và vị trí của window.
                return 0;
            case WM_DESTROY: // hủy bỏ window. Giải phóng các DL liên quan
                return 0;
            // …
            // Xử lý các message khác
            default: return DefWindowProc(hwnd, uMsg, wParam, lParam);
        }
}
```

GUI – Window …(tt)

- Các thuộc tính của cửa sổ:
  - tên lớp (class name)
  - tiêu đề (window title)
  - kiểu dáng (window style)
  - kiểu dáng mở rộng (extended window style)
  - vị trí (position – x, y)
  - kích thước (size – w, h)
  - cửa sổ cha (parent or owner window handle)
  - menu hay ID (menu handle or child-window identifier)
  - instance của ứng dụng sở hữu (application instance handle)

GUI – Window …(tt)

VD. Tạo lập 1 cửa sổ bằng hàm CreateWindow

```
HWND hWnd =
   CreateWindow("MyClass",                  // class name
        "XYZ Application",                  // window title
        WS_OVERLAPPEDWINDOW,                // window style
        CW_USEDEFAULT,                      // default x
        CW_USEDEFAULT,                      // default y
        CW_USEDEFAULT,                      // default Width
        CW_USEDEFAULT,                      // default Height
        (HWND) NULL,                        // parent handle
        (HMENU) NULL,                       // MenuBar
        hInstance,                          // application instance
        NULL);                              // extra default data
```

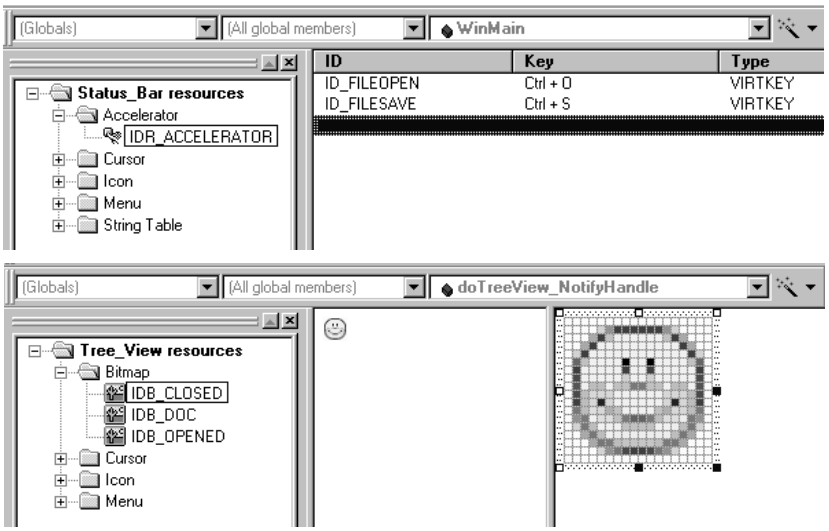
Tài nguyên của ứng dụng (Resources)

- Resouce:
  - là 1 đối tượng (object) được sử dụng trong ứng dụng (VD. Menu bar, dialog, bitmap, icon, cursor,…)
  - được định nghĩa bên ngoài và được thêm vào trong file thi hành của ứng dụng khi biên dịch (linking)

Tài nguyên của ứng dụng (Resources)…(tt)

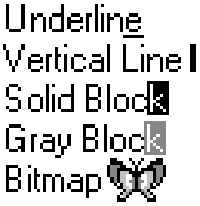
- Các dạng resource:
  - Accelerator table: - bảng mô tả phím tắt (hot-key)
  - Bitmap: - ảnh bitmap
  - Caret: - con trỏ văn bản
  - Cursor: - con trỏ chuột
  - Dialog box: - khung hộp thoại (Dialogbox Template)
  - Enhanced metafile: - tập hợp các cấu trúc để lưu ảnh (picture) theo định dạng “độc lập thiết bị” (Device-Independent format)
  - Font: - font chữ
  - Icon: - biểu tượng
  - Menu: - menu
  - String-table entry: - bảng mô tả các chuỗi ký tự
  - Version information: - bảng mô tả thông tin “phiên bản”

Tài nguyên của ứng dụng (Resources)…(tt)

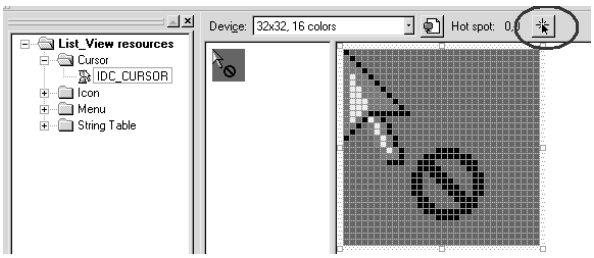


Accelerator và Bitmap

Tài nguyên của ứng dụng (Resources)…(tt)

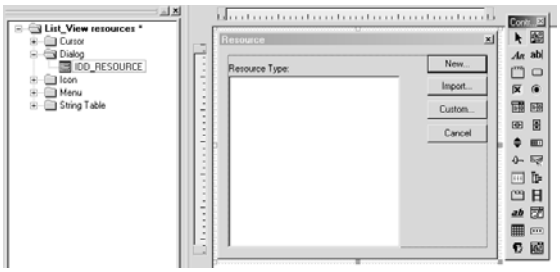


Caret

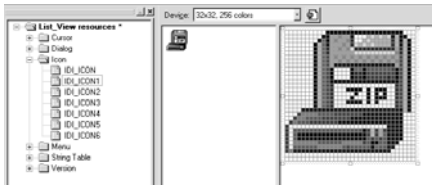


Cursor

Tài nguyên của ứng dụng (Resources)…(tt)

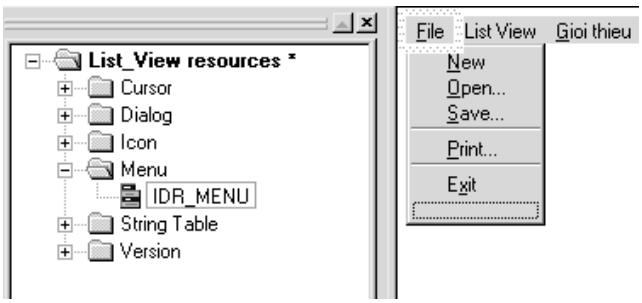


DialogBox template

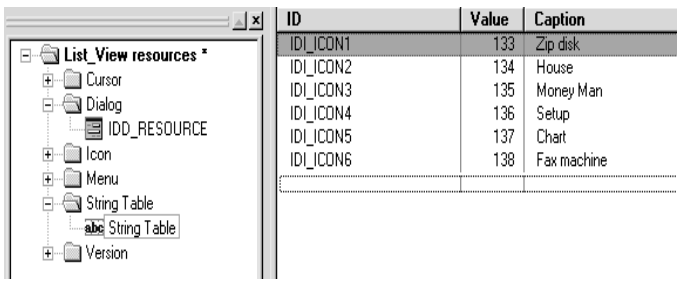


Icon

Tài nguyên của ứng dụng (Resources)…(tt)

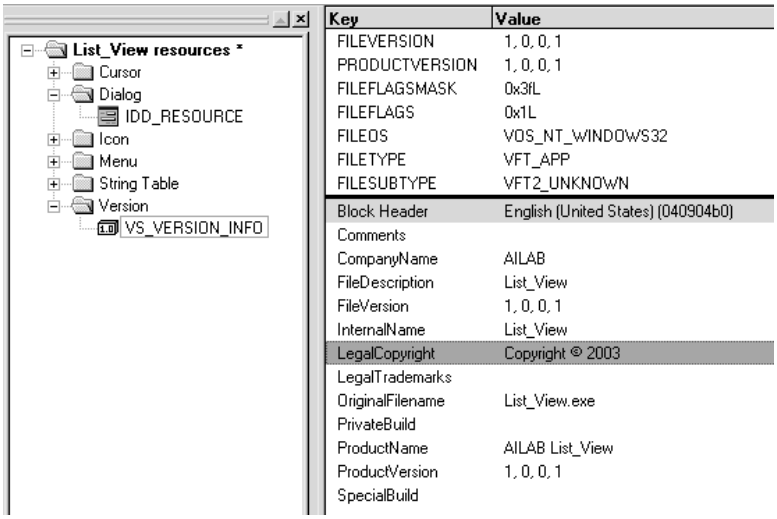


Menubar



String table

Tài nguyên của ứng dụng (Resources)…(tt)



Version Information

Một chương trình tổng quát xây dựng trên Win32 API

- Các bước tạo lập chương trình
- Hàm bắt đầu
  - WinMain
  - Đăng ký lớp cửa sổ
  - Tạo lập cửa sổ giao diện
  - Thực hiện vòng lặp xử lý message
- Menu script
- Hàm xử lý message của cửa sổ

Một chương trình tổng quát xây dựng trên Win32 API…(tt)

- WinMain:

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

  - hInstance: handle của ứng dụng
  - hPrevInstance: handle của thể hiện trước của cùng một ứng dụng. Trong Win32, giá trị này = NULL
  - lpCmdLine: chuỗi chứa dòng lệnh (command line).
    VD. “winword c:\My documents\baitap.doc”
  - nCmdShow: xác định cách thức cửa sổ được hiển thị.
    VD. MAXIMIZE, MINIMIZE, SHOW, HIDE,…

Một chương trình tổng quát xây dựng trên Win32 API…(tt)

- Đăng ký lớp cửa sổ:
  ATOM MyRegisterClass(HINSTANCE hInstance)
- Tạo lập cửa sổ giao diện:
  BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
- Thực hiện vòng lặp xử lý message

Một chương trình tổng quát xây dựng trên Win32 API…(tt)

- Menu script:

```
IDC_MYAPP MENU DISCARDABLE
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "E&xit",          IDM_EXIT
    END
    POPUP "&Help"
    BEGIN
        MENUITEM "&About ...",     IDM_ABOUT
    END
END
```


Một chương trình tổng quát xây dựng trên Win32 API…(tt)

- Hàm xử lý message của cửa sổ:
  - Xử lý message cho cửa sổ giao diện chính

```
LRESULT  CALLBACK  WndProc(HWND hWnd, UINT message,
                           WPARAM wParam,
                           LPARAM lParam)
```

  - Xử lý message cho dialog box “About”

```
LRESULT  CALLBACK  About(HWND hDlg, UINT message,
                         WPARAM wParam,
                         LPARAM lParam)
```

Cám ơn - Hỏi & Đáp

Lập trình C trên Windows

Thư viện liên kết động (DLL – Dynamic Link Library)

Nguyễn Tri Tuấn
Khoa CNTT – ĐH.KHTN.Tp.HCM
Email: nttuan@ fit.hcmuns.edu.vn

Nội dung

- Giới thiệu
- Xây dựng DLL
- Cách thức gọi DLL trong ứng dụng

Giới thiệu

- Liên kết (Linking) là gì ?
- Các loại thư viện
- Định nghĩa DLL
- DLL và cơ chế quản lý bộ nhớ

Giới thiệu - Liên kết (Linking) là gì ?

- Liên kết là cách thức mà trình biên dịch nhúng/kết hợp các đoạn mã thực thi của những module thư viện (Lib) vào chương trình
- Có 2 cách liên kết:
  - Liên kết tĩnh (Static linking)
  - Liên kết động (Dynamic linking)

Giới thiệu – Các loại thư viện

- Thư viện liên kết tĩnh (Static linking library)
  - Dạng file: .LIB
  - Chứa mã lệnh nhị phân của các hàm thư viện
  - Dùng để nhúng vào file chương trình khi thực hiện giai đoạn liên kết (linking) của quá trình biên dịch
  - Trình biên dịch sẽ copy đoạn mã lệnh của hàm thư viện vào trong những module gọi
  - ***Ưu điểm:***
    - Dễ thực hiện
    - Chương trình có thể chạy độc lập, không cần các file kèm theo
    - Chương trình sẽ chạy nhanh hơn (nếu kích thước nhỏ)

Giới thiệu – Các loại thư viện…(tt)

- Thư viện liên kết tĩnh (tt)
  - ***Khuyết điểm:***
    - Đoạn mã lệnh được nhúng vào file chương trình → kích thước chương trình lớn, tốn bộ nhớ
    - Đoạn mã lệnh được nhúng vào nhiều file chương trình khác nhau → không tối ưu
    - Khi thay đổi hàm thư viện → phải biên dịch lại toàn bộ các file chương trình

Giới thiệu – Các loại thư viện…(tt)

- Thư viện liên kết động (Dynamic linking library)
  - Dạng file: .LIB và .DLL
  - File .Lib:
    - Thư viện nhập (Import library).
    - Không chứa mã lệnh của các hàm,
    - Chỉ chứa các thông tin cần thiết để Hệ điều hành nạp thư viện DLL và xác định các hàm export trong DLL
    - Sử dụng khi dùng cách thức *load-time dynamic linking*
  - File .DLL:
    - Chứa mã lệnh nhị phân của các hàm thư viện
    - Được tải vào bộ nhớ khi ứng dụng gọi hàm thư viện
    - Cần có khi thực hiện ứng dụng
  - Mã lệnh của các hàm sẽ không được nhúng vào trong file chương trình của ứng dụng
  - Ứng dụng chỉ cần lưu thông tin của hàm thư viện, và khi cần, Hệ điều hành sẽ tải các hàm thư viện vào bộ nhớ

Giới thiệu – Các loại thư viện…(tt)

- Thư viện liên kết động (tt)
  - ***Ưu điểm:***
    - Kích thước của ứng dụng nhỏ
    - Nhiều ứng dụng có thể dùng chung 1 DLL, do đó, tiết kiệm bộ nhớ (thông thường, các ứng dụng có data riêng, nhưng có thể chia xẻ mã lệnh)
    - Khi không còn sử dụng, có thể giải phóng DLL khỏi bộ nhớ
    - Khi cần nâng cấp, chchỉ cần thay thế file DLL, các file chương trình khác không bị ảnh hưởng
  - ***Khuyết điểm:***
    - Khó sử dụng hơn

Giới thiệu - Định nghĩa DLL

- Là thư viện chứa các hàm và dữ liệu có thể được gọi từ các module khác (module có thể là 1 ứng dụng EXE hay là một DLL khác)
- DLL được nạp vào bộ nhớ lúc run-time và được map vào vùng nhớ của tiến trình gọi
- DLL có thể chứa 2 loại hàm thư viện:
  - Export: được phép gọi từ các module khác
  - Internal: chỉ được dùng nội bộ trong DLL

Giới thiệu - DLL và cơ chế quản lý bộ nhớ

- Mỗi tiến trình (process) sẽ nạp DLL vào không gian địa chỉ ảo (virtual address space) của chính tiến trình đó
- Hệ điều hành quản lý 1 biến đếm (`Count`) cho mỗi DLL. Biến `Count` được tăng lên 1 khi DLL được nạp (bởi hàm `LoadLibrary`); và bị giảm đi 1 khi được giải phóng (bởi hàm `FreeLibrary`)
- Khi biến `Count=0` hay process kết thúc, DLL được giải phóng khỏi bộ nhớ

Giới thiệu - DLL và cơ chế quản lý bộ nhớ…(tt)

- Các hàm DLL được thực thi trong "ngữ cảnh" của tiểu trình (Thread) gọi hàm, do đó:
  - DLL sử dụng Stack của tiểu trình
  - DLL sử dụng không gian địa chỉ ảo của tiến trình gọi
  - DLL cấp phát bộ nhớ cho những biến động trên không gian địa chỉ ảo của tiến trình gọi

Xây dựng DLL

- Xây dựng DLL bằng thư viện Win32 API
- Xây dựng DLL bằng thư viện MFC

Xây dựng DLL – Dùng thư viện Win32 API

- Ví dụ tạo lập DLL
- Phân tích các khai báo
- Phân tích hàm DllMain
- Ví dụ liên kết DLL với 1 ứng dụng

Xây dựng DLL – Dùng thư viện Win32 API…(tt)

- Ví dụ tạo lập DLL
  - Chọn menu File → New
  - Chọn tab Projects
  - Chọn loại project “Win32 Dynamic Link Library”
  - Đặt tên project và xác định đường dẫn thư mục trong ô “Location”
  - Step 1: Chọn loại ứng dụng “A DLL that exports some symbols”
  - Nhấn Finish để kết thúc

Xây dựng DLL – Dùng thư viện Win32 API…(tt)

- Phân tích các khai báo

```
<DLL-Name>.h
#define DLLEXPORT  __declspec(dllexport)
#define DLLIMPORT  __declspec(dllimport)

// Ví dụ khai báo 1 biến "xuất khẩu"
DLLEXPORT int nDll=0;

// Ví dụ khai báo 1 hàm "xuất khẩu"
DLLEXPORT int  fnDll(void);
```

  - dllexport (xuất khẩu): cung cấp hàm, dữ liệu, tài nguyên,… cho các chương trình/DLL khác sử dụng

Xây dựng DLL – Dùng thư viện Win32 API…(tt)

- Phân tích các khai báo (tt)
  - dllimport (nhập khẩu): là load 1 hàm, hay dữ liệu, hay tài nguyên,… từ 1 DLL khác để sử dụng

```
<App-Name>.h
#define DLLIMPORT  __declspec(dllimport)

// Ví dụ khai báo 1 biến "nhập khẩu"
DLLIMPORT int nDll;

// Ví dụ khai báo 1 hàm "nhập khẩu"
DLLIMPORT int  fnDll(void);
```

Xây dựng DLL – Dùng thư viện Win32 API…(tt)

- Phân tích hàm DllMain

```
<DLL-Name>.cpp
BOOL APIENTRY DllMain( HANDLE hModule,
                       DWORD  ul_reason_for_call,
                       LPVOID lpReserved
                     )
{
    switch (ul_reason_for_call)
      {
            case DLL_PROCESS_ATTACH:
            case DLL_THREAD_ATTACH:
            case DLL_THREAD_DETACH:
            case DLL_PROCESS_DETACH:
                  break;
    }
    return TRUE;
}
```


Xây dựng DLL – Dùng thư viện Win32 API…(tt)

- Phân tích hàm DllMain (tt)
  - Hàm DllMain là hàm đầu vào chính của DLL
  - Hàm DllMain được gọi khi DLL được load vào bộ nhớ hoặc khi Windows yêu cầu DLL kết thúc (unload khỏi bộ nhớ)
  - Hàm DllMain có nhiệm vụ khởi tạo hoặc giải phóng các tài nguyên sử dụng cho DLL đó (nếu có)
  - Các tham số:
    - `hModule`: handle của DLL, do Windows tạo ra
    - `ul_reason_for_call`: nguyên nhân hàm DllMain được gọi

Xây dựng DLL – Dùng thư viện Win32 API…(tt)

- Phân tích hàm DllMain (tt)
  - **DLL_PROCESS_ATTACH:**
    - Hàm DllMain với tham số `DLL_PROCESS_ATTACH` được gọi khi process tiến hành load DLL
    - Thư viện DLL đang được Windows ánh xạ vào vùng nhớ của tiến trình (thực hiện lời gọi DLL).
    - Đây là cơ hội để DLL khởi tạo các biến, cấp phát vùng nhớ hay những thao tác cần thiết khác trước khi cho phép tiến trình gọi đến các hàm của thư viện
  - **DLL_PROCESS_DETATCH:**
    - Thư viện DLL được giải phóng khỏi vùng nhớ của tiến trình do 1 trong 3 nguyên nhân: nạp DLL không thành công, tiến trình kết thúc, hay tiến trình gọi hàm `FreeLibrary`
    - Đây là cơ hội để giải phóng các biến hay tài nguyên mà DLL đã cấp phát

Xây dựng DLL – Dùng thư viện Win32 API…(tt)

- Phân tích hàm DllMain (tt)
  - **DLL_THREAD_ATTACH:**
    - Khi tiến trình tạo mới 1 tiểu trình (thread), Windows gọi hàm DllMain của tất cả các thư viện DLL đang được sử dụng với tiến trình đó
    - Đây là cơ hội để khởi tạo các biến dùng cho tiểu trình
    - Lưu ý rằng tình huống này chỉ xảy ra khi tiểu trình được tạo sau khi thư viện DLL đã load vào tiến trình, có nghĩa rằng nếu DLL được load bằng hàm `LoadLibrary` thì tất cả các tiểu trình hiện có (trong tiến trình) sẽ không gọi hàm DllMain với tham số này
  - **DLL_THREAD_DETACH:**
    - Khi 1 tiểu trình kết thúc, Windows gọi hàm DllMain của tất cả các thư viện DLL đang được sử dụng với tiến trình này
    - Đây là cơ hội để giải phóng các biến dùng cho tiểu trình

Xây dựng DLL – Dùng thư viện Win32 API…(tt)

Ví dụ liên kết DLL với 1 ứng dụng

Xây dựng DLL – Dùng thư viện MFC

- Ví dụ tạo lập DLL
- Phân tích các khai báo
- Định nghĩa hàm export
- Các dạng thư viện DLL của MFC

Xây dựng DLL – Dùng thư viện MFC…(tt)

- Ví dụ tạo lập DLL
  - Chọn menu File → New
  - Chọn tab Projects
  - Chọn loại project “MFC AppWizard (dll)”
  - Đặt tên project và xác định đường dẫn thư mục trong ô “Location”
  - Step 1: Chọn loại ứng dụng “Regular DLL using shared MFC DLL”
  - Nhấn Finish để kết thúc

Xây dựng DLL – Dùng thư viện MFC…(tt)

- Phân tích các khai báo
  - CMyDllApp: lớp kế thừa từ lớp CWinApp, quản lý toàn bộ DLL

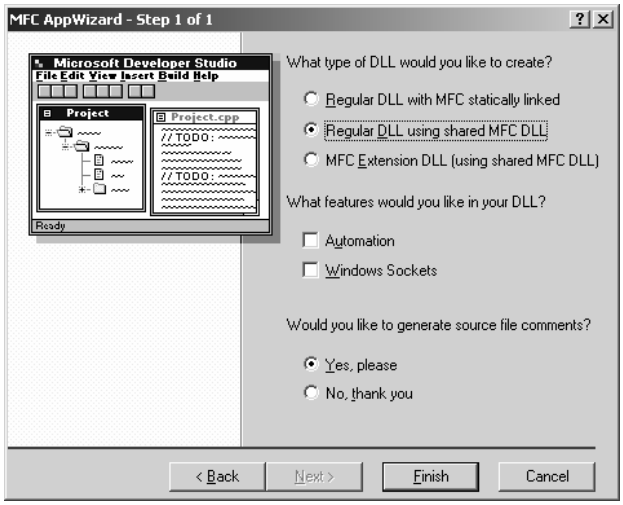
```
class CMyDllApp : public CWinApp
{
     public:
          CMyDllApp();
          DECLARE_MESSAGE_MAP()
};
```

Xây dựng DLL – Dùng thư viện MFC…(tt)

- Định nghĩa hàm export

```
<DLL-Name>.cpp
#define DLLEXPORT   __declspec(dllexport)
#define DLLIMPORT   __declspec(dllimport)

// Ví dụ khai báo 1 hàm "xuất khẩu"
DLLEXPORT int  fnDll(void) {
      AFX_MANAGE_STATE(AfxGetStaticModuleState());
      // Các lệnh của hàm …
      ….
      ….
}
```


Xây dựng DLL – Dùng thư viện MFC…(tt)



Các dạng thư viện DLL của MFC

Xây dựng DLL – Dùng thư viện MFC…(tt)

- Các dạng thư viện DLL của MFC
  - Regular DLL:
    - Chỉ export các hàm theo dạng C-style, không thể export class, hàm thành phần của class, các hàm định nghĩa chồng (overloaded function)
    - Các ứng dụng Win32 và MFC đều có thể dùng với loại DLL này
    - "with MFC statically linked": DLL sẽ được liên kết với các DLL chuẩn của MFC theo cách liên kết tĩnh.
    - "using shared MFC DLL": DLL sẽ được liên kết với các DLL chuẩn của MFC theo cách liên kết động.
  - MFC Extention DLL:
    - Cho phép export các class. Ứng dụng khác có thể tạo các object từ class, hoặc xây dựng các lớp kế thừa từ class
    - DLL sẽ được liên kết với các DLL chuẩn của MFC theo cách liên kết động.
    - Chỉ có các ứng dụng MFC mới có thể dùng với loại DLL này

Cách thức gọi DLL trong ứng dụng

- Cách thức tìm kiếm file DLL
- Load-time Dynamic Linking
- Run-time Dynamic Linking

Cách thức tìm kiếm file DLL

- Hệ điều hành Windows sẽ tìm file DLL trong các thư mục sau:
  - Nơi chứa chương trình ứng dụng gọi DLL
  - Thư mục hiện hành
  - Thư mục system, system32
  - Thư mục Windows
  - Các thư mục được liệt kê trong biến môi trường PATH

Cách thức gọi DLL trong ứng dụng – Load-time

- Ứng dụng sẽ gọi hàm export của DLL một cách "tường minh" (giống như cách gọi hàm của thư viện liên kết tĩnh)
- Ứng dụng cần phải liên kết với file thư viện nhập (Import Lib) của DLL
- Ứng dụng sẽ nạp DLL ngay thời điểm đầu tiên chương trình chạy
- ***Ưu điểm:***
  - Đơn giản, dễ sử dụng vì giống như cách dùng thư viện tĩnh
- ***Khuyết điểm:***
  - Nếu không tìm ra DLL lúc nạp, ứng dụng sẽ kết thúc ngay

Cách thức gọi DLL trong ứng dụng – Load-time…(tt)

- Cách làm:
  - Add file thư viện nhập (`DllName.lib`) vào project
  - Khai báo các hàm, biến IMPORT từ DLL
  - Gọi hàm của thư viện DLL như gọi hàm bình thường

Cách thức gọi DLL trong ứng dụng – Run-time

- Cách làm:
  - Ứng dụng sẽ dùng hàm `LoadLibrary` hay `LoadLibraryEx` để nạp DLL tại thời điểm cần thiết
  - Sau khi DLL được nạp, ứng dụng sẽ dùng hàm `GetProcAddress` để lấy địa chỉ của hàm export trong DLL
  - Ứng dụng sẽ gọi hàm export trong DLL bằng cách dùng con trỏ hàm trả về từ hàm `GetProcAddress`
  - Cách này không cần dùng đến file thư viện nhập (Import Lib.)

Cách thức gọi DLL trong ứng dụng – Run-time…(tt)

- Ưu điểm:
  - Có thể xử lý lỗi không tìm thấy DLL, tránh kết thúc ứng dụng tức thời, thể hiện thông báo lỗi do ứng dụng qui định

Cách thức gọi DLL trong ứng dụng – Run-time…(tt)

- Ví dụ: gọi hàm `myFunc(LPTSTR)` từ thư viện `MYDLL.DLL`

```
// Định nghĩa Prototype của hàm
typedef VOID (*MYPROC)(LPTSTR);
HINSTANCE hinstLib;
MYPROC ProcAddr;
// Nạp DLL và lấy handle của DLL module
hinstLib = LoadLibrary("MYDLL");
// Nếu load thành công, lấy địa chỉ của hàm myFunc
// trong DLL
if (hinstLib != NULL) {
    ProcAddr = (MYPROC) GetProcAddress(hinstLib,
                                       "myFunc");
    // Nếu lấy được địa chỉ hàm, gọi thực hiện hàm
    if (ProcAddr != NULL)
          (ProcAddr) ("A parameter string \n");
    // Giải phóng thư viện DLL
    FreeLibrary(hinstLib);
}
```

Cám ơn - Hỏi & Đáp

Lập trình C trên Windows

Kỹ thuật lập trình Visual C++ (MFC)

Nguyễn Tri Tuấn
Khoa CNTT – ĐH.KHTN.Tp.HCM
Email: nttuan@ fit.hcmuns.edu.vn

Nội dung

- Giới thiệu về MFC
- Chương trình MFC đầu tiên
- Xử lý Mouse và Keyboard
- Xử lý menu
- Toolbar, Statusbar
- Các Control
- Xây dựng và xử lý hộp thoại (Dialog box)
- Documents và Views: Scroll view, List view, Tree view
- SDI – Single Document Interface
- MDI - Multi Document Interface

Giới thiệu về MFC

- MFC là gì ?
- Một số tính năng của MFC qua từng version
- Các thành phần của 1 ứng dụng trong VC++
- Các màn hình giao diện chính của VC++ 6

Giới thiệu về MFC – MFC là gì ?

- Microsoft Foundation Class
- Là một thư viện các lớp (class, OOP) trong ngôn ngữ Visual C++, dùng cho việc lập trình trên Windows
- Được xây dựng trên cơ sở các hàm thư viện API của Windows
- Version 6 có khoảng 200 class
- Giúp cho người lập trình có thể xây dựng ứng dụng nhanh và ít tốn công sức hơn so với việc sử dụng đơn thuần các hàm thư viện API của Windows
- Ta vẫn có thể gọi các hàm Windows API trong MFC

Giới thiệu về MFC – MFC là gì ?…(tt)

- Trong 1 ứng dụng MFC, ta thường không gọi hàm Windows API trực tiếp, mà sẽ tạo các object từ những lớp của MFC, và gọi phương thức của object đó
- Đa số các phương thức của MFC class có cùng tên với những hàm Windows API
- MFC tạo ra một *Application Framework*, giúp:
  - Thiết lập kiến trúc của ứng dụng một cách nhất quán và khoa học
  - Che dấu đi nhiều phần chi tiết mà Windows API đòi hỏi, giúp developer "thảnh thơi" hơn

Giới thiệu về MFC - Một số tính năng của MFC

- Version 1:
  - Các lớp List, Array, String, Time, Date, File access,…
  - Các lớp giao diện cơ bản
  - MDI, OLE 1.0
- Version 2:
  - File open, save
  - Print preview, printing
  - Scrolling window, Splitter window
  - Toolbar, Statusbar
  - Truy xuất được đến các control của VB
  - Trợ giúp theo ngữ cảnh (Context-sensitive help)
  - DLL

Giới thiệu về MFC - Một số tính năng của MFC…(tt)

- Version 2.5:
  - Hỗ trợ ODBC (Open Database Connectivity), cho phép truy xuất đến các CSDL Access, FoxPro, SQL Server,…
  - OLE 2.01
- Version 3:
  - Hỗ trợ tab dialog (property sheet)
  - Docking control bar
- Version 3.1:
  - Hỗ trợ các control chuẩn của Windows 95
  - ODBC level 2 with Access Jet database engine
  - Các lớp Winsock phục vụ lập trình TCP/IP

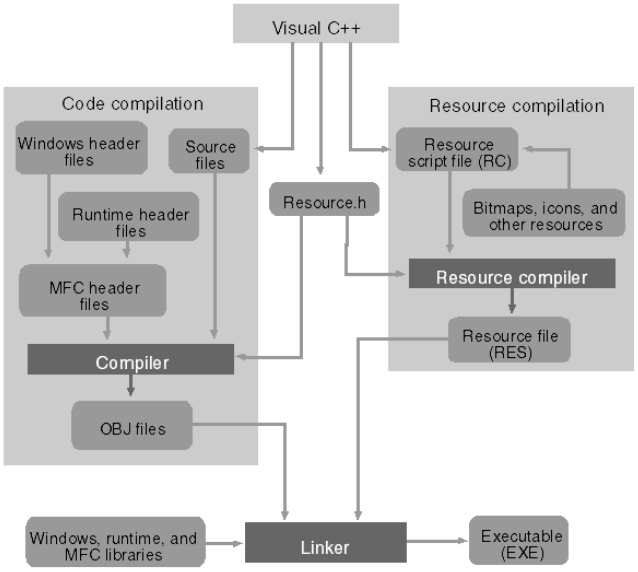
Giới thiệu về MFC - Một số tính năng của MFC…(tt)

- Version 4.0:
  - ADO (Data Access Object)
  - Windows 95 docking control bar
  - Bổ sung thêm lớp TreeView và RichEdit
  - Các lớp đồng bộ hoá các tiểu trình
- Version 4.2:
  - Các lớp WinInet
  - Các lớp ActiveX document server
  - Các tính năng mở rộng của ActiveX control
  - Tăng cường một số khả năng của ODBC

Giới thiệu về MFC - Một số tính năng của MFC…(tt)

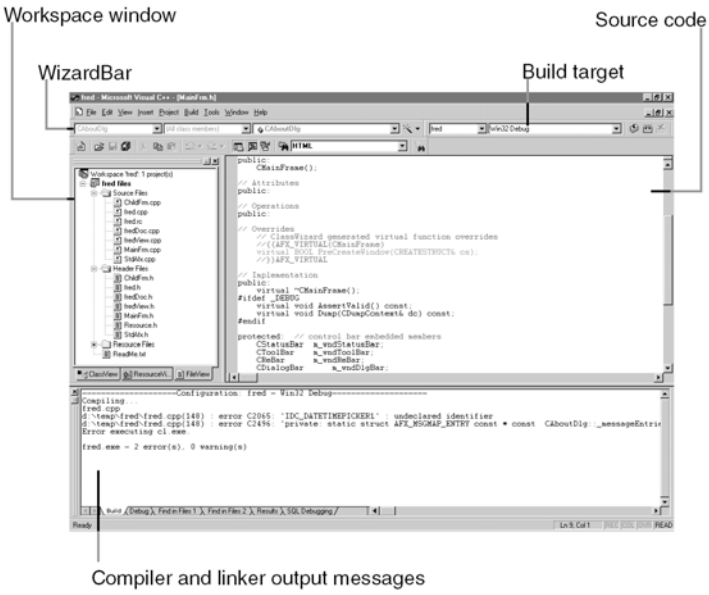
- Version 6:
  - Hỗ trợ các lớp cho những control chuẩn trong IE 4.0
  - Hỗ trợ Dynamic HTML, cho phép tạo lập động các trang HTML
  - Active Document Containment, cho phép ứng dụng MFC có thể chứa các Active Document
  - OLE DB và ADO

Giới thiệu về MFC - Các thành phần của 1 ứng dụng



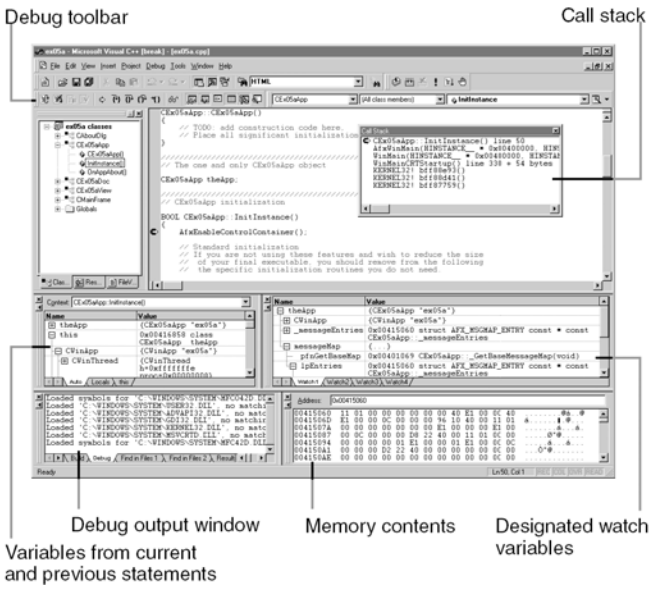
Sơ đồ biên dịch các thành phần của 1 ứng dụng trong VC++

Giới thiệu về MFC - Các màn hình giao diện chính



Các thành phần giao diện chính của VC++

Giới thiệu về MFC - Các màn hình giao diện chính…(tt)



Các thành phần giao diện chính của VC++ (run-time)

Chương trình MFC đầu tiên

- Ứng dụng đơn giản dùng Application Framework
- Ứng dụng phức tạp hơn (Dialog-based App)

Chương trình MFC đầu tiên - Ứng dụng đơn giản

- Tạo ứng dụng
- Các thành phần của chương trình

Ứng dụng đơn giản - Tạo ứng dụng

- Chọn menu File → New
- Chọn tab Projects
- Chọn loại project “Win32 Application”
- Đặt tên project và xác định đường dẫn thư mục trong ô “Location”
- Step 1: Chọn loại ứng dụng “An empty project”
- Nhấn Finish để kết thúc
- Add các file Hello.h và Hello.cpp vào project
- Chọn menu Project → Settings
  - Chọn project trong cửa sổ bên trái
  - Chọn tab General
  - Chọn “Use MFC In A Shared DLL”

Ứng dụng đơn giản - Tạo ứng dụng…(tt)

Hello.h

```
class CMyApp : public CWinApp {
        public:
                virtual BOOL InitInstance ();
};
class CMainWindow : public CFrameWnd {
        public:
                CMainWindow ();
        protected:
                afx_msg void OnPaint ();
                DECLARE_MESSAGE_MAP ()
};
```

Ứng dụng đơn giản - Tạo ứng dụng…(tt)

Hello.cpp

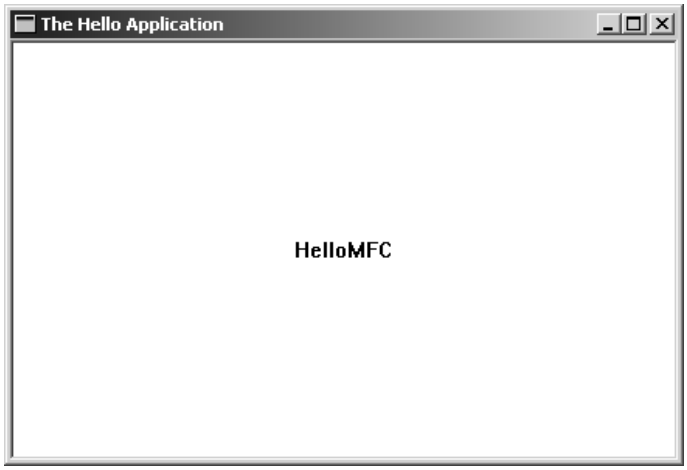
```
#include <afxwin.h>
#include "Hello.h"
CMyApp myApp;
//////////////////////////////////////////////////////////////////////////
// CMyApp member functions
BOOL CMyApp::InitInstance () {
        m_pMainWnd = new CMainWindow;
        m_pMainWnd->ShowWindow (m_nCmdShow);
        m_pMainWnd->UpdateWindow ();
        return TRUE;
}
//////////////////////////////////////////////////////////////////////////
// CMainWindow message map and member functions
BEGIN_MESSAGE_MAP (CMainWindow, CFrameWnd)
        ON_WM_PAINT ()
END_MESSAGE_MAP ()
```


Ứng dụng đơn giản - Tạo ứng dụng…(tt)

Hello.cpp…(tt)

```
CMainWindow::CMainWindow () {
        Create (NULL, _T ("The Hello Application"));
}

void CMainWindow::OnPaint () {
        CPaintDC dc (this);
        CRect rect;
        GetClientRect (&rect);
        dc.DrawText (_T ("Hello, MFC"), -1, &rect,
                DT_SINGLELINE ¦ DT_CENTER ¦
                DT_VCENTER);
}
```

Ứng dụng đơn giản - Tạo ứng dụng…(tt)



Ứng dụng MFC đơn giản

Ứng dụng đơn giản – Các thành phần của c.trình

- CWinApp: lớp chính của MFC để quản lý ứng dụng. Chứa đựng vòng lặp nhận message và phân phối message đến các cửa sổ của ứng dụng
- CMyApp: lớp kế thừa từ lớp CWinApp

```
class CMyApp : public CWinApp {
    public:
          virtual BOOL InitInstance ();
};
```

- InitInstance(): hàm khởi tạo ứng dụng, override lên hàm chuẩn của lớp CWinApp

```
BOOL CMyApp::InitInstance () {
    m_pMainWnd = new CMainWindow;
    m_pMainWnd->ShowWindow (m_nCmdShow);
    m_pMainWnd->UpdateWindow ();
    return TRUE;
}
```

Ứng dụng đơn giản – Các thành phần của c.trình…(tt)

- CWnd: lớp chính của MFC để quản lý các loại cửa sổ giao diện. Có nhiều lớp được kế thừa từ lớp này để quản lý các loại cửa sổ khác nhau (CTreeCtrl, CListBox, Cedit,…)
- CFrameWnd: lớp kế thừa từ lớp CWnd, để quản lý cửa sổ giao diện chính của ứng dụng
- CMainWindow: lớp kế thừa từ lớp CFrameWnd

```
class CMainWindow : public CFrameWnd {
    public:
          CMainWindow ();
    protected:
          afx_msg void OnPaint ();
          DECLARE_MESSAGE_MAP ()
};
```


Ứng dụng đơn giản – Các thành phần của c.trình…(tt)

- CMainWindow(): hàm khởi tạo cửa sổ giao diện của ứng dụng, override lên hàm chuẩn của lớp CFrameWnd

```
CMainWindow::CMainWindow () {
  Create (NULL, _T ("The Hello Application"));
}
```

- OnPaint: hàm thành phần của lớp CMainWindow, được gọi khi cần cập nhật nội dung cửa sổ. Hàm này được định nghĩa chồng lên hàm chuẩn của lớp CFrameWnd.

```
void CMainWindow::OnPaint () {
  CPaintDC   dc(this);
  CRect              rect;
  GetClientRect (&rect);
  dc.DrawText (_T ("Hello, MFC"), -1, &rect,
              DT_SINGLELINE ¦ DT_CENTER ¦
              DT_VCENTER);
}
```

Ứng dụng đơn giản – Các thành phần của c.trình…(tt)

- Message Map:
  - Làm sao để xử lý 1 message ?
  - MFC dùng Message Map để liên kết các message với những hàm thành phần của lớp cửa sổ
  - Mỗi message sẽ được xử lý bởi 1 hàm thành phần tương ứng

```
BEGIN_MESSAGE_MAP (CMainWindow, CFrameWnd)
      ON_WM_PAINT ()
END_MESSAGE_MAP ()
```

  - ON_WM_PAINT là 1 macro được định nghĩa trong Afxmsg_.h, mặc nhiên liên kết message WM_PAINT với hàm OnPaint

Ứng dụng đơn giản – Các thành phần của c.trình…(tt)

- Xử lý thêm message WM_LBUTTONDOWN
  - Bổ sung thêm 1 hàm thành phần vào khai báo của lớp CMainWindow:

```
afx_msg void OnLButtonDown(UINT nFlags,
                           CPoint point);
```

  - Bổ sung thêm 1 macro vào khai báo Message Map:

```
ON_WM_LBUTTONDOWN ()
```

  - Định nghĩa hàm thành phần **OnLButtonDown**:

```
void CMainWindow::OnLButtonDown(UINT nFlags,
                                CPoint point)
{
       MessageBox("Left button clicked !",
                  "Mouse", MB_OK);
}
```

Ứng dụng đơn giản – Các thành phần của c.trình…(tt)

- Xử lý thêm message WM_MOUSELEAVE
  - Bổ sung thêm 1 hàm thành phần vào khai báo của lớp CMainWindow:

```
afx_msg LRESULT OnMouseLeave();
```

  - Bổ sung thêm 1 macro vào khai báo Message Map:

```
ON_MESSAGE (WM_MOUSELEAVE, OnMouseLeave)
```

  - Định nghĩa hàm thành phần **OnLButtonDown**:

```
LRESULT CMainWindow::OnMouseLeave()
{
   MessageBox("Mouse leaved !", "Mouse", MB_OK);
   return 0;
}
```

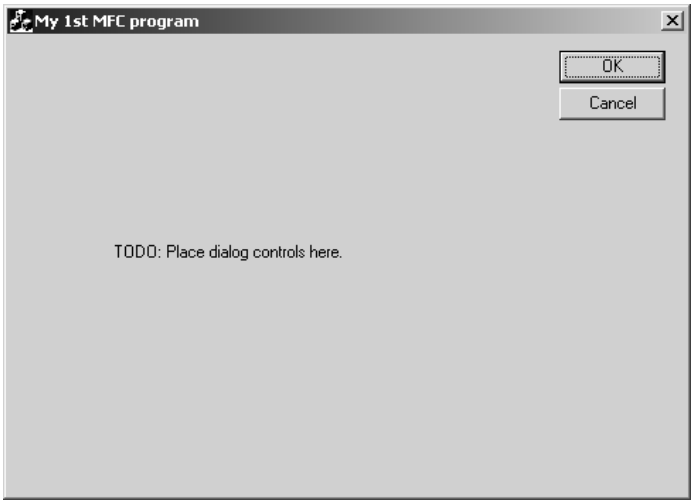

Chương trình MFC đầu tiên – Dialog-based App

- Tạo ứng dụng bằng cách sử dụng MFC AppWizard
- Các thành phần của chương trình

Dialog-based App - Tạo ứng dụng bằng MFC AppWizard

- Chọn menu File → New
- Chọn tab Projects
- Chọn loại project “MFC AppWizard (exe)”
- Đặt tên project và xác định đường dẫn thư mục trong ô “Location”
- Step 1: Chọn loại ứng dụng “Dialog-based”
- Step 2: Chỉ chọn option “3D controls”. Gõ tiêu đề của ứng dụng vào ô “Enter a title…”
- Step 3: chọn theo chế độ mặc định
- Nhấn Finish để kết thúc

Dialog-based App - Tạo ứng dụng bằng MFC AppWizard…(tt)



Ứng dụng MFC (Dialog-based)

Dialog-based App - Các thành phần của chương trình

- Các file chương trình: (xxx là tên project)
  - xxx.h: header file của file xxx.cpp, chứa khai báo lớp CxxxApp để quản lý toàn bộ ứng dụng. Lớp CxxxApp kế thừa từ lớp CWinApp của MFC
  - xxxDlg.h: header file của file xxxDlg.cpp, chứa khai báo lớp CxxxDlg để quản lý cửa sổ Dialog giao diện của ứng dụng. Lớp CxxxDlg kế thừa từ lớp CDialog của MFC
  - Resource.h: header file, chứa các hằng ID của các resource được định nghĩa trong file xxx.rc
  - xxxDlg.cpp: cài đặt các hàm thành phần của lớp CxxxDlg
  - xxx.cpp: cài đặt các hàm thành phần của lớp CxxxApp
  - xxx.rc: mô tả các resource (tài nguyên) của ứng dụng

Dialog-based App- Các thành phần của chương trình…(tt)

- Lớp CxxxDlg:
  - Trong ứng dụng Dialog-based, cửa sổ giao diện chính là 1 Dialog, nên ứng dụng dùng lớp CxxxDlg thay vì lớp CMainWindow

```
class CxxxDlg : public CDialog
{
 public:
        CxxxDlg(CWnd* pParent = NULL);
        enum { IDD = IDD_xxx_DIALOG };
 protected:
        virtual void DoDataExchange(CDataExchange* pDX);
 protected:
        HICON m_hIcon;
 virtual BOOL OnInitDialog();
 afx_msg void OnPaint();
 afx_msg HCURSOR OnQueryDragIcon();
 DECLARE_MESSAGE_MAP()
};
```

Xử lý Mouse và Keyboard

- Xử lý mouse
  - Thông điệp của mouse
  - Ví dụ: Vẽ hình bằng mouse
- Xử lý keyboard
  - Thông điệp của keyboard
  - Ví dụ: Xử lý phím nhấn

Xử lý Mouse

- Thông điệp của mouse
  - WM_LBUTTONDOWN
  - WM_LBUTTONUP
  - WM_LBUTTONDBLCLK
  - WM_RBUTTONDOWN
  - WM_RBUTTONUP
  - WM_RBUTTONDBLCLK
  - WM_MOUSEMOVE
  - WM_MOUSEWHEEL

Xử lý Mouse…(tt)

- Thông điệp của mouse (tt)
  - Với mỗi thông điệp của mouse, Windows gởi kèm 2 tham số wParam và lParam
  - wParam: cho biết phím nào đang được nhấn (Ctrl, Shift)
  - lParam: cho biết toạ độ hiện tại
    - LOWORD(lParam): tọa độ x
    - HIWORD(lParam): tọa độ y

Xử lý Mouse…(tt)

- Ví dụ: Vẽ hình bằng mouse
  - *Mô tả:* khi user nhấn giữ nút trái chuột & di chuyển → vẽ 1 đường thẳng
  - Các xử lý cần thiết:
    - WM_LBUTTONDOWN ⇔ OnLButtonDown
    - WM_MOUSEMOVE ⇔ OnMouseMove
  - Các bước thực hiện:
    - Định nghĩa 2 biến m_PrevX, m_PrevY trong class CxxxDlg
    - Định nghĩa hàm xử lý message WM_LBUTTONDOWN trong class CxxxDlg
    - Định nghĩa hàm xử lý message WM_MOUSEMOVE trong class CxxxDlg

Xử lý Mouse…(tt)

- Vẽ hình bằng mouse…(tt)

```
void CxxxDlg::OnLButtonDown(UINT nFlags,
                            CPoint point)
{
    // TODO: Add your message handler code here
    // and/or call default
    m_PrevX = point.x;
    m_PrevY = point.y;

    CDialog::OnLButtonDown(nFlags, point);
}
```


Xử lý Mouse…(tt)

- Vẽ hình bằng mouse…(tt)

```
void CxxxDlg::OnMouseMove(UINT nFlags, CPoint point)
{
    // TODO: Add your message handler code here
    if ((nFlags & MK_LBUTTON) == MK_LBUTTON) {
       // Get the Device Context
       CClientDC dc(this);

       // Draw a line from the prev point to current point
       dc.MoveTo(m_StartX, m_StartY);
       dc.LineTo(point.x, point.y);

       // Save the current point as the previous point
       m_PrevX = point.x;
       m_PrevY = point.y;
    }

    CDialog::OnMouseMove(nFlags, point);
}
```

Xử lý keyboard

- Thông điệp của keyboard
  - WM_KEYDOWN / WM_KEYUP: phát sinh khi 1 phím (không phải là phím hệ thống) được nhấn xuống/thả ra
    - Hàm xử lý tương ứng: CWnd::OnKeyDown, CWnd::OnKeyUp
    - wParam: virtual-key code
    - lParam: chứa các thông tin khác (số lần lặp lại phím, scan code, extended key,…)
  - WM_CHAR: là kết quả phát sinh do message WM_KEYDOWN, báo hiệu 1 ký tự in được (printed character) đã được tạo ra
    - Hàm xử lý tương ứng: CWnd::OnChar
    - wParam: mã ký tự
    - lParam: chứa các thông tin khác (số lần lặp lại do nhấn giữ phím, có phím Alt nhấn kèm,…)

Xử lý keyboard…(tt)

- Ví dụ: Xử lý phím nhấn
  - *Mô tả:* khi user nhấn một phím → hiển thị 1 MessageBox thông báo
  - Các xử lý cần thiết
    - WM_KEYDOWN ⇔ OnKeyDown
  - Các bước thực hiện
    - Định nghĩa hàm xử lý message WM_KEYDOWN trong class CxxxDlg

Xử lý menu

- Một vài khái niệm
- Tạo lập menu
- Load và hiển thị menu
- Xử lý khi menu item được chọn
- Thay đổi trạng thái menu
- Ví dụ

Xử lý menu - Một vài khái niệm

- Menu bar: thanh menu. Bao gồm nhiều drop-down menu và menu item
- Drop-down menu: một phần của menu bar, chứa các menu item hoặc các drop-down menu khác. VD. File, Edit, …
- Menu item: tương ứng với 1 lệnh của chương trình. Mỗi menu item được xác định bằng 1 số nguyên phân biệt, gọi là item ID hay command ID. VD. Open, Save, …
- Popup menu: giống như drop-down menu, nhưng có thể xuất hiện ở vị trí bất kỳ trên màn hình (thường khi nhấn nút phải mouse)
- System menu: chứa các lệnh hệ thống điều khiển cửa sổ. VD. Minimize, Maximize, Close, …

Xử lý menu - Tạo lập menu

- Thường có 2 cách chính để tạo menu:
  - Tạo menu ở dạng resource của ứng dụng, và load vào khi chạy
  - Tạo trực tiếp bằng các hàm khi ứng dụng đang chạy.
    - Lớp sử dụng để quản lý menu: CMenu
    - Các hàm thành phần: CreateMenu, InsertMenu, …

Xử lý menu - Tạo lập menu…(tt)

xxx.rc

```
IDR_MAINFRAME MENU PRELOAD DISCARDABLE
BEGIN
        POPUP "&File"
        BEGIN
                MENUITEM "&New\tCtrl+N", ID_FILE_NEW
                MENUITEM "&Open...\tCtrl+O", ID_FILE_OPEN
                MENUITEM SEPARATOR
                MENUITEM "E&xit", ID_APP_EXIT
        END
        POPUP "&Edit"
        BEGIN
                MENUITEM "&Undo\tCtrl+Z", ID_EDIT_UNDO
                MENUITEM SEPARATOR
                MENUITEM "Cu&t\tCtrl+X", ID_EDIT_CUT
                MENUITEM "&Copy\tCtrl+C", ID_EDIT_COPY
                MENUITEM "&Paste\tCtrl+V", ID_EDIT_PASTE
        END
END
```

Xử lý menu - Load và hiển thị menu

- Xác định menu bar khi tạo cửa sổ:

```
Create(NULL, _T("My Application"),
    WS_OVERLAPPEDWINDOW, rectDefault, NULL,
    MAKEINTRESOURCE(IDR_MAINFRAME));
```

- Thay đổi menu bar:

```
CMenu menu;
menu.LoadMenu(IDR_MAINFRAME);
SetMenu(&menu);
menu.Detach();
```


Xử lý menu - Load và hiển thị menu…(tt)

- MAKEINTRESOURCE: macro dùng để chuyển đổi 1 số nguyên (resource ID) thành dạng LPSTR
- CMenu::LoadMenu: load 1 resource menu bar và gán vào đối tượng CMenu
- CWnd::SetMenu: gán menu bar cho 1 cửa sổ
- CMenu::Detach: gỡ bỏ menu bar ra khỏi đối tượng CMenu, để menu bar không bị huỷ bỏ cùng với đối tượng CMenu khi ra khỏi phạm vi khai báo

Xử lý menu - Xử lý khi menu item được chọn

- Các thông điệp của menu
- Xử lý lệnh của menu item
- Nhóm lệnh (Command range)

Xử lý menu - Xử lý khi menu item được chọn…(tt)

- Các thông điệp của menu:
  - WM_MENUSELECT: phát sinh khi user tác động lên menu. Thông điệp này có thể dùng để cập nhật trạng thái của menu (trường hợp menu thay đổi theo ngữ cảnh – Context-sensitive Menu)
    - Hàm xử lý tương ứng: CWnd::OnMenuSelect
    - wParam:
      - LOWORD(wParam): ID của menu item hoặc index của menu popup
      - HIWORD(wParam): các thông tin khác (trạng thái menu, loại menu, …)
    - lParam: handle của menu

Xử lý menu - Xử lý khi menu item được chọn…(tt)

- Các thông điệp của menu: (tt)
  - WM_COMMAND: phát sinh khi user chọn 1 menu item
    - Hàm xử lý tương ứng: CWnd::OnCommand
    - wParam:
      - LOWORD(wParam): ID của menu item hoặc của control
      - HIWORD(wParam): nguồn gốc phát sinh, 1 nếu sinh ra do 1 phím tắt; 0 nếu chọn trực tiếp từ menu
    - lParam:
      - NULL nếu message này phát sinh từ menu
      - Nếu message phát sinh từ 1 control, lParam sẽ chứa handle của control đó

Xử lý menu - Xử lý khi menu item được chọn…(tt)

- Xử lý lệnh của menu item
  - Dựa trên message WM_COMMAND
  - Định nghĩa message map

```
ON_COMMAND (ID_FILE_OPEN, OnMyFileOpen)
ON_COMMAND (ID_FILE_EXIT, OnMyFileExit)
```

  - Viết hàm thành phần xử lý cho menu item tương ứng

```
void CMainFrame::OnMyFileOpen () {
 // Thực hiện thao tác mở file
 …
}
void CMainFrame::OnMyFileExit () {
 PostMessage (WM_CLOSE, 0, 0);
}
```

Xử lý menu - Xử lý khi menu item được chọn…(tt)

- Nhóm lệnh (Command range)
  - Là 1 nhóm menu item hoạt động theo nguyên tắc “Chỉ có 1 phần tử được chọn tại 1 thời điểm”
  - VD. Chức năng vẽ hình “Line / Circle / Rectangle”
  - Cách thức xử lý ?
    - Cách 1: map tất cả xử lý của các menu item này vào chung 1 hàm xử lý
    - Cách 2: dùng macro ON_COMMAND_RANGE

Xử lý menu - Xử lý khi menu item được chọn…(tt)

- Nhóm lệnh (Command range) (tt)
  - Cách 1: map tất cả xử lý của các menu item này vào chung 1 hàm xử lý

```
// Định nghĩa Message map
ON_COMMAND (ID_DRAW_LINE, OnDraw)
ON_COMMAND (ID_DRAW_CIRCLE, OnDraw)
ON_COMMAND (ID_DRAW_RECTANGLE, OnDraw)

// Hàm xử lý chung, xác định item hiện hành
void CMainFrame::OnDraw () {
   m_nCurrentDraw =
      (UINT) LOWORD(GetCurrentMessage()->wParam);
}
```

Xử lý menu - Xử lý khi menu item được chọn…(tt)

- Nhóm lệnh (Command range) (tt)
  - Cách 2: dùng macro ON_COMMAND_RANGE

```
// Định nghĩa Message map
ON_COMMAND_RANGE (ID_DRAW_LINE,
                  ID_DRAW_RECTANGLE, OnDraw)

// Hàm xử lý chung, xác định item hiện hành
void CMainFrame::OnDraw (UINT nID) {
      m_nCurrentDraw = nID;
}
```


Xử lý menu - Thay đổi trạng thái menu

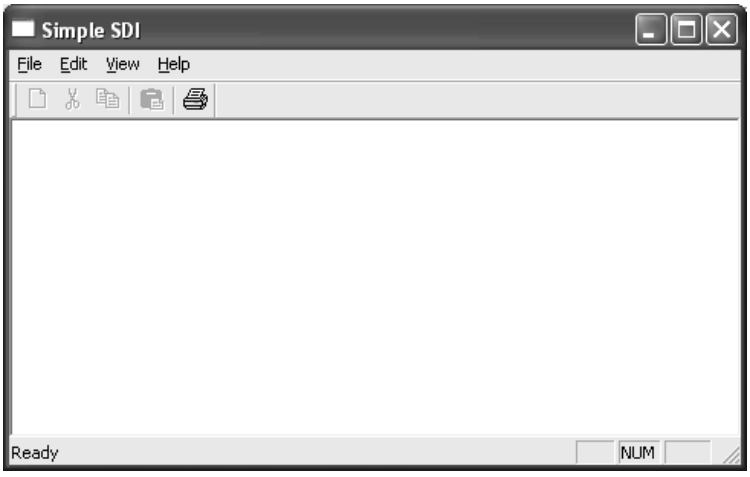
- Các ví dụ:
  - Khi user chọn chức năng vẽ Circle → cần thể hiện 1 dấu check (☑) phía trước
  - Chức năng Cut/Copy/Delete chỉ được kích hoạt khi user đánh dấu chọn 1 đoạn text
  - Chức năng Paste chỉ được kích hoạt khi clipboard khác rỗng
  - ...
- Cách xử lý

```
void CMainFrame::OnDraw(UINT nID) {
    CMenu* pMenu = GetMenu();
    pMenu->CheckMenuItem(m_nCurrentDraw,MF_UNCHECKED);
    m_nCurrentDraw = nID;
    pMenu->CheckMenuItem(m_nCurrentDraw, MF_CHECKED);
}
```

Xử lý menu – Ví dụ

- Tạo 1 ứng dụng SDI
  - Chọn menu File → New
  - Chọn tab Projects
  - Chọn loại project “MFC AppWizard (exe)”
  - Đặt tên project và xác định đường dẫn thư mục trong ô “Location”
  - Step 1: Chọn loại ứng dụng “Single Document”, bỏ option “Document/View architecture support”
  - Nhấn Finish để kết thúc

Xử lý menu – Ví dụ…(tt)



Xử lý menu – Ví dụ…(tt)

- Xử lý lệnh của menu item
  - Vẽ thêm vào menu popup File các item: New, Open, Save
  - Định nghĩa Message Map cho các hàm xử lý item

```
ON_COMMAND(ID_FILE_NEW, OnFileNew)
ON_COMMAND(ID_FILE_OPEN, OnFileOpen)
ON_COMMAND(ID_FILE_SAVE, OnFileSave)
```


Xử lý menu – Ví dụ…(tt)

- Xử lý lệnh của menu item (tt)
  - Viết xử lý lệnh cho từng item

```
void CMainFrame::OnFileNew()
{
   // TODO: Add your command handler code here
   MessageBox("Ban vua chon item New", "File");
}
void CMainFrame::OnFileOpen()
{
   // TODO: Add your command handler code here
   MessageBox("Ban vua chon item Open", "File");
}
void CMainFrame::OnFileSave()
{
   // TODO: Add your command handler code here
   MessageBox("Ban vua chon item Save", "File");
}
```

Xử lý menu – Ví dụ…(tt)

- Xử lý chọn nhóm lệnh
  - Vẽ thêm menu popup Draw với các item: Line, Circle, Rectangle
  - Định nghĩa message map

```
ON_COMMAND_RANGE (ID_DRAW_LINE,
                  ID_DRAW_RECTANGLE, OnDraw)
```

  - Viết hàm xử lý

```
void CMainFrame::OnDraw(UINT nID) {
        CMenu* pMenu = GetMenu();
        pMenu->CheckMenuItem(m_nCurrentDraw,
                    MF_UNCHECKED);
        m_nCurrentDraw = nID;
        pMenu->CheckMenuItem(m_nCurrentDraw,
                    MF_CHECKED);
}
```

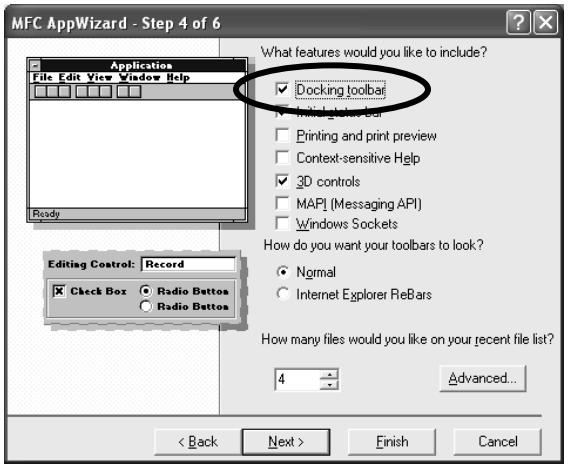

Toolbar

- Tạo một ứng dụng có Toolbar bằng AppWizard
- Tạo Toolbar bằng lớp CToolBar

Toolbar - Tạo một ứng dụng bằng AppWizard

- Tạo một ứng dụng có Toolbar bằng AppWizard
  - Chọn menu File → New
  - Chọn tab Projects
  - Chọn loại project “MFC AppWizard (exe)”
  - Đặt tên project và xác định đường dẫn thư mục trong ô “Location”
  - Step 1: Chọn loại ứng dụng “Single Document”, bỏ option “Document/View architecture support”
  - Nhấn Finish để kết thúc

Toolbar - Tạo một ứng dụng bằng AppWizard…(tt)



Chọn option này để AppWizard tự động tạo ra một Docking Toolbar

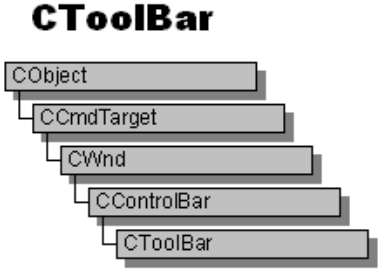
Toolbar - Tạo một ứng dụng bằng AppWizard…(tt)

- Các xử lý trong hàm OnCreate của lớp CMainFrame

```
if (!m_wndToolBar.CreateEx(this, TBSTYLE_FLAT,
          WS_CHILD | WS_VISIBLE | CBRS_TOP|
          CBRS_GRIPPER | CBRS_TOOLTIPS | CBRS_FLYBY |
          CBRS_SIZE_DYNAMIC) ||
    !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
{
    TRACE0("Failed to create toolbar\n");
    return -1;      // fail to create
}
// Xác định thuộc tính Docking
m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
EnableDocking(CBRS_ALIGN_ANY);
DockControlBar(&m_wndToolBar);
```


Toolbar - Tạo Toolbar bằng lớp CToolBar

- Tạo lập và hiển thị
- Ẩn/hiện thanh ToolBar
- Thêm các ToolTip và FlyBy text



Toolbar - Tạo Toolbar bằng lớp CToolBar…(tt)

- Tạo lập và hiển thị:
  - Bước 1: thiết kế DrawToolBar bằng RC editor, bao gồm các chức năng: Line, Circle, Rectangle, có ID là IDR_DRAWTOOLBAR

  - Bước 2: trong class CMainFrame, định nghĩa biến quản lý DrawToolBar

```
// class CMainFrame
CToolBar  m_wndDrawToolBar;
```


Toolbar - Tạo Toolbar bằng lớp CToolBar…(tt)

- Bước 3: trong hàm OnCreate của lớp CMainFrame, viết lệnh tạo lập DrawToolBar

```
// Trong hàm CMainFrame::OnCreate
if (!m_wndDrawToolBar.Create(this) ||
    !m_wndDrawToolBar.LoadToolBar(IDR_DRAWTOOLBAR))
{
    TRACE0("Khong the tao duoc DrawToolBat\n");
    return -1;
}
// Xác định tính chất của ToolBar
m_wndDrawToolBar.SetBarStyle(
    m_wndDrawToolBar.GetBarStyle() | CBRS_TOOLTIPS |
    CBRS_FLYBY | CBRS_SIZE_DYNAMIC);
// Xác định tính chất Docking
m_wndDrawToolBar.EnableDocking(CBRS_ALIGN_ANY);
// Docking toolbar
DockControlBar(&m_wndDrawToolBar);
```

Toolbar - Tạo Toolbar bằng lớp CToolBar…(tt)

- Ẩn/hiện thanh ToolBar
  - Cách thực hiện:
    - Thêm 1 menu item mới vào menu popup View, với ID là ID_VIEW_DRAWTOOLBAR
    - Viết hàm xử lý cho menu item này

```
void CMainFrame::OnViewDrawtoolbar()
{
  // TODO: Add your command handler code here
  BOOL  bVisible = m_wndDrawToolBar.GetStyle()
                        & WS_VISIBLE;
  ShowControlBar(&m_wndDrawToolBar, !bVisible,
                        FALSE);

  CMenu* pMenu = GetMenu();
  pMenu->CheckMenuItem(ID_VIEW_DRAWTOOLBAR,
    (!bVisible==1) ? MF_CHECKED : MF_UNCHECKED);
}
```


Toolbar - Tạo Toolbar bằng lớp CToolBar…(tt)

- Thêm các ToolTip và FlyBy text
  - ToolTip là 1 cửa sổ nhỏ chứa câu giải thích ngắn về công dụng của 1 button trên ToolBar

  - FlyBy text là 1 câu thông báo được hiển thị trên StatusBar khi user di chuyển mouse đến 1 button của ToolBar

Toolbar - Tạo Toolbar bằng lớp CToolBar…(tt)

- Thêm các ToolTip và FlyBy text (tt)
  - Cách thực hiện:
    - ToolBar phải có thuộc tính CBRS_TOOLTIPS ¦ CBRS_FLYBY
    - Tạo 1 bảng mô tả chuỗi (StringTable)
    - ID của chuỗi trùng với ID của các button trên ToolBar
    - Chuỗi có thể gồm 2 phần:

```
<FlyBy Text>\n<ToolTip>
VD.
STRINGTABLE DISCARDABLE
BEGIN
   ID_DRAW_LINE        "Draw a line\nLine"
   ID_DRAW_CIRCLE      "Draw a circle\nCircle"
   ID_DRAW_RECTANGLE   "Draw a rect\nRectangle"
END
```

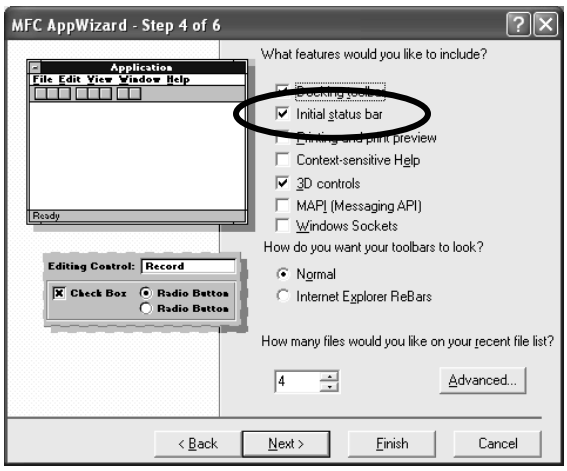

Statusbar

- Tạo một ứng dụng có Statusbar bằng AppWizard
- Tạo Statusbar bằng lớp CStatusBar

Statusbar - Tạo một ứng dụng bằng AppWizard

- Tạo một ứng dụng có Statusbar bằng AppWizard
  - Chọn menu File → New
  - Chọn tab Projects
  - Chọn loại project “MFC AppWizard (exe)”
  - Đặt tên project và xác định đường dẫn thư mục trong ô “Location”
  - Step 1: Chọn loại ứng dụng “Single Document”, bỏ option “Document/View architecture support”
  - Nhấn Finish để kết thúc

Statusbar - Tạo một ứng dụng bằng AppWizard…(tt)



Chọn option này để AppWizard tự động tạo ra một Statusbar

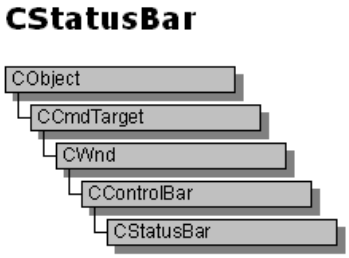
Statusbar - Tạo một ứng dụng bằng AppWizard…(tt)

- Các xử lý tương ứng

```
// Định nghĩa các vùng trên Statusbar
// (file MainFrm.cpp)
static UINT indicators[] =
{
    ID_SEPARATOR,           // status line indicator
    ID_INDICATOR_CAPS,
    ID_INDICATOR_NUM,
    ID_INDICATOR_SCRL,
};
// Tạo lập Statusbar (hàm OnCreate của lớp CMainFrame)
if (!m_wndStatusBar.Create(this) ||
    !m_wndStatusBar.SetIndicators(indicators,
            sizeof(indicators)/sizeof(UINT)))
{
    TRACE0("Không thể tạo được Statusbar\n");
    return -1;      // fail to create
}
```


Statusbar - Tạo Statusbar bằng lớp CStatusBar

- Tạo lập và hiển thị
- Ẩn/hiện Statusbar
- Thể hiện giúp đỡ cho các menu item
- Phân vùng trên Statusbar



Lập trình C trên Windows

Thư viện đồ họa GDI (Graphics Device Interface)

Nguyễn Tri Tuấn
Khoa CNTT – ĐH.KHTN.Tp.HCM
Email: nttuan@ fit.hcmuns.edu.vn

Nội dung

- Giới thiệu
- Các khái niệm cơ bản
- Các ví dụ vẽ hình đồ họa
- Xử lý ảnh Bitmap
- In ấn (Printing)

Giới thiệu

- GDI là thư viện cung cấp các hàm (functions) và các cấu trúc dữ liệu (structures) cần thiết để ứng dụng tạo ra những kết xuất dưới dạng đồ họa (hiển thị lên màn hình, in ra máy in,…)
- Các hàm GDI cho phép vẽ đường thẳng, đường cong, các hình đa giác, xuất ký tự, hiển thị ảnh bitmap, in ấn,…
- GDI không tương tác trực tiếp với thiết bị phần cứng, mà thông qua các driver

Các khái niệm cơ bản

- Thiết bị đồ họa (Graphics device)
- Ngữ cảnh của thiết bị (DC - Device Context)
- DC trong MFC
- Đối tượng vẽ (Drawing object)
- Đối tượng vẽ trong MFC

Các khái niệm cơ bản - Thiết bị đồ họa

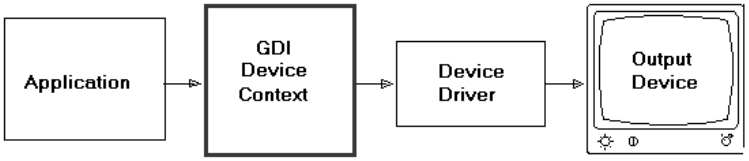
- Thiết bị đồ họa (Graphics device): là những thiết bị cho phép thể hiện các kết xuất dạng đồ họa trên đó
  - Thiết bị vật lý (physical device): là những thiết bị thật như màn hình, cửa sổ, máy in,…
  - Thiết bị "ảo" (logical device): là các thiết bị được giả lập trong bộ nhớ, còn gọI là "memory device", có tác dụng mô phỏng thiết bị vật lý
- Thư viện GDI cho phép thực hiện các lệnh vẽ trên cả thiết bị vật lý lẫn logic

Các khái niệm cơ bản - Ngữ cảnh của thiết bị

- Ngữ cảnh của thiết bị (Device Context): là một cấu trúc lưu trữ các thông số của thiết bị đồ họa, ví dụ:
  - Chế độ vẽ hiện tại (drawing mode),
  - Vị trí bút vẽ hiện tại (pen position),
  - Các đối tượng vẽ hiện tại (Pen, Brush, Font)
  - …
- Các thông số này chi phối những thao tác vẽ lên thiết bị tại thời điểm thực hiện
- Mỗi thiết bị được đặc trưng bởi 1 cấu trúc DC

Các khái niệm cơ bản - Ngữ cảnh của thiết bị…(tt)

- Ứng dụng thực hiện các thao tác vẽ lên thiết bị thông qua DC của thiết bị đó



- DC được xem như là một phương tiện liên kết giữa ứng dụng và thiết bị, giúp tạo nên tính độc lập thiết bị (Device Independent) cho ứng dụng

Các khái niệm cơ bản - Ngữ cảnh của thiết bị…(tt)

Thuộc tính	Gía trị mặc định	Ý nghĩa/Các hàm liên quan
Background color	White	Màu nền khi xuất ký tự (**`SetBkColor`**)
Text color	Black	Màu ký tự (**`SetTextColor`**)
Font	SYSTEM_FONT	Font chữ khi xuất ký tự (**`CreateFont`**, **`CreateFontIndirect`**, **`SelectObject`**)
Pen	BLACK_PEN	Bút vẽ, sử dụng khi vẽ đường thẳng, đường cong (**`CreatePen`**, **`CreatePenIndirect`**, **`SelectObject`**)
Current Pen Position	(0, 0)	Tọa độ hiện hành của Pen (**`MoveTo`**, **`LineTo`**)

Một số thuộc tính do DC quản lý

Các khái niệm cơ bản - Ngữ cảnh của thiết bị…(tt)

Thuộc tính	Gía trị mặc định	Ý nghĩa/Các hàm liên quan
Brush	WHITE_BRUSH	Màu tô, sử dụng khi tô màu các vùng kín (**`CreateSolidBrush`**, **`CreateBrushIndirect`**, **`CreatePatternBrush`**, **`SelectObject`**,…)
Drawing mode	R2_COPYPEN	Chế độ vẽ. Xác định cách phối hợp các bit màu của 2 pixel có cùng tọa độ (**`SetROP2`**)
Bitmap	NULL	(**`CreateBitmap`**, **`CreateBitmapIndirect`**, **`CreateCompatibleBitmap`**, **`SelectObject`**)

Một số thuộc tính do DC quản lý

Các khái niệm cơ bản - Ngữ cảnh của thiết bị…(tt)

Thuộc tính	Gía trị mặc định	Ý nghĩa/Các hàm liên quan
Color Palette	DEFAULT_PALETTE	Bảng màu (**`CreatePalette`**, **`RealizePalette`**, **`SelectPalette`**, **`UnrealizeObject`**)
Mapping mode	MM_TEXT	Xác định đơn vị đo. VD. - **`MM_TEXT`** qui định đơn vị đo theo trục x,y là 1 pixel. - **`MM_HIMETRIC`** qui định đơn vị đo theo trục x,y là 0.01 milimet (**`SetMapMode`**)
… … …		

Một số thuộc tính do DC quản lý

Các khái niệm cơ bản - Ngữ cảnh của thiết bị…(tt)

- Các loại DC:
  - Display DC: sử dụng trong các thao tác vẽ lên màn hình/cửa sổ
  - Printer DC: sử dụng để kết xuất dữ liệu đồ họa ra máy in
  - Memory DC: tương ứng với thiết bị "logic", thường dùng để "chuẩn bị" dữ liệu trước khi hiển thị ra thiết bị vật lý

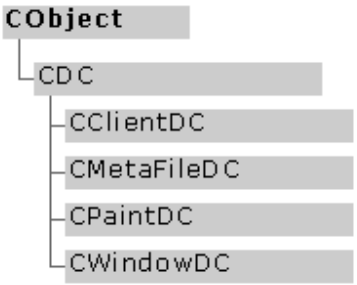
Các khái niệm cơ bản - Ngữ cảnh của thiết bị…(tt)

Loại DC	Các hàm liên quan
Display DC	**`BeginPaint, GetDC, GetDCEx`** **`EndPaint, ReleaseDC`**
Printer DC	**`CreateDC`** **`DeleteDC`**
Memory DC	**`CreateCompatibleDC`** **`DeleteDC`**

Các hàm tạo lập/hủy bỏ DC

Các khái niệm cơ bản - DC trong MFC

- Các lớp liên quan đến DC
  - CDC: là lớp cơ sở dùng để quản lý DC
  - CPaintDC:
    - Dẫn xuất từ lớp CDC,
    - Dùng với các thao tác vẽ trong vùng client của cửa sổ
    - Chỉ sử dụng trong xử lý thông điệp **`WM_PAINT`** (hàm **`CWnd::OnPaint`**)



Các khái niệm cơ bản - DC trong MFC…(tt)

- Các lớp liên quan đến DC (tt)
  - CClientDC:
    - Dẫn xuất từ lớp CDC,
    - Dùng với các thao tác vẽ trong vùng client của cửa sổ
    - Sử dụng bất kỳ lúc nào, ngoại trừ hàm **`CWnd::OnPaint`**
  - CWindowDC:
    - Dẫn xuất từ lớp CDC,
    - Dùng với các thao tác vẽ trên cửa sổ (kể cả vùng client và non-client)

Các khái niệm cơ bản - DC trong MFC…(tt)

```
VD1. Vẽ bằng CDC
CDC* pDC = GetDC();
// Các lệnh vẽ …
ReleaseDC(pDC);

VD2. Xử lý thông điệp WM_PAINT, dùng lớp CDC
PAINTSTRUCT ps;
CDC* pDC = BeginPaint(&ps);
// Các lệnh vẽ …
EndPaint(&ps);

VD3. Xử lý thông điệp WM_PAINT, dùng lớp CPaintDC
CPaintDC dc(this);
// Các lệnh vẽ …
```


Các khái niệm cơ bản - DC trong MFC…(tt)

```
VD4. Vẽ trên vùng client, dùng lớp CClientDC

void CMainWindow::OnLButtonDown(UINT nFlags,
                                CPoint point)
{
  CRect rect;
  GetClientRect(&rect);
  CClientDC dc(this);

  dc.MoveTo(rect.left, rect.top);
  dc.LineTo(rect.right, rect.bottom);
  dc.MoveTo(rect.right, rect.top);
  dc.LineTo(rect.left, rect.bottom);
}
```

Các khái niệm cơ bản - DC trong MFC…(tt)

Thuộc tính	Gía trị mặc định	Ý nghĩa/Các hàm liên quan
Background color	White	**CDC::SetBkColor** **CDC::GetBkColor**
Text color	Black	**CDC::SetTextColor** **CDC::GetTextColor**
Font	SYSTEM_FONT	**CDC::SelectObject**
Pen	BLACK_PEN	**CDC::SelectObject**
Current Pen Position	(0, 0)	**CDC::MoveTo** **CDC::GetCurrentPosition**
Drawing mode	R2_COPYPEN	**CDC::SetROP2** **CDC::GetROP2**

Một số thuộc tính do lớp CDC quản lý

Các khái niệm cơ bản - DC trong MFC…(tt)

Hàm	Ý nghĩa
MoveTo	Di chuyển Pen đến vị trí mới
LineTo	Vẽ 1 đoạn thẳng từ vị trí Pen hiện hành đến vị trí mới
Polyline / PolylineTo	Vẽ 1 dãy các cạnh
Arc / ArcTo	Vẽ 1 cung
PolyBezier / PolyBezierTo	Vẽ đường cong Bezier
PolyDraw	Vẽ đường cong Bezier và các cạnh nối giữa các điểm

Một số hàm vẽ do lớp CDC cung cấp

Các khái niệm cơ bản - Đối tượng vẽ

- Đối tượng vẽ (Drawing object): là những đối tượng sẽ chi phối các thao tác vẽ của bạn. VD:
  - Đối tượng Pen sẽ chi phối thao tác vẽ đường thẳng, đường cong;
  - Đối tượng Brush sẽ chi phối thao tác tô màu;
  - Đối tượng Font sẽ chi phối thao tác xuất ký tự
- Đối tượng vẽ chứa các thông tin về màu sắc (color), kiểu dáng (style)

Các khái niệm cơ bản - Đối tượng vẽ…(tt)

- Các đối tượng vẽ cần phải được tạo mới (Create) hay lấy ra từ kho (stock) để dùng
  - Nếu tạo mới, sau khi dùng phải giải phóng
  - Nếu lấy từ kho có sẵn, sau khi dùng không cần giải phóng

```
VD1. Tạo mới 1 Pen
HPEN hPen1 = CreatePen(PS_SOLID, 2, RGB(255, 0, 0);

VD2. Lấy 1 Pen từ "kho" có sẵn
HPEN hPen2 = (HPEN) GetStockObject(WHITE_PEN);

VD3. Xoá đối tượng hPen1 sau khi sử dụng
DeleteObject(hPen1);
```

Các khái niệm cơ bản - Đối tượng vẽ…(tt)

- Các đối tượng vẽ cần được "gán" vào DC trước khi thực hiện thao tác vẽ. VD:

```
// "Gán" đối tượng hPen vào DC
HPEN hOldPen = (HPEN) SelectObject(hdc, hPen);
// Vẽ 1 đoạn thẳng, sử dụng đối tượng hPen vừa tạo
MoveTo(hdc, 50, 50);
LineTo(hdc, 100, 100);
```

- Cần phải "lấy" đối tượng vẽ ra khỏi DC trước khi giải phóng đối tượng hay giải phóng DC

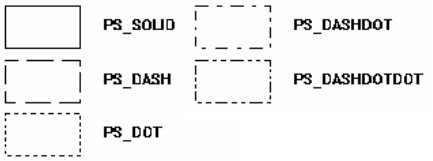
```
// "Lấy" đối tượng vẽ ra khỏi DC
SelectObject(hdc, hOldPen);
// Giải phóng DC và đối tượng vẽ (nếu đối tượng được
// tạo mới)
ReleaseDC(hFrameWnd, hdc);
DeleteObject(hPen);
```


Các khái niệm cơ bản - Đối tượng vẽ…(tt)

- Các đối tượng vẽ:
  - Pen (HPEN)
  - Brush (HBRUSH)
  - Font (HFONT)

Các khái niệm cơ bản - Đối tượng vẽ…(tt)

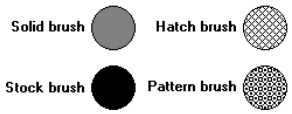
- Pen: có các thuộc tính
  - Style: kiểu dáng
  - Width: độ rộng (pixel)
  - Color: màu sắc (RGB)

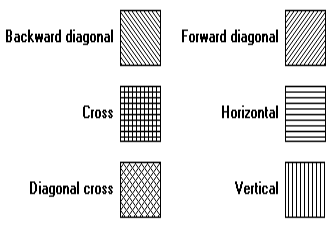



```
VD1. Tạo mới 1 Pen
HPEN hPen = CreatePen(PS_SOLID, 2, RGB(255, 0, 0);
VD2. Tạo mới 1 Pen, sử dụng cấu trúc LOGPEN
LOGPEN  lp;
lp.lopnStyle = PS_DOT;
lp.lopnWidth.x = 3;
lp.lopnWidth.y = 0;         // không sử dụng
lp.lopnColor = RGB(0, 255, 0);
HPEN hPen = CreatePenIndirect(&lp);
VD3. Lấy 1 Pen từ "kho" có sẵn
HPEN hPen;
hPen = (HPEN) GetStockObject(WHITE_PEN);
```

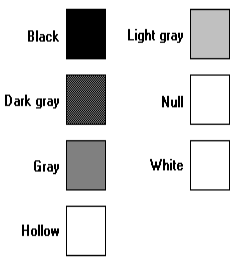

Các khái niệm cơ bản - Đối tượng vẽ…(tt)

- Brush: có các thuộc tính
  - Style: kiểu dáng
  - Color: màu sắc (RGB)
  - Hatch: mẫu tô

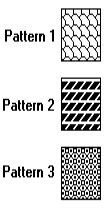




Hatch brush



Stock brush



Pattern brush

Các khái niệm cơ bản - Đối tượng vẽ…(tt)

- Brush (tt):

```
VD1. Tạo mới 1 Brush
LOGBRUSH lb;
HBRUSH hbr1;

lb.lbStyle = BS_HATCHED;
lb.lbColor = RGB(255, 0, 0);
lb.lbHatch = HS_CROSS;

hbr1 = CreateBrushIndirect(&lb);

VD2. Lấy 1 Brush từ "kho" có sẵn
HBRUSH hbr2;

hbr2 = (HBRUSH) GetStockObject(LTGRAY_BRUSH);
```


Các khái niệm cơ bản - Đối tượng vẽ…(tt)

- Font: có các thuộc tính
  - Typeface: loại font (Times, Courier, Arial,…)
  - Style: kiểu dáng (normal, thin, bold,…)
  - Size: kích cỡ chữ, được xác định theo đơn vị point, 1 point = 1/72 inch = 0.013837 inch
- Có thể tạo đối tượng font bằng cách:
  - Thay đổi mẫu font có sẵn
  - Chọn thuộc tính Font từ hộp thoại ChooseFont

Các khái niệm cơ bản - Đối tượng vẽ…(tt)

- Font (tt)

```
VD1. Tạo mới 1 font từ mẫu có sẵn
LOGFONT  lf;
// Lấy font mẫu từ "kho"
HFONT hFont = (HFONT) GetStockObject(SYSTEM_FONT);
// Lấy thuộc tính của font mẫu
GetObject(hFont, sizeof(LOGFONT), (LOGFONT *) &lf);

// Sửa đổi các thuộc tính cho phù hợp yêu cầu
lf.lfHeight = -28;
lf.lfWidth = 0;
strcpy(lf.lfFaceName, "VNI-Helve");

// Tạo lập font mới
hFont = CreateFontIndirect(&lf);
```

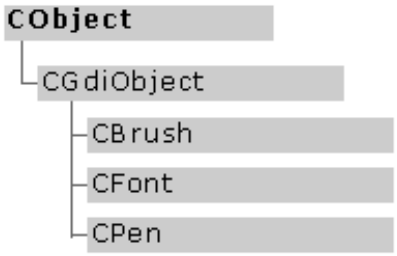

Các khái niệm cơ bản - Đối tượng vẽ…(tt)

- Font (tt)

```
VD2. Chọn thuộc tính font từ hộp thoại ChooseFont
LOGFONT lf;
CHOOSEFONT cf;
HFONT hFont;
// Khởi tạo cấu trúc CHOOSEFONT
ZeroMemory(&cf, sizeof(CHOOSEFONT));
cf.lStructSize = sizeof(CHOOSEFONT);
cf.hwndOwner = hFrameWnd;
cf.lpLogFont = &lf;     // Nhận thuộc tính font trả về
cf.Flags = CF_SCREENFONTS | CF_EFFECTS;
cf.rgbColors = RGB(0, 255, 255); // light blue
cf.nFontType = SCREEN_FONTTYPE;

if (ChooseFont(&cf)) {
  // Tạo lập font mới
  hFont = CreateFontIndirect(&lf);
  …
}
```

Các khái niệm cơ bản - Đối tượng vẽ trong MFC

Đối tượng	Lớp MFC
Pen	CPen
Brush	CBrush
Font	CFont




```
VD1. tạo đối tượng Pen (C1)
CPen pen(PS_SOLID, 1, RGB (255, 0, 0));

VD2. tạo đối tượng Pen (C2)
CPen pen;
pen.CreatePen(PS_SOLID, 1, RGB (255, 0, 0));
```


Các khái niệm cơ bản - Đối tượng vẽ trong MFC…(tt)

```
   VD3. tạo đối tượng Pen,dùng cấu trúc LOGPEN (C3)
CPen pen;
LOGPEN lp;
lp.lopnStyle = PS_SOLID;
lp.lopnWidth.x = 1;
lp.lopnColor = RGB(255, 0, 0);
pen.CreatePenIndirect(&lp);

   VD4. sử dụng đối tượng Pen để vẽ
CPen* pOldPen = dc.SelectObject(&pen);
dc.Ellipse(0, 0, 100, 100);

   VD5. hai cách để tạo solid brush
CBrush brush(RGB(255, 0, 0));

CBrush brush;
brush.CreateSolidBrush(RGB(255, 0, 0));
```

Các khái niệm cơ bản - Đối tượng vẽ trong MFC…(tt)

```
VD6. tạo Hatch brush và dùng để tô bên trong HCN
CBrush brush(HS_DIAGCROSS, RGB(255, 255, 255));
dc.SelectObject(&brush);
dc.SetBkColor(RGB(192, 192, 192)); // tô trên nền xám
dc.Rectangle(0, 0, 100, 100);

VD7. tạo đối tượng Font có cỡ chữ 12 point
CFont font;
font.CreatePointFont(120, _T("Times New Roman"));

VD8. tạo đối tượng Font bằng cấu trúc LOGFONT
LOGFONT lf;
::ZeroMemory(&lf, sizeof(lf));
lf.lfHeight = 120;
lf.lfWeight = FW_BOLD;
lf.lfItalic = TRUE;
::lstrcpy(lf.lfFaceName, _T("Times New Roman"));
CFont font;
font.CreatePointFontIndirect(&lf);
```


Các khái niệm cơ bản - Đối tượng vẽ trong MFC…(tt)

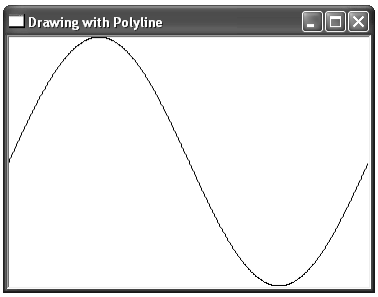
```
VD9. tạo font và dùng để xuất ký tự
CClientDC dc(this);
CFont font;
font.CreatePointFont(120, _T("Arial")));

CFont* def_font = dc.SelectObject(&font);

dc.TextOut(5, 5, _T("Hello, World !"), 14);

dc.SelectObject(def_font);
font.DeleteObject();
```

Các ví dụ vẽ hình đồ họa

```
VD1. Vẽ 1 đoạn thẳng
dc.MoveTo(0, 0);
dc.LineTo(0, 100);

VD2. Vẽ 1 tứ giác
POINT aPoint[5]={0, 0, 0, 100, 100, 100, 100, 0, 0, 0};
dc.Polyline(aPoint, 5);

VD3. Vẽ 1 cung
CRect rect (0, 0, 200, 100);
CPoint point1 (0, -500);
CPoint point2 (-500, 0);
dc.Arc (rect, point1, point2);
```

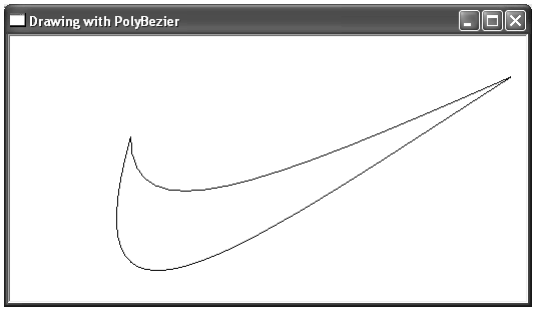

Các ví dụ vẽ hình đồ họa…(tt)

```
VD4. Vẽ 1 đường cong hình sin bằng Polyline
#include <math.h>
#define SEGMENTS 500
#define PI 3.1415926
void CMainWindow::OnPaint() {
  CRect rect;
  GetClientRect(&rect);
  int nWidth = rect.Width();
  int nHeight = rect.Height();
  CPaintDC dc(this);
  CPoint aPoint[SEGMENTS];
  for (int i=0; i<SEGMENTS; i++) {
      aPoint[i].x = (i*nWidth)/SEGMENTS;
      aPoint[i].y = (int)((nHeight/2)*(1-(sin((2*PI*i)
                          / SEGMENTS))));
  }
  dc.Polyline (aPoint, SEGMENTS);
}
```



Các ví dụ vẽ hình đồ họa…(tt)

```
VD5. Vẽ 1 đường cong với hàm PolyBezier
POINT aPoint1[4] = {120,100,120,200,250,150,500,40};
POINT aPoint2[4] = {120,100,50,350,250,200,500,40};
dc.PolyBezier(aPoint1, 4);
dc.PolyBezier(aPoint2, 4);
```



Xử lý ảnh Bitmap

- DDB và DIB
- Vẽ một DDB

Xử lý ảnh Bitmap - DDB và DIB

- DDB (Device Dependent Bitmap)
  - Có nhiều loại thiết bị khác nhau để lưu trữ/hiển thị ảnh bitmap → cần có nhiều định dạng (format) khác nhau để lưu ảnh trên thiết bị
  - Các định dạng này gọi là DDB
- DIB (Device Independent Bitmap)
  - Các định dạng DDB sẽ không tương thích trên các loại thiết bị khác nhau
  - DIB là một định dạng do Windows cung cấp nhằm đảm bảo ảnh được lưu trữ "độc lập với thiết bị" → tương thích với nhiều loại thiết bị
  - Khi ta load một DIB, device driver sẽ chuyển đổi nó thành dạng DDB ứng với thiết bị

Xử lý ảnh Bitmap - Vẽ một DDB

- Hàm **`BitBlt (CDC::BitBlt)`**: copy 1 ảnh bitmap từ DC nguồn (**`hdcSrc`**) sang DC đích (**`hdcDest`**)

```
BOOL BitBlt(HDC hdcDest, DC int nXDest,
          int nYDest, int nWidth, int nHeight,
          HDC hdcSrc, int nXSrc, int nYSrc,
          DWORD dwRop);
```

- Hàm **`StretchBlt (CDC::StretchBlt)`**: copy 1 ảnh bitmap từ DC nguồn (**`hdcSrc`**) sang DC đích (**`hdcDest`**), đồng thời co/giãn ảnh

```
BOOL StretchBlt(HDC hdcDest, int nXDest, int nYDest,
           int nWidthDest, int nHeightDest,
           HDC hdcSrc, int nXSrc, int nYSrc,
           int nWidthSrc, int nHeightSrc,
           DWORD dwRop);
```

Xử lý ảnh Bitmap - Vẽ một DDB…(tt)

- Lấy thông tin của ảnh bitmap:
  - API: Hàm **GetObject**

```
BITMAP bm;
GetObject(hBitmap,sizeof(BITMAP),&bm);
```

  - MFC: hàm **CBitmap::GetBitmap**

```
BITMAP bm;
bitmap.GetBitmap(&bm);
```


Xử lý ảnh Bitmap - Vẽ một DDB…(tt)

VD. Load bitmap từ resource,vẽ lên cửa sổ View

```
CxxxView::OnDraw(CDC* pDC) {
  CBitmap bitmap;
  CBitmap oldBitmap;
  BITMAP bm;
  CDC dcMemory;

  bitmap.LoadBitmap(IDB_SAMPLE);
  bitmap.GetBitmap(&bm);
  dcMemory.CreateCompatibleDC(pDC);
  oldBitmap = dcMemory.SelectObject(&bitmap);
  pDC->BitBlt(100, 100, bm.bmWidth, bm.bmHeight,
              &dcMemory, 0, 0, SRCCOPY);
  dcMemory.SelectObject(&oldBitmap);
}
```

In ấn (Printing)

- Tổng quan về in ấn trong Windows
- In ấn trong MFC

In ấn (Printing) - Tổng quan về in ấn trong Windows

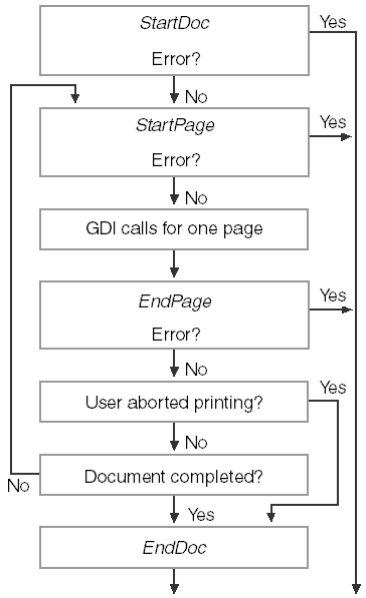
- Nguyên tắc cơ bản về in ấn
- Print job
- Banding
- Spooling
- Printer DC
- Một chương trình in đơn giản dùng API

In ấn (Printing) - Tổng quan về in ấn trong Windows…(tt)

- Nguyên tắc cơ bản về in ấn
  - Tạo DC của máy in bằng cách dùng lệnh **`CreateDC`** hoặc dùng hộp thoại **`PrintDlg`**
  - Gọi hàm **`StartDoc`** để bắt đầu in một tài liệu mới
  - Gọi hàm **`StartPage`** để bắt đầu 1 trang in
  - Gọi các hàm vẽ của GDI để thể hiện văn bản (text) hay hình ảnh (bitmap, graphics) ra trang in
  - Gọi hàm **`EndPage`** để kết thúc 1 trang in
  - Gọi hàm **`EndDoc`** để kết thúc in tài liệu

  Note:
  - Các hàm **`StartDoc/EndDoc`**, **`StartPage/EndPage`** đều thuộc thư viện GDI
  - Bước [3], [4], [5] có thể làm nhiều lần nếu muốn in trên nhiều trang

In ấn (Printing) - Tổng quan về in ấn trong Windows…(tt)



In ấn (Printing) - Tổng quan về in ấn trong Windows…(tt)

- Print job (tác vụ in)
  - Là một tài liệu cần in
  - Có thể chứa 1 hay nhiều trang in
  - Print job được định nghĩa bởi 1 cặp lệnh **`StartDoc/EndDoc`**
  - Cấu trúc dữ liệu được dùng cho print job là Enhanced Metafile (EMF)
  - EMF: cấu trúc dùng để lưu trữ các lệnh vẽ text, vẽ đồ họa,…

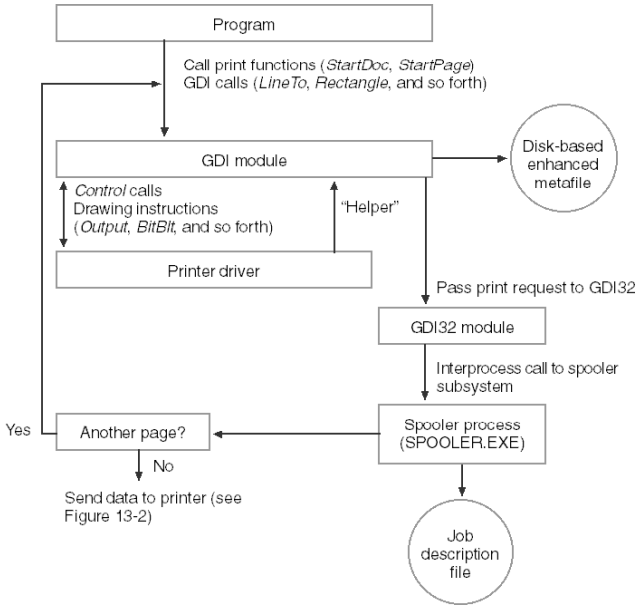
In ấn (Printing) - Tổng quan về in ấn trong Windows…(tt)

- Banding
  - GDI lưu trữ các lệnh vẽ lên máy in trong một file có dạng ~EMF*.TMP (Enhanced Metafile)
  - Khi ứng dụng kết thúc in ấn 1 trang (bằng lệnh **`EndPage`**), printer driver sẽ chuyển đổi các lệnh vẽ trong metafile thành các kết xuất thực sự trên máy in → cần thiết phải tạo ra 1 trang đồ họa
  - Kích thước bộ nhớ cho 1 trang đồ họa khá lớn (VD. Máy in 600DPI, giấy letter cần khoảng 4MB)
  - Do đó, printer driver thường sử dụng kỹ thuật “chia band”: chia 1 trang đồ họa thành nhiều vùng hình chữ nhật
  - GDI sẽ lấy kích thước band từ printer driver (tùy thuộc máy in) và thực hiện các lệnh vẽ trong metafile lên các band của 1 trang

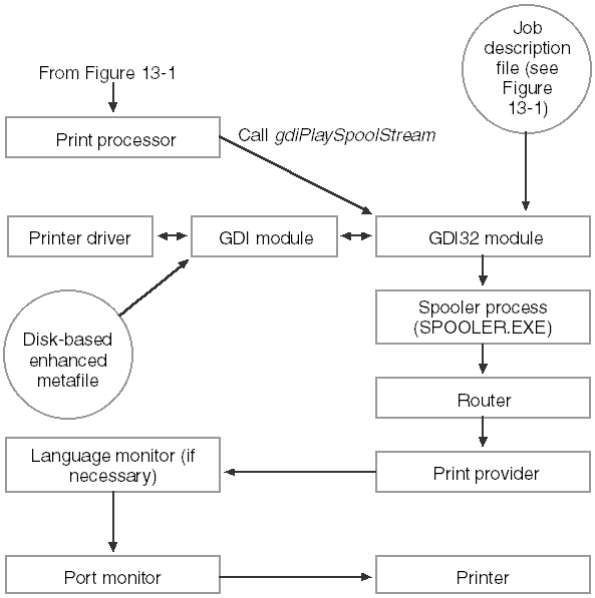
In ấn (Printing) - Tổng quan về in ấn trong Windows…(tt)

- Spooling
  - Windows quản lý quá trình in ấn bằng trình quản lý in (Printer Spooler)
  - Printer spooler được tự động nạp khi Windows khởi động và kết thức khi HĐH shutdown
  - Printer spooler giúp:
    - Xác định trình điều khiển máy in; nạp vào bộ nhớ
    - Lập lịch cho việc in ấn
    - Gởi dữ liệu ra máy in qua cổng (parallel, serial)
    - Xoá các file TMP sau khi in xong…
- User có thể tắt cơ chế Spooling đối với một máy in (trong Printer properties/Ports)
  - Để in nhanh hơn, vì GDI không lưu dữ liệu cần in lên file mà gởi trực tiếp ra máy in
  - Hoặc dùng một trình điều khiển spooling khác (VD. Máy in mạng)

In ấn (Printing) - Tổng quan về in ấn trong Windows…(tt)



In ấn (Printing) - Tổng quan về in ấn trong Windows…(tt)



In ấn (Printing) - Tổng quan về in ấn trong Windows…(tt)

- Printer DC
  - Việc đầu tiên khi in là phải tạo ra Printer DC
  - Sử dụng Printer DC trong các lệnh GDI giống như dùng Window DC
  - User có thể cài đặt nhiều máy in, nhưng chỉ có 1 máy in mặc định (default)
  - Mỗi máy in đều có trình điều khiển tương ứng
  - Cần phải xác định tên của máy in khi tạo Printer DC
  - Có 2 cách tạo Printer DC
    - Tạo Printer DC trực tiếp
    - Tạo Printer DC bằng cách dùng hộp thoại chuẩn **`PrintDlg`**

In ấn (Printing) - Tổng quan về in ấn trong Windows…(tt)

```
VD1. Tạo Printer DC trực tiếp
HDC GetPrinterDC() {
  DWORD dwNeeded, dwReturned;
  HDC hdc;
  PRINTER_INFO_4 * pinfo4;
  PRINTER_INFO_5 * pinfo5;

  if (GetVersion() & 0x80000000) // Windows 98
  {
      EnumPrinters(PRINTER_ENUM_DEFAULT, NULL, 5, NULL,
                   0, &dwNeeded, &dwReturned);
      pinfo5 = malloc(dwNeeded);
      EnumPrinters(PRINTER_ENUM_DEFAULT, NULL, 5,
         (PBYTE)pinfo5,dwNeeded,&dwNeeded,&dwReturned);
      hdc = CreateDC(NULL, pinfo5->pPrinterName,
                     NULL, NULL);
      free(pinfo5);
  }                                                  ...
```


In ấn (Printing) - Tổng quan về in ấn trong Windows…(tt)

```
VD1. Tạo Printer DC trực tiếp… (tt)

  else // Windows NT
  {
      EnumPrinters(PRINTER_ENUM_LOCAL, NULL, 4,
                   NULL, 0, &dwNeeded, &dwReturned);
      pinfo4 = malloc(dwNeeded);
      EnumPrinters(PRINTER_ENUM_LOCAL, NULL, 4,
         (PBYTE)pinfo4,dwNeeded,&dwNeeded,&dwReturned);
      hdc = CreateDC(NULL, pinfo4->pPrinterName,
                     NULL, NULL);
      free(pinfo4);
  }
  return hdc;
}
```

In ấn (Printing) - Tổng quan về in ấn trong Windows…(tt)

```
VD 2. Tạo Printer DC: dùng hộp thoại chuẩn PrintDlg
HDC GetPrinterDC() {
  PRINTDLG pd;
  HWND hwnd = GetFocus();
  // Initialize PRINTDLG
  ZeroMemory(&pd, sizeof(PRINTDLG));
  pd.lStructSize = sizeof(PRINTDLG);
  pd.hwndOwner = hwnd;
  pd.hDevMode = NULL;
  pd.hDevNames = NULL;
  pd.Flags = PD_USEDEVMODECOPIESANDCOLLATE |
             PD_RETURNDC;
  pd.nCopies = 1;
  pd.nFromPage = 0xFFFF; pd.nToPage = 0xFFFF;
  pd.nMinPage = 1; pd.nMaxPage = 0xFFFF;
  if (PrintDlg(&pd)==TRUE) return pd.hDC
}
```


In ấn (Printing) - Tổng quan về in ấn trong Windows…(tt)

```
VD. Một chương trình in đơn giản dùng API
int doPrint() {
  DOCINFO di={sizeof(DOCINFO), TEXT("SamplePrinting")};
  HDC dcPrint = GetPrinterDC();
  if (dcPrint != NULL) {
      if (StartDoc(dcPrint, &di) <= 0) return 0;
      if (StartPage(dcPrint) <= 0) {
            EndDoc(dcPrint);
            return 0;
      }
      // các lệnh thể hiện text, graphics lên trang in
      ……
      EndPage(dcPrint);
      EndDoc(dcPrint);
      DeleteDC(dcPrint);
  }
  return 1;  // thành công
}
```

In ấn (Printing) - In ấn trong MFC

- Sử dụng các kỹ thuật in của Windows trong MFC
- Kỹ thuật in của MFC

In ấn trong MFC - Sử dụng các kỹ thuật in của Windows

- VD1. Tạo Printer DC trực tiếp
- VD2. Tạo Printer DC từ máy in mặc định
- VD3. Tạo Printer DC dùng hộp thoại Print
- VD4. Một chương trình in

In ấn trong MFC - Sử dụng các kỹ thuật in của Windows…(tt)

```
VD1. Tạo Printer DC trực tiếp
CDC dc;
dc.CreateDC(NULL, _T("HP LaserJet IIP"),
              NULL, NULL);

VD2. Tạo Printer DC từ máy in mặc định
CDC dc;
CPrintDialog dlg(FALSE);
dlg.GetDefaults();
dc.Attach(dlg.GetPrinterDC());

VD3. Tạo Printer DC dùng hộp thoại Print
CDC dc;
CPrintDialog dlg(FALSE);
if (dlg.DoModal()==IDOK)
  dc.Attach(dlg.GetPrinterDC());
```


In ấn trong MFC - Sử dụng các kỹ thuật in của Windows…(tt)

```
VD4. Một chương trình in
DOCINFO di;
::ZeroMemory(&di, sizeof (DOCINFO));
di.cbSize = sizeof(DOCINFO);
di.lpszDocName = _T("Sample of print");
if (dc.StartDoc(&di) > 0) {
  BOOL bContinue = TRUE;
  for (int i=1; i<=nPageCount && bContinue; i++) {
      dc.StartPage();
      // Xác lập các tham số cho Printer DC
      // In trang thứ i
      // …
      if (dc.EndPage() <= 0) bContinue = FALSE;
  }
  if (bContinue) dc.EndDoc();
  else dc.AbortDoc();
}
```

In ấn trong MFC - Kỹ thuật in của MFC

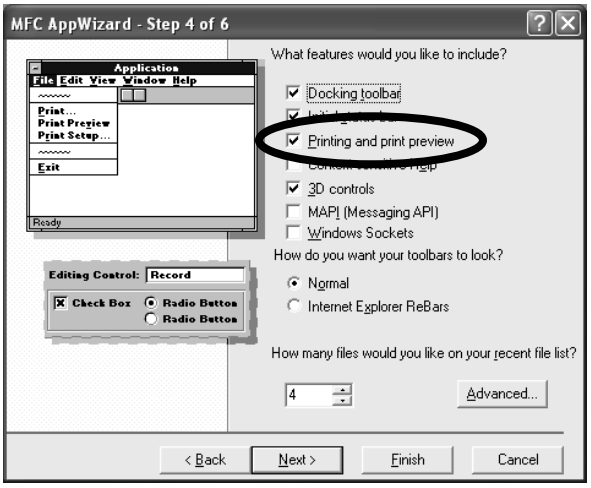
- Một ví dụ đơn giản với chức năng Print và Print Preview
- Tổng quan
- In trên nhiều trang

Kỹ thuật in của MFC

- Một ví dụ đơn giản với chức năng Print và Print Preview
  - Tạo 1 ứng dụng bằng MFC AppWizard
  - Chọn option Single Document
  - Chọn option Print and Print Preview
  - Thêm dòng lệnh sau vào hàm **CxxxView::OnDraw**

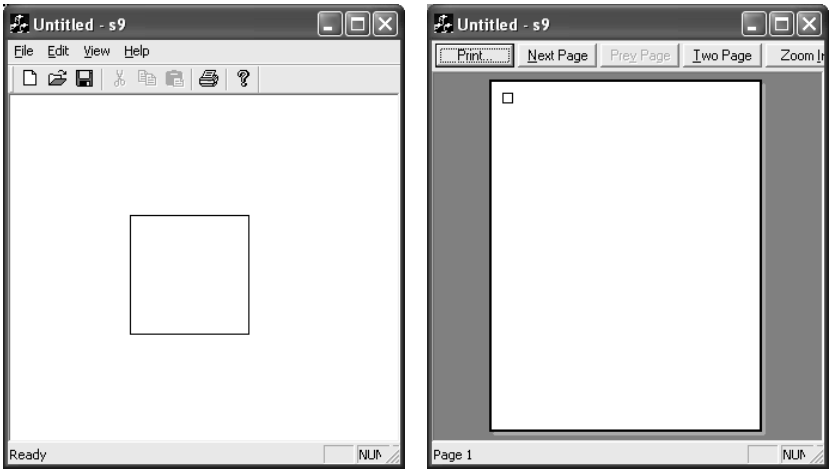
```
pDC->Rectangle(100, 100, 200, 200);
```

Kỹ thuật in của MFC



Chọn option hỗ trợ in trong AppWizard

Kỹ thuật in của MFC



Ứng dụng đơn giản có hỗ trợ Print và Print Preview của MFC

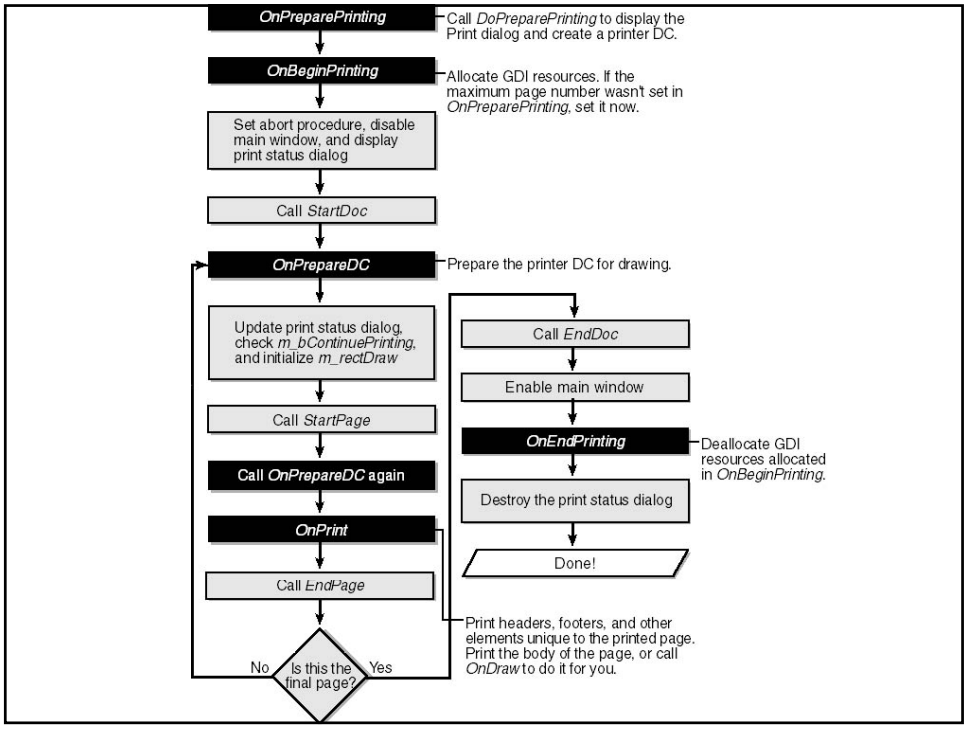
Kỹ thuật in của MFC - Tổng quan

- Giới thiệu
  - MFC framework tích hợp sẵn kỹ thuật in ấn, thực hiện phần lớn các công việc; giúp người lập trình tiết kiệm rất nhiều công sức và loại bỏ nhiều sự phức tạp như khi sử dụng Windows SDK
    - Tự động tạo và xóa Printer DC
    - Tự động gọi **`StartDoc/EndDoc`**, **`StartPage/EndPage`**
    - Hỗ trợ hộp thoại theo dõi trạng thái của print job; khả năng hủy bỏ in (Abort procedure)
    - Khả năng Print Preview,…

Kỹ thuật in của MFC - Tổng quan…(tt)

- Giới thiệu…(tt)
  - Các hàm quan trọng của **`CView`** liên quan đến in ấn

`OnPreparePrinting()`	Được gọi khi bắt đầu print job. Xác định số trang in;…
`OnBeginPrinting()`	Được gọi trước khi việc in bắt đầu. Định nghĩa các tài nguyên sử dụng cho việc in (Font, Brush,…)
`OnPrepareDC()`	Được gọi trước khi in mỗi trang. Thay đổi tính chất của Printer DC
`OnPrint()`	Được gọi để thực hiện in mỗi trang. In Header, Footer,… và gọi đến hàm OnDraw để in phần chính của trang
`OnEndPrinting()`	Được gọi khi kết thúc print job. Giải phóng các tài nguyên

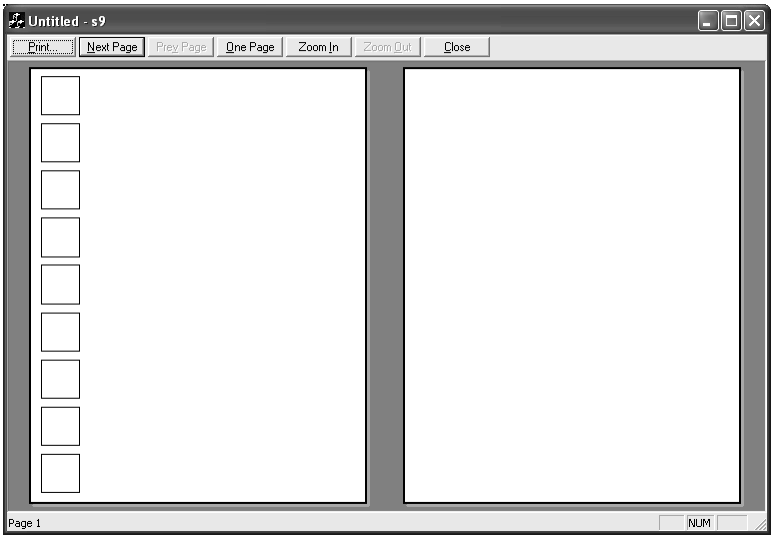


Kỹ thuật in của MFC - In trên nhiều trang

- Tạo 1 ứng dụng bằng MFC AppWizard với các options Single Document và Print and Print Preview
- Thêm đoạn lệnh sau vào hàm **`CxxxView::OnDraw`**

```
// Chuyển đổi 1 pixel = 1/100 inch
pDC->SetMapMode(MM_LOENGLISH);
// m_numRects: số hình chữ nhật muốn vẽ
// W = H = 100 pixel, khoảng cách 20 pixel
for (int i=0; i < m_numRects; i++)
  pDC->Rectangle(0, -i*120,
                 100, -(i+1)*120+20);
```

Kỹ thuật in của MFC - In trên nhiều trang…(tt)



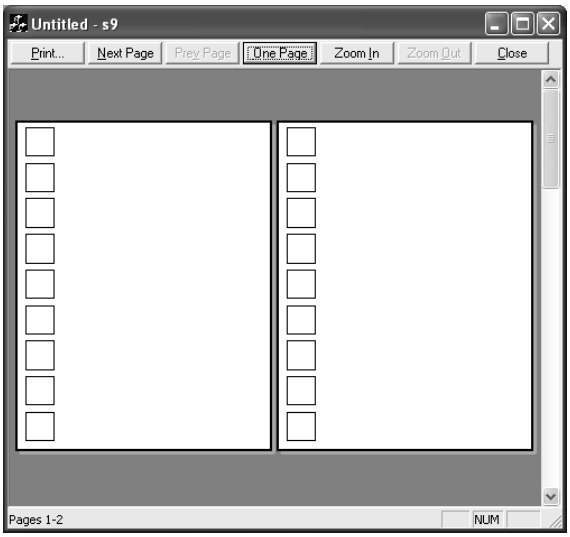
Các hình chữ nhật không hiển thị đúng trên nhiều trang in

Kỹ thuật in của MFC - In trên nhiều trang…(tt)

- Các bước thực hiện để in nhiều trang
  - B1. Thông báo cho MFC biết số trang cần in

```
void CxxxView::OnBeginPrinting(CDC* pDC,
                               CPrintInfo* pInfo)
{
  CxxxDoc* pDoc = GetDocument();
  ASSERT_VALID(pDoc);
  int pageHeight =
       pDC->GetDeviceCaps(VERTRES);
  int logPixelsY =
       pDC->GetDeviceCaps(LOGPIXELSY);
  int rectHeight = (int)(1.2 * logPixelsY);
  int numPages =
       pDoc->m_numRects*rectHeight/pageHeight + 1;
  pInfo->SetMaxPage(numPages);
}
```

Kỹ thuật in của MFC - In trên nhiều trang…(tt)



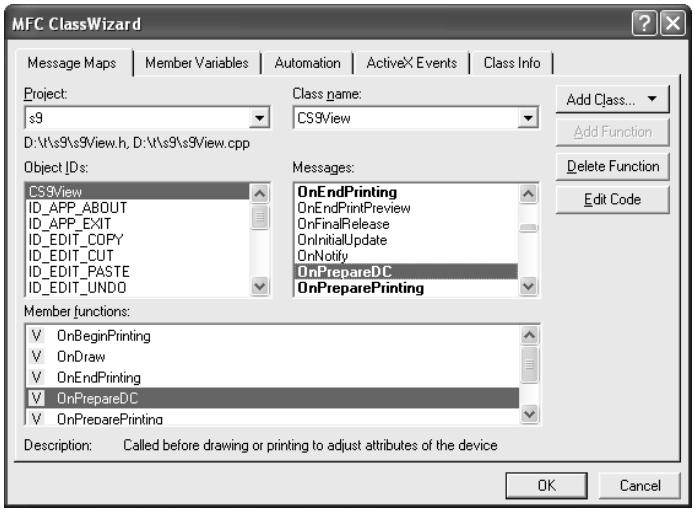
Các hình chữ nhật đã hiển thị trên nhiều trang in, nhưng nội dung các trang giống hệt nhau

Kỹ thuật in của MFC - In trên nhiều trang…(tt)

- Các bước thực hiện để in nhiều trang…(tt)
  - B2. Định nghĩa lại hàm CxxxView::OnPrepareDC

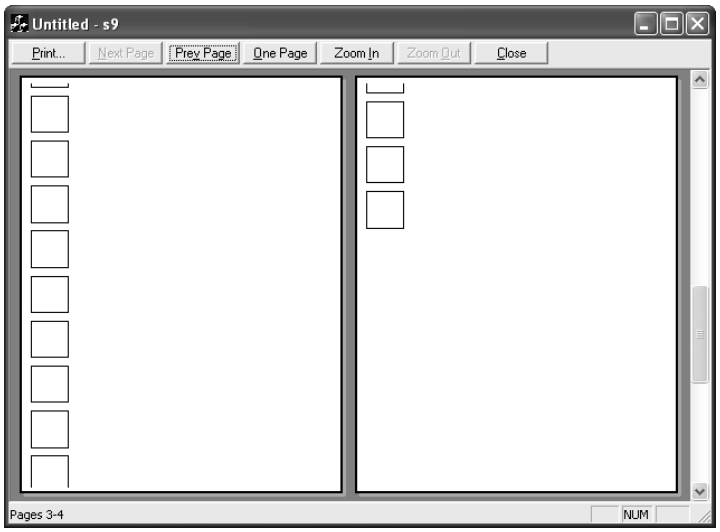
```
void CxxxView::OnPrepareDC(CDC* pDC,
                           CPrintInfo* pInfo)
{
   if (pDC->IsPrinting()) {
       int pageHeight =
             pDC->GetDeviceCaps(VERTRES);
       int originY = pageHeight *
                     (pInfo->m_nCurPage - 1);
       pDC->SetViewportOrg(0, -originY);
   }

   CView::OnPrepareDC(pDC, pInfo);
}
```

Kỹ thuật in của MFC - In trên nhiều trang…(tt)



Dùng ClassWizard định nghĩa lại hàm OnPrepareDC

Kỹ thuật in của MFC - In trên nhiều trang…(tt)



Các hình chữ nhật đã hiển thị đúng trên nhiều trang

Cám ơn - Hỏi & Đáp

Lập trình C trên Windows

Các kỹ thuật xử lý Clipboard

Nguyễn Tri Tuấn
Khoa CNTT – ĐH.KHTN.Tp.HCM
Email: nttuan@ fit.hcmuns.edu.vn

Nội dung

- Giới thiệu Clipboard
- Các kiểu định dạng sử dụng trong Clipboard
- Các kỹ thuật cơ bản sử dụng Clipboard

[1] Giới thiệu Clipboard

- Clipboard là gì ?
- Nhu cầu sử dụng Clipboard
- Các cơ chế Clipboard trong Windows
- Giới thiệu về tiện ích Clipboard Viewer

[1] Giới thiệu Clipboard - Clipboard là gì ?

- *Clipboard* là một *vùng nhớ chung* của Windows mà tất cả các ứng dụng đều có thể truy cập đến
- *Clipboard* là một phương thức chuyển dữ liệu chuẩn được Windows cung cấp, cho phép chia xẻ thông tin giữa các ứng dụng.

[1] Giới thiệu Clipboard - Nhu cầu sử dụng Clipboard

- Clipboard được sử dụng để cài đặt cho các thao tác thông dụng: Cut, Copy, Paste, Drag and Drop. Một ứng dụng có nhu cầu đặt dữ liệu vào Clipboard để sau đó một ứng dụng khác (hoặc chính nó) có thể truy xuất và sử dụng
- Một ứng dụng chỉ nên chuyển dữ liệu vào và ra Clipboard khi có yêu cầu từ người sử dụng. Không được sử dụng Clipboard để chuyển dữ liệu mà người sử dụng không biết

[1] … - Các cơ chế Clipboard trong Windows

- Cơ chế Windows Clipboard API chuẩn
- Cơ chế OLE Clipboard

[1] … - Giới thiệu về tiện ích Clipboard Viewer

- *Clipboard Viewer* là một cửa sổ hiển thị nội dung hiện thời của Clipboard
- *Clipboard Viewer* là một tiện ích hỗ trợ cho người sử dụng và không tác động đến chức năng chuyển giao dữ liệu của Clipboard

[1] … - Giới thiệu về tiện ích Clipboard Viewer

- Có nhiều Clipboard Viewer có thể chạy trên Windows ở cùng một thời điểm. Tuy nhiên, Windows chỉ giữ handle của một *Clipboard Viewer hiện hành*
- Chỉ có Clipboard Viewer hiện hành được Windows gửi thông điệp mỗi khi có sự thay đổi nội dung Clipboard…
- …Clipboard Viewer hiện hành có nhiệm vụ gửi các thông điệp này đến cho các Clipboard Viewer khác trong *chuỗi xích Clipboard Viewer*

[2] Các kiểu định dạng sử dụng trong Clipboard

- Giới thiệu
- Định dạng chuẩn
- Định dạng riêng

[2] Các kiểu định dạng … - Giới thiệu

- Dữ liệu chuyển vào Clipboard cần phải có một định dạng nhất định để các chương trình sử dụng nó có thể truy xuất chính xác
- Định dạng dữ liệu được xác định bởi tham số **`uFormat`** trong hàm :
  - **`SetClipboardData(UINT uFormat, HANDLE hMem)`**
  - **`hMem`** là handle của khối bộ nhớ chứa dữ liệu có định dạng tương ứng với **`uFormat`**

[2] Các kiểu định dạng … - Giới thiệu

- Clipboard API chuẩn cho phép sử dụng các kiểu định dạng sau đây:
  - Các định dạng chuẩn
  - Các định dạng riêng

[2] Các kiểu định dạng … - Giới thiệu

- *Lưu ý :*
  - Một ứng dụng có thể cùng lúc đặt nhiều đối tượng dữ liệu vào Clipboard (bằng cách gọi liên tiếp hàm ***`SetClipboardData`***), các đối tượng này thể hiện cùng một nội dung dữ liệu nhưng ở các định dạng khác nhau (và do đó đôi khi có hàm lượng thông tin khác nhau)

[2] Các kiểu định dạng … - Định dạng chuẩn

- Các định dạng Clipboard chuẩn (*Standard Clipboard Formats*) là các định dạng sử dụng cho Clipboard được Windows hỗ trợ
- Định danh của định dạng chuẩn được định nghĩa trong **`Winuser.h`**

[2] Các kiểu định dạng … - Định dạng chuẩn

- Các định dạng sử dụng với dữ liệu text:
  - **`CF_TEXT`**: dữ liệu là chuỗi ký tự ANSI , mỗi dòng kết thúc với 2 ký tự carriage return và linefeed (CR,LF). Ký tự NULL báo hiệu kết thúc dữ liệu
  - **`CF_UNICODETEXT`**: dữ liệu là chuỗi ký tự Unicode, mỗi dòng chấm dứt bằng CR,LF. Ký tự NULL (2 byte 0) báo hiệu kết thúc dữ liệu. Chỉ được hỗ trợ trong môi trường Windows NT/2000/XP
  - **`CF_OEMTEXT`**: tương tự như **`CF_TEXT`** nhưng sử dụng cho tập ký tự OEM

[2] Các kiểu định dạng … - Định dạng chuẩn

- Định dạng sử dụng với các bitmap:
  - **CF_BITMAP** : handle của một bitmap (**HBITMAP**)
  - **CF_DIB**: khối bộ nhớ định nghĩa một Device Independent Bitmap (DIB), bắt đầu bằng cấu trúc **BITMAPINFO**, theo sau là các bit của bitmap
  - **CF_DIBV5**: khối bộ nhớ chứa cấu trúc **BITMAPV5HEADER**, theo sau là thông tin về bảng màu và các bit của bitmap

[2] Các kiểu định dạng … - Định dạng chuẩn

- Định dạng sử dụng cho dữ liệu Metafile:
  - **CF_METAFILEPICT**: một Metafile Picture được định nghĩa bởi cấu trúc **METAFILEPICT**, dựa trên hỗ trợ metafile cũ của Windows
  - **CF_ENHMETAFILE**: handle của một metafile mở rộng (**HENHMETAFILE**)

[2] Các kiểu định dạng … - Định dạng chuẩn

- Một số định dạng khác:
  - **CF_HDROP**: Một danh sách các tập tin sử dụng với các dịch vụ cắt/dán file, kéo/thả file
  - **CF_PALETE**: handle của một bảng màu, thường được sử dụng kết hợp khi dữ liệu được đặt vào Clipboard phụ thuộc vào một bảng màu
  - Ngoài ra, còn một số định dạng: **CF_WAVE, CF_SYLK, CF_DIF, CF_TIFF, CF_PENDATA, CF_RIFF, CF_LOCALE** và các định dạng kết hợp với định dạng dữ liệu riêng

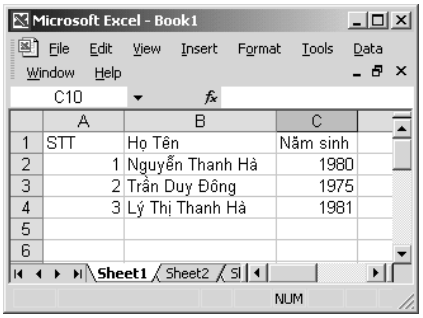
[2] Các kiểu định dạng … - Định dạng chuẩn

- Hệ thống tự thực hiện việc chuyển đổi định dạng dữ liệu giữa các định dạng sau:
  - **CF_TEXT, CF_OEMTEXT, CF_UNICODETEXT**
  - **CF_BITMAP, CF_DIB, CF_DIBV5**
  - **Từ CF_DIB và CF_DIBV5 sang CF_PALETE**
  - **CF_METAFILEPICT, CF_ENHMETAFILE**

[2] Các kiểu định dạng … - Định dạng riêng

- *Nhu cầu:*
  - Nhiều ứng dụng làm việc với định dạng riêng. Nếu chuyển dữ liệu vào hoặc ra Clipboard bằng các định dạng chuẩn có thể sẽ không bảo toàn được thông tin

Muốn chuyển dữ liệu bảng tính qua lại giữa các ứng dụng Excel thì phải sử dụng định dạng riêng



[2] Các kiểu định dạng … - Định dạng riêng

- *Phương pháp:* Windows cho phép một chương trình có thể sử dụng định dạng dữ liệu riêng cho mình theo một trong 2 cách sau:
  - Cách 1: Đăng ký định dạng mới (Registered Clipboard Formats) với hàm…
  - … **`UINT RegisterClipboardFormat (LPCTSTR lpszFormat)`**
  - **`lpszFormat`**: tên của định dạng mới

[2] Các kiểu định dạng … - Định dạng riêng

- Cách 1: Đăng ký định dạng mới…
  - Hàm trả về giá trị ***usigned int*** là định danh của định dạng mới. Định danh này sẽ được sử dụng như tham số trong các hàm chuyển/nhận dữ liệu vào/từ Clipboard: ***SetClipboardData*** và ***GetClipboardData***
  - Nếu 2 hay nhiều ứng dụng đăng kí định dạng với *cùng một tên* thì định dạng chỉ được đăng kí một lần, và giá trị trả về trong các lời gọi hàm ***RegisterClipboardFormat*** là như nhau. Điều này cho phép các ứng dụng chia xẻ dữ liệu với cùng một định dạng riêng

[2] Các kiểu định dạng … - Định dạng riêng

- Cách 2: Sử dụng định dạng riêng do Windows cung cấp
  - Không cần đăng ký
  - ...mà sử dụng một giá trị từ **CF_PRIVATEFIRST (0x200)** đến **CF_PRIVATELAST (0x2FF)** như là định danh của định dạng riêng

[2] Các kiểu định dạng … - Định dạng riêng

- *Vấn đề:*

Dữ liệu Text theo định dạng riêng trong MS Word …

- Clipboard Viewer
- MS Paint
- MS Notepad
- ……

… Các ứng dụng thông thường không hiểu được

[2] Các kiểu định dạng … - Định dạng riêng

- *Giải pháp:*
  - Một chương trình sử dụng định dạng riêng có thể chuyển cùng nội dung dữ liệu vào Clipboard *nhưng ở một số định dạng chuẩn* như:
    - **`CF_DSPTEXT, CF_DSPBITMAP, CF_DSPMETAFILEPICT, CF_DSPENHMETAFILE`**: các định dạng này cho phép Clipboard Viewer hiển thị dữ liệu dưới dạng Text, Bitmap, Metafile Picture hoặc Enhanced Metafile
    - **`CF_OWNERDISPLAY`**: *Chủ Clipboard* (ứng dụng cuối cùng chuyển dữ liệu vào Clipboard) có trách nhiệm hiển thị và cập nhật cho cửa sổ Clipboard Viewer bằng cách đáp ứng các thông điệp do cửa sổ này gửi đến

[2] Các kiểu định dạng … - Định dạng riêng

- *Giải pháp…: (tt)*
  - Các định dạng chuẩn khác như **`CF_TEXT`**, **`CF_BITMAP`**, ... để các ứng dụng thông dụng như Notepad, Paint … có thể hiển thị được nội dung dữ liệu trong Clipboard

[3] Các kỹ thuật cơ bản sử dụng Clipboard

- Vấn đề định vị bộ nhớ trong Windows
- Chuyển dữ liệu vào Clipboard
- Nhận dữ liệu từ Clipboard
- Truy vấn trên nhiều định dạng
- Kỹ thuật viết một Clipboard Viewer đơn giản

[3] Các kỹ thuật … - Định vị bộ nhớ trong Windows

- Sơ lược vấn đề định vị bộ nhớ trong Windows
- Một số hàm quản lý vùng nhớ toàn cục
- Ví dụ

[3] … - Định vị bộ nhớ trong Windows – Sơ lược

- Windows 32 bits quản lý bộ nhớ ảo (virtual memory) và sử dụng kỹ thuật phân trang
- *Vùng nhớ toàn cục* là vùng nhớ dùng chung cho tất cả các tiến trình. Khái niệm này chỉ có trên hệ điều hành Windows 16 bits nhưng vẫn được hỗ trợ trên Win32

[3] … - Định vị bộ nhớ trong Windows – Sơ lược

- Các hàm quản lý vùng nhớ toàn cục chậm và cung cấp ít tính năng hơn các hàm quản lý bộ nhớ khác nên ít được dùng ...
- …Tuy nhiên, chúng vẫn được dùng với DDE, **Clipboard** và các đối tượng dữ liệu OLE

[3] … - Định vị bộ nhớ trong Windows – Các hàm…

- **`HGLOBAL GlobalAlloc (UINT uFlags, T_SIZE dwBytes)`**
  - Hàm dùng để cấp phát một khối nhớ toàn cục mới
  - Nếu thành công, hàm trả về handle của khối nhớ toàn cục, nếu không, trả về NULL
  - **`dwBytes`**: số byte được cấp phát
  - **`uFlags`**: xác định cách cấp phát vùng nhớ
    - **`GMEM_FIXED`**: cấp phát vùng nhớ cố định
    - **`GMEM_MOVEABLE`**: cấp phát một vùng nhớ có thể di chuyển (địa chỉ trong không gian địa chỉ ảo có thể thay đổi)
    - **`GMEM_ZEROINIT`**: cấp phát vùng nhớ với các byte được khởi tạo bằng 0
    - **`GHND`**: kết hợp **`GMEM_MOVEABLE`** và **`GMEM_ZEROINIT`**
    - **`GPTR`**: kết hợp **`GMEM_FIXED`** và **`GMEM_ZEROINIT`**

[3] … - Định vị bộ nhớ trong Windows – Các hàm…

- *GlobalAlloc*…:
  - Khi sử dụng hàm **`GlobalAlloc`** để cấp phát vùng nhớ cho dữ liệu trong Clipboard, nên sử dụng cờ **`GMEM_MOVEABLE`**
  - …Lý do: cho phép hệ thống di chuyển khối nhớ trong không gian địa chỉ ảo, hạn chế tình trạng phân mảnh không gian địa chỉ ảo khi phải xóa và cấp phát lại nhiều lần

[3] … - Định vị bộ nhớ trong Windows – Các hàm…

- **`HGLOBAL GlobalRealloc(HGLOBAL hMem, T_SIZE dwBytes, UINT uFlags)`**
  - Hàm dùng để cấp phát lại một khối nhớ toàn cục
  - Hàm trả về handle của khối nhớ vừa được cấp phát lại
  - **`hMem`**: handle của khối nhớ global cần cấp phát lại
  - **`dwBytes`**: số byte của khối nhớ mới
  - **`uFlags`**: cờ xác định cách cấp phát lại

[3] … - Định vị bộ nhớ trong Windows – Các hàm…

- **SIZE_T *GlobalSize* (HGLOBAL hMem)**
  - Hàm được sử dụng để lấy kích thước của một khối nhớ global
  - Trả về số bytes của khối nhớ
  - **hMem**: handle của khối nhớ cần lấy kích thước

[3] … - Định vị bộ nhớ trong Windows – Các hàm…

- **HGLOBAL *GlobalFree*(HGLOBAL hMem)**
  - Hàm dùng để giải phóng một khối nhớ toàn cục
  - Nếu thành công, trả về NULL, nếu không, trả về giá trị bằng với handle của khối nhớ
  - **hMem**: handle của khối nhớ cần được giải phóng

[3] … - Định vị bộ nhớ trong Windows – Các hàm…

- **LPVOID *GlobalLock* (HGLOBAL hMem)**
  - Hàm dùng để khóa một khối nhớ toàn cục (cố định khối nhớ), sau mỗi lần gọi hàm, số lần khóa tăng lên 1. Khối nhớ có thuộc tính **GMEM_FIXED** luôn có số lần khóa bằng 0
  - Trả về con trỏ, trỏ đến phần tử đầu tiên của khối
  - **hMem**: handle của khối nhớ toàn cục

[3] … - Định vị bộ nhớ trong Windows – Các hàm…

- **BOOL *GlobalUnlock*(HGLOBAL hMem)**
  - Số lần khóa giảm đi 1 sau mỗi lần gọi hàm, khóa được bỏ (khối nhớ được phép di chuyển) khi số lần khóa bằng 0
  - Không tác động đến khối nhớ được cấp phát với cờ **GMEM_FIXED**
  - Trả về TRUE nếu khối vẫn còn bị khóa (số lần khóa lớn hơn 0)
  - Nếu trả về FALSE và hàm **GetLastError** trả về NO_ERROR, khóa khối được mở

[3] … - Định vị bộ nhớ trong Windows – Ví dụ

```
// Cấp một khối nhớ toàn cục, kích thước 256 bytes
HGLOBAL hGlobal;
hGlobal = GlobalAlloc(GMEM_MOVEABLE, 256);

// Khai báo một con trỏ kiểu int
int  *p;

// Truy xuất khối nhớ, gán giá trị cho các
// phần tử kiểu int
p = (int*) GlobalLock(hGlobal);
for (int i=0; i<GlobalSize(hGlobal)/sizeof(int); i++)
    p[i] = 1;

// Kết thúc truy xuất
GlobalUnlock(hGlobal);
```


[3] Các kỹ thuật … - Chuyển dữ liệu vào Clipboard

- Các bước cơ bản
- Dữ liệu với định dạng chuẩn
- Dữ liệu với định dạng riêng
- Kỹ thuật Delayed Rendering

[3] … - Chuyển dữ liệu … – Các bước cơ bản

- Các bước cần thực hiện để chuyển dữ liệu vào Clipboard:
  - *Bước 1:* Mở Clipboard với hàm ***OpenClipboard***
  - *Bước 2:* Loại bỏ dữ liệu hiện có trong Clipboard với hàm ***EmptyClipboard***
  - *Bước 3:* Sử dụng hàm ***SetClipboardData*** để chuyển *khối nhớ toàn cục* hoặc *các đối tượng khác* (như bitmap handle) chứa dữ liệu vào Clipboard
  - *Bước 4:* Đóng Clipboard với hàm ***CloseClipboard***

[3] … - Chuyển dữ liệu … – Các bước cơ bản

- Các hàm liên quan:
  - **`BOOL OpenClipboard(HWND hWnd)`**
    - ***`hWnd`***: handle cửa sổ
    - **`(MFC) BOOL CWnd::OpenClipboard()`**
  - **`BOOL EmptyClipboard(void)`**
  - **`HANDLE SetClipboardData(UINT uFormat, HANDLE hMem)`**
    - ***`uFormat`***: định dạng dữ liệu trong Clipboard
    - ***`hMem`***: handle của dữ liệu thuộc định dạng xác định
    - *Trả về*: handle khối nhớ toàn cục của dữ liệu
  - **`BOOL CloseClipboard(void)`**

[3] … - Chuyển dữ liệu … – Các bước cơ bản

- *Lưu ý:*
  - *Khối nhớ toàn cục* là một khối bộ nhớ được trả về từ hàm **`GlobalAlloc`**. Có thể sử dụng thêm các hàm liên quan như **`GlobalLock`** để nhận con trỏ trỏ tới khối nhớ và thiết lập giá trị cho dữ liệu trước khi chuyển vào Clipboard
  - Trong Win32, **`GlobalAlloc`** ít được sử dụng (và được thay bằng **`HeapAlloc`**). Tuy nhiên, **`GlobalAlloc`** còn được dùng trong lập trình Clipboard vì Clipboard yêu cầu *handle vùng nhớ* chứ không phải *con trỏ*

[3] … - Chuyển dữ liệu … – Các bước cơ bản

- *Lưu ý…:(tt)*
  - Tại một thời điểm chỉ có một chương trình có thể mở Clipboard
  - Lời gọi **`OpenClipboard`** giúp ngăn cản nội dung Clipboard bị thay đổi trong khi một chương trình đang sử dụng Clipboard
  - Clipboard nên được làm rỗng trước khi được nhận dữ liệu mới

[3] … - Chuyển dữ liệu … – DL với định dạng chuẩn

VD.1 Chuyển dữ liệu text vào Clipboard

```
char szText[ ] = "Hello, World" ;
int nLen = strlen(szText);
// Mở Clipboard
if (OpenClipboard())
{
  // Xoá dữ liệu trong Clipboard
  EmptyClipboard();
  // Định vị khối bộ nhớ với k.thước đủ lưu chuỗi
  HANDLE hData = GlobalAlloc(GHND, nLen + 1);
  // Khoá khối bộ nhớ để nhận con trỏ tương ứng
  char *pszData = (char *) GlobalLock(hData);
```


[3] … - Chuyển dữ liệu … – DL với định dạng chuẩn

VD.1 …

```
    // Chép nội dung dữ liệu vào vùng nhớ
    strcpy(pszData, szText);
    // Bỏ khoá khối
    GlobalUnlock(hData) ;
    // Chuyển dữ liệu vào Clipboard với
    // định dạng chuẩn CF_TEXT
    SetClipboardData(CF_TEXT, hData);
    // Đóng Clipboard
    CloseClipboard();
  }
```

[3] … - Chuyển dữ liệu … – DL với định dạng chuẩn

- *Lưu ý:*
  - Không được đưa handle vùng nhớ bị khoá cho Clipboard
  - Vùng nhớ đã được chuyển cho Clipboard không còn thuộc phạm vi sử dụng của ứng dụng.
  - *Giải pháp:* tạo sẵn bản sao của vùng nhớ hoặc sử dụng handle do **`SetClipboardData`** trả về

[3] … - Chuyển dữ liệu … – DL với định dạng chuẩn

VD.2

- Chuyển dữ liệu bitmap vào Clipboard
- Không như Ví dụ 1 chuyển một khối nhớ cho Clipboard, trong ví dụ này chúng ta sẽ chuyển vào Clipboard một *handle của bitmap*
- *Input* : hBitmap là handle của bitmap cần chuyển vào Clipboard

[3] … - Chuyển dữ liệu … – DL với định dạng chuẩn

```
VD.2 …
 // Mở Clipboard
 if (OpenClipboard())
 {
    // Làm rỗng Clipboard
    EmptyClipboard();
    // Chuyển dữ liệu vào Clipboard với định
    // dạng chuẩn CF_BITMAP
    SetClipboardData(CF_BITMAP, hBitmap);
    // Đóng Clipboard
    CloseClipboard();
 }
```


[3] … - Chuyển dữ liệu … – DL với định dạng chuẩn

VD.3: Chuyển dữ liệu với định dạng **CF_HDROP**

- Là phương pháp mà Windows 98 và Windows 2000 sử dụng để thực hiện các thao tác Cut, Copy và Paste trên các tập tin hay thư mục
- **HDROP** là handle của vùng nhớ toàn cục. Vùng nhớ này chứa một *cấu trúc* **DROPFILES** và theo sau là một *danh sách các tên file* kết thúc bằng 2 ký tự NULL
- Cấu trúc **DROPFILES**:

```
typedef struct _DROPFILES
{
      DWORD pFiles; // Offset of file list
      POINT pt;
      BOOL fNC;
      BOOL fWide;  // ANSI or Unicode text
} DROPFILES, FAR * LPDROPFILES;
```

[3] … - Chuyển dữ liệu … – DL với định dạng chuẩn

VD.3:…

```
// Danh sách file names
TCHAR szFiles [3][32] = {
          _T ( "C:\\TaiLieu.doc" ) ,
          _T ( "C:\\TaiLieu.zip" ) ,
          _T ( "" ) } ;
  // Mở Clipboard
  if (OpenClipboard())
  {
    // Làm rỗng Clipboard
    EmptyClipboard();
    // Định vị vùng nhớ đủ kích thước
    int nSize=sizeof(DROPFILES)+sizeof(szFiles);
    HANDLE hData = GlobalAlloc(GHND , nSize);
```


[3] … - Chuyển dữ liệu … – DL với định dạng chuẩn

VD.3:…

```
// Khởi tạo dữ liệu cho vùng nhớ
LPDROPFILES pDropFiles = (LPDROPFILES)
                         GlobalLock (hData);
pDropFiles → pFiles = sizeof(DROPFILES);
pDropFiles → fWide = FALSE ;  // Ansi text
LPBYTE pData = (LPBYTE) pDropFiles +
                         sizeof(DROPFILES);
CopyMemory(pData, szFiles, sizeof(szFiles));
GlobalUnlock(hData);
// Chuyển dliệu định dạng CF_HDROP vào Clipboard
SetClipboardData(CF_HDROP, hData);
// Đóng Clipboard
CloseClipboard();
}
```

[3] … - Chuyển dữ liệu … – DL với định dạng riêng

- Phương pháp
- Thêm vào các định dạng khác

[3] … - Chuyển dữ liệu … – DL với định dạng riêng

- Phương pháp:
  - Để sử dụng định dạng riêng , trước hết cần xác định định danh cho định dạng riêng bằng cách
    - Đăng ký định dạng riêng mới, hoặc …
    - … Sử dụng một số nguyên từ **`CF_PRIVATEFIRST`** đến **`CF_PRIVATELAST`** làm định danh cho định dạng
  - …sau đó chuyển dữ liệu với định dạng này vào Clipboard theo cách tương tự như định dạng chuẩn

[3] … - Chuyển dữ liệu … – DL với định dạng riêng

Ví dụ:

- Giả sử rằng chúng ta có một cấu trúc mô tả một hình tròn như sau:

```
struct  CIRCLE
{
     POINT center;     // Tâm
     int  radius;      // Bán kính
     COLORREF  color; // Màu tô
};
```

- Chúng ta cần chuyển một hình tròn được mô tả bởi cấu trúc **CIRCLE** vào Clipboard

[3] … - Chuyển dữ liệu … – DL với định dạng riêng

Ví dụ: (tt)

- Tạo một con trỏ **CIRCLE *pData** trỏ tới một đối tượng hình tròn cụ thể

- Đăng ký định dạng riêng cho cấu trúc hình tròn này

```
UINT nID =
    RegisterClipboardFormat(_T("CircleFormat"));
```

- Lấy khối bộ nhớ toàn cục **hGlobal** đủ lưu dữ liệu, sau đó chép tất cả dữ liệu cần thiết định nghĩa hình tròn vào khối nhớ này

```
// Cấp phát khối bộ nhớ đủ lưu cấu trúc hình tròn
HGLOBAL hGlobal =
            GlobalAlloc(GHND,sizeof(CIRCLE));
// Khoá khối để lấy địa chỉ khối
CIRCLE *pGlobal = (CIRCLE*) GlobalLock(hGlobal);
// Chép dữ liệu định nghĩa đường tròn vào khối nhớ
CopyMemory(pGlobal, pData, sizeof(CIRCLE));
// Bỏ khoá khối
GlobalUnlock(hGlobal);
```

[3] … - Chuyển dữ liệu … – DL với định dạng riêng

Ví dụ: (tt)

- Thực hiện thao tác chuyển dữ liệu vào Clipboard:

```
// Mở và làm rỗng Clipboard
OpenClipboard();
EmptyClipboard();
// Chuyển dữ liệu với định dạng nID
// vào Clipboard
SetClipboardData(nID, hGlobal);
// Đóng Clipboard
CloseClipboard();
```


[3] … - Chuyển dữ liệu … – DL với định dạng riêng

- Thêm vào các định dạng khác:
  - Các ứng dụng sử dụng định dạng riêng thường sử dụng thêm một số định dạng chuẩn khác để đặt dữ liệu vào Clipboard. Điều này tạo cơ hội cho các ứng dụng chuẩn thông thường có thể lấy được nội dung dữ liệu
  - *Ví dụ*: Khi được copy, một bảng tính của Excel sẽ được lưu với hơn 30 định dạng, nhờ đó chúng ta có thể sử dụng MS Paint và MS Notepad để nhận dữ liệu. Tuy nhiên, chỉ có một định dạng là thực sự thể hiện đầy đủ định dạng gốc của một bảng tính Excel

[3] … - Chuyển dữ liệu … – DL với định dạng riêng

- Thêm vào các định dạng khác: (tt)
  - Chuyển nhiều định dạng vào Clipboard: gọi liên tiếp các hàm ***SetClipboardData*** giữa 2 thao tác làm rỗng và đóng Clipboard
  - **Ví dụ:**

```
EmptyClipboard();
SetClipboardData(CF_TEXT, hGlobalText);
SetClipboardData(CF_BITMAP, hGlobalBmp);
CloseClipboard();
```

  - Nhận dữ liệu từ Clipboard: sử dụng nhiều lần hàm ***GetClipboardData*** với định dạng thích hợp

[3] … - Chuyển dữ liệu … – DL với định dạng riêng

Lưu ý:

- Khi một chương trình gọi **`EmptyClipboard`**, tất cả handle được giữ bởi Clipboard đều được loại bỏ

Vấn đề:

- Khi chuyển dữ liệu vào Clipboard, chúng ta tạo bản sao của dữ liệu trong một khối nhớ toàn cục và trao khối nhớ này cho Clipboard
- … nếu dữ liệu quá lớn
- … và người dùng không bao giờ dán dữ liệu vào ứng dụng khác

  → gây lãng phí bộ nhớ

Khắc phục:

- Dùng kỹ thuật *Delayed Rendering* (DR)

- DR cho phép một ứng dụng nói rằng: "*Tôi có dữ liệu sẵn sàng trong Clipboard, nhưng tôi sẽ không chép nó vào Clipboard nếu không được ứng dụng nào yêu cầu"*

Phương pháp:

- Dùng ***`SetClipboardData`*** với tham số handle vùng nhớ bằng NULL

[3] … - Chuyển dữ liệu … – Kỹ thuật Delayed Rendering

Phương pháp:…(tt)

- Xử lý các thông điệp:
  - **`WM_RENDERFORMAT`**: cho biết có một ứng dụng yêu cầu dữ liệu. Chủ Clipboard cần xử lý thông điệp này để chuyển dữ liệu vào Clipboard
    - *Chủ Clipboard* (Clipboard owner) là cửa sổ cuối cùng đặt dữ liệu vào Clipboard. Khi một ứng dụng gọi OpenClipboard, Windows lưu handle cửa sổ yêu cầu bởi hàm này. Đây là handle của cửa sổ mở Clipboard. Khi nhận một lời gọi **`EmptyClipboard`**, Windows sẽ đặt cửa sổ này là *chủ Clipboard*
    - *Lưu ý*: Chủ Clipboard không được gọi hàm **`OpenClipboard`** và **`CloseClipboard`** lần nữa

[3] … - Chuyển dữ liệu … – Kỹ thuật Delayed Rendering

- Xử lý các thông điệp:…(tt)
  - **`WM_RENDERALLFORMATS`**: được Windows gởi cho chủ Clipboard trước khi ứng dụng chủ Clipboard kết thúc. Khi đó, Clipboard vẫn còn chứa handle dữ liệu NULL
    - Chủ Clipboard cần: mở Clipboard, xóa Clipboard, chuyển tất cả dữ liệu vào Clipboard và đóng nó lại
  - **`WM_DESTROYCLIPBOARD`**: được Windows gởi cho chủ Clipboard để thông báo cho ứng dụng biết nó không còn trách nhiệm với dữ liệu trong Clipboard nữa (xảy ra khi hàm **`EmptyClipboard`** được gọi). Ứng dụng có thể giải phóng tài nguyên đã sử dụng cho trường hợp Delayed Render.

[3] … - Chuyển dữ liệu … – Kỹ thuật Delayed Rendering

Ví dụ:

```
/*****************************************************
    Chuyển bitmap vào Clipboard với DR
*****************************************************/
// MessageMap
ON_COMMAND(ID_EDIT_COPY, OnEditCopy)
ON_WM_RENDERFORMAT()
ON_WM_RENDERALLFORMATS()

// Chép handle NULL vào Clipboard khi user Copy
void CMyWindow::OnEditCopy()
{
    OpenClipboard();
    EmptyClipboard();
    SetClipboardData(CF_BITMAP, NULL);
    CloseClipboard();
}
```


[3] … - Chuyển dữ liệu … – Kỹ thuật Delayed Rendering

Ví dụ:…(tt)

```
// Xử lý thông điệp WM_RENDERFORMAT
// nFormat là giá trị wParam trong thông điệp,
// chứa định dạng do chương trình khác yêu cầu
void CMyWindow::OnRenderFormat(UINT  nFormat)
{
    if (nFormat == CF_BITMAP)
    {
       // Tạo bản sao của bitmap, là hBitmap
       ……
       // Đưa dữ liệu vào clipboard
       SetClipboardData(CF_BITMAP, hBitmap);
    }
}
```

[3] … - Chuyển dữ liệu … – Kỹ thuật Delayed Rendering

Ví dụ:…(tt)

```
// Xử lý WM_RENDERALLFORMATS
void CMyWindow::OnRenderAllFormats()
{
   OpenClipboard();
   EmptyClipboard();
   OnRenderFormat(CF_BITMAP);
   CloseClipboard();
 }
```


[3] Các kỹ thuật … - Nhận dữ liệu từ Clipboard

- Các bước cơ bản
- Ví dụ - Nhận dữ liệu với định dạng chuẩn
- Ví dụ - Nhận dữ liệu với định dạng riêng

[3] … - Nhận dữ liệu … - Các bước cơ bản

- Quy trình nhận dữ liệu từ Clipboard:
  - *Bước 1:* Mở Clipboard
  - *Bước 2:* Sử dụng **`GetClipboardData`** để nhận handle của *khối bộ nhớ toàn cục* hoặc của *các đối tượng khác* với định dạng dữ liệu xác định

    ```
    HANDLE  GetClipboardData(UINT uFormat );
    ```

    Nếu Clipboard không chứa dữ liệu với định dạng ***`uFormat`***, hàm trả về NULL
  - *Bước 3:* Tạo dữ liệu sao chép của dữ liệu trong khối bộ nhớ
  - *Bước 4:* Đóng Clipboard

[3] … - Nhận dữ liệu … - Các bước cơ bản

- Lưu ý:
  - Handle nhận từ **`GetClipboardData`** không thuộc chương trình gọi, do đó phải thực hiện sao chép. Chương trình không thể giải phóng hay thay đổi dữ liệu do handle này tham chiếu
  - Các bước sao chép dữ liệu khối nhớ:
    - Khoá khối để nhận con trỏ
    - Sao chép dữ liệu con trỏ ra bên ngoài
    - Bỏ khoá khối

[3] … - Nhận dữ liệu … - Ví dụ (Định dạng chuẩn)

VD.1: Nhận text từ Clipboard

```
char  *szText;              // Chứa dữ liệu
if (OpenClipboard())
{
    // Nhận handle vùng nhớ
    HANDLE hData = GetClipboardData(CF_TEXT);
    if (hData != NULL)
    {
         // Lấy con trỏ tương ứng
         char *pData = (char *)GlobalLock(hData);
         szText = new char[strlen(pData)+1];
         strcpy(szText, pData);   // Copy dữ liệu
         GlobalUnlock(hData);     // Bỏ khoá khối
    }
    // Đóng Clipboard
    CloseClipboard();
}
```


[3] … - Nhận dữ liệu … - Ví dụ (Định dạng chuẩn)

Ví dụ 2: Nhận bitmap từ Clipboard

```
// Mở Clipboard
if (OpenClipboard())
{
    // Lấy handle bitmap
    HBITMAP hBitmap =
          (HBITMAP) GetClipboardData(CF_BITMAP);
    if (hBitmap != NULL)
    {
          //  Tạo bản sao chép của bitmap
          ……
    }
    // Đóng Clipboard
    CloseClipboard();
 }
```

[3] … - Nhận dữ liệu … - Ví dụ (Định dạng chuẩn)

VD.3: Nhận dữ liệu định dạng **CF_HDROP**

- Khi nhận dữ liệu định dạng **CF_HDROP** từ Clipboard, chúng ta có thể sử dụng hàm ***DragQueryFile*** để đọc danh sách các tên file trong khối bộ nhớ
  - **UINT DragQueryFile(HDROP hDrop, UINT iFile, LPTSTR lpszFile, UINT cch)**
    - ***hDrop*** : handle cấu trúc **HDROP**
    - ***iFile*** : chỉ số trong danh sách. Giá trị -1: hàm trả về số lượng phần tử của danh sách
    - ***lpszFile*** : lưu chuỗi kết quả
    - ***cch*** : số ký tự của **lpszFile**

[3] … - Nhận dữ liệu … - Ví dụ (Định dạng chuẩn)

```
Ví dụ 3:…(tt)
if (OpenClipboard())
{
   // Lấy handle khối nhớ
   HDROP hDrop = (HDROP) GetClipboardData(CF_HDROP);
   if (hDrop)  {
       // Đếm số tên flle trong danh sách
       int nCount = DragQueryFile(hDrop, (UINT) -1, NULL, 0);
       // Duyệt danh sách tên file
       if  (nCount)  {
          TCHAR  szFile[MAX_PATH];
          for (int i=0; i < nCount; i++)  {
              DragQueryFile(hDrop, i, szFile,
                                 sizeof(szFile)/sizeof(TCHAR));
              //  Xử lý tương ứng với từng tên file
              ……................
          }
       }
   }
   CloseClipboard();
}
```

[3] … - Nhận dữ liệu … - Ví dụ (Định dạng riêng)

VD.4: Nhận cấu trúc mô tả hình tròn

```
struct CIRCLE  strCircle; //  Lưu kết quả
if (OpenClipboard())
{
    HANDLE hData = GetClipboardData(nID);
    struct CIRCLE *p =
         (struct CIRCLE *) GlobalLock(hData);
    //  Sao chép dữ liệu
    strCircle.center = p→center;
    ……
    GlobalUnlock(hData);
    CloseClipboard();
}
```


[3] Các kỹ thuật … - Truy vấn trên nhiều định dạng

- Các hàm thường dùng
- Các ví dụ

[3] Các kỹ thuật … - Truy vấn trên nhiều định dạng

- Xảy ra khi trong Clipboard có nhiều dữ liệu với các định dạng khác nhau.
- Các hàm thường sử dụng:

Tên hàm	Mô tả
`CountClipboardFormats`	Đếm số định dạng trong Clipboard
`EnumClipboardFormats`	Duyệt qua các định dạng
`IsClipboardFormatAvailable`	Xác định một định dạng có trong Clipboard hay không
`GetPriorityClipboardFormat`	Xác định định dạng đầu tiên có trong Clipboard từ một danh sách ưu tiên các định dạng

[3] Các kỹ thuật … - Truy vấn trên nhiều định dạng

VD.1: Xác định CF_TEXT có trong Clipboard ?

```
if (OpenClipboard())
{
 if (IsClipboardFormatAvailable(CF_TEXT))
 {
      //  Lấy dữ liệu …
      ……
 }
 CloseClipboard();
}
```

[3] Các kỹ thuật … - Truy vấn trên nhiều định dạng

VD.2: Duyệt qua danh sách các định dạng

```
if (OpenClipboard())
{
    UINT nFormat = 0; // Bắt đầu duyệt từ vị trí 0
    while (nFormat = EnumClipboardFormats(nFormat))
    {
          // nFormat chứa ID của format kế tiếp trong
          // Clipboard
          …
     }
    CloseClipboard();
}
```


[3] Các kỹ thuật … - Truy vấn trên nhiều định dạng

VD.3: **truy xuất dữ liệu trong Clipboard dựa trên 1 danh sách các định dạng có độ ưu tiên**

```
UINT nFormats[3] = {
      nID,          //  ưu tiên 1
      CF_TEXT,      //  ưu tiên 2
      CF_BITMAP     //  ưu tiên 3
};
if (OpenClipboard())
{
   UINT nFormat = GetPriorityClipboardFormat(nFormats, 3);
   if (nFormat > 0) {
        // nFormat chứa định dạng của dữ liệu đầu tiên trong
        // Clipboard trùng với nID, CF_TEXT, hoặc CF_BITMAP
        ……
   }
   CloseClipboard();
}
```

[3] Các kỹ thuật … - Xây dựng Clipboard Viewer

- Chuỗi xích Clipboard Viewer
- Các thông điệp liên quan
- Một số hàm thông dụng
- Ví dụ

[3] … - Clipboard Viewer - Chuỗi xích Clipboard Viewer

- Các ứng dụng Clipboard Viewer phải tham gia vào "chuỗi xích Clipboard Viewer"
- Trong chuỗi xích Clipboard Viewer chỉ có một Clipboard Viewer hiện hành trực tiếp nhận thông điệp từ Windows khi có sự thay đổi nội dung Clipboard hoặc thay đổi chuỗi xích Clipboard Viewer
- Các Clipboard Viewer có nhiệm vụ gửi thông điệp đến cho Clipboard Viewer đứng kề sau nó trong chuỗi xích Clipboard Viewer (trừ khi nó là viewer cuối cùng)

[3] … - Clipboard Viewer - Chuỗi xích Clipboard Viewer

- Ứng dụng tham gia vào chuỗi xích Clipboard Viewer bằng cách gọi hàm **`SetClipboardViewer`**:

```
HWND hWndNextViewer =
        SetClipboardViewer(hWndNewViewer);
(MFC) HWND CWnd::SetClipboardViewer()
```

  - **`hWndNewViewer`**: handle của cửa sổ mới được đưa vào chuỗi xích Clipboard Viewer. Cửa sổ mới thêm vào sẽ là Clipboard Viewer hiện hành
  - Trả về handle của cửa sổ Clipboard Viewer liền sau trong chuỗi xích, bằng NULL nếu cửa sổ mới là Viewer duy nhất

[3] … - Clipboard Viewer - Chuỗi xích Clipboard Viewer

- Các Clipboard Viewer phải lưu giữ handle của Viewer kề sau nó trong chuỗi xích Clipboard Viewer. (Nhận được khi gọi hàm **`SetClipboardViewer`** và có thể thay đổi khi có thông điệp **`WM_CHANGECBCHAIN`**)
- Các Clipboard Viewer sử dụng hàm ***`SendMessage`*** để gửi thông điệp đến cho Clipboard Viewer liền sau nó (ngoại trừ viewer cuối cùng)

[3] … - Clipboard Viewer - Chuỗi xích Clipboard Viewer

- Khi chương trình muốn gỡ khỏi Clipboard Viewer chain, gọi hàm ***ChangeClipboardChain***:

```
BOOL  ChangeClipboardChain(HWND hWndRemove,
                           HWND hWndNext);
(MFC) BOOL CWnd::ChangeClipboardChain
                           (HWND hWndNext);
```

  - **hWndRemove**: handle của cửa sổ muốn gỡ khỏi chuỗi xích Clipboard Viewer
  - **hWndNext**: handle cửa sổ nằm liền sau **hWndRemove** trong chuỗi xích Clipboard Viewer

[3] … - Clipboard Viewer – Các thông điệp liên quan

- Thông điệp **WM_DRAWCLIPBOARD**:
  - Được gửi đến cho cửa sổ Clipboard Viewer khi có sự thay đổi nội dung Clipboard
  - Viewer khi nhận thông điệp cần cập nhật lại vùng hiển thị của mình cho phù hợp với nội dung Clipboard mới
  - Viewer sau khi xử lý thông điệp sẽ gửi thông điệp đến cho Viewer kế

[3] … - Clipboard Viewer – Các thông điệp liên quan

- Thông điệp **`WM_CHANGECBCHAIN`**:
  - Được gửi đến cho các Clipboard Viewer khi có một Viewer gọi hàm **`ChangeClipboardChain`**.
  - **`wParam`**: handle của cửa sổ được gỡ khỏi chuỗi xích
  - **`lParam`**: handle của cửa sổ liền sau cửa sổ được gỡ khỏi chuỗi.
  - Nếu Viewer nhận ra **`wParam`** là handle của cửa sổ liền sau (**`hWndNextViewer`**), nó sẽ cập nhật lại **`hWndNextViewer =(HWND) lParam`**;
  - …Nếu không, chuyển thông điệp đến cho Viewer kế tiếp.

[3] … - Clipboard Viewer – Các thông điệp liên quan

- Các thông điệp:
  - **`WM_ASKCBFORMATNAME`**
  - **`WM_PAINTCLIPBOARD`**
  - **`WM_SIZECLIPBOARD`**
  - **`WM_VSCROLLCLIPBOARD`**
  - **`WM_HSCROLLCLIPBOARD`**

Được cửa sổ Clipboard Viewer gửi đến chủ Clipboard khi cần hiển thị dữ liệu có định dạng **`CF_OWNERDISPLAY`**.

[3] … - Clipboard Viewer – Một số hàm thông dụng

- **`HWND GetClipboardOwner(void)`:**
  - Trả về handle của cửa sổ hiện đang là chủ Clipboard
  - Nếu Clipboard không có chủ, trả về NULL (Clipboard vẫn có thể có dữ liệu khi không có chủ)

- **`UINT EnumClipboardFormats(UINT uFormat)`:**
  - Liệt kê các định dạng hiện có trong Clipboard
  - Thực hiện việc liệt kê tuần tự với tham số ***`uFormat`*** =0 cho lần gọi hàm đầu tiên, các lần gọi sau ***`uFormat`*** là định dạng được trả về từ lần gọi trước
  - Nếu thành công, trả về định dạng kế tiếp trong Clipboard
  - Nếu trả về 0 và hàm **`GetLastError`** trả về **`ERROR_SUCCESS`**, không còn định dạng nào trong Clipboard nữa

[3] … - Clipboard Viewer – Một số hàm thông dụng

- **`int GetClipboardFormatName(UINT uFormat,`**
  `LPSTR lpszFormatName,`
  `int iMaxLength)`:

  - Hàm lấy tên của một định dạng đã được đăng ký
  - **`uFormat`**: định danh của định dạng
  - **`lpszFormatName`**: con trỏ của buffer sẽ nhận tên định dạng
  - **`iMaxLength`**: chiều dài tối đa buffer

[3] … - Clipboard Viewer – Ví dụ

```
// xử lý thông điệp WM_CREATE
int CMainWindow::OnCreate
          (LPCREATESTRUCT lpCreateStruct)
{
     // tham gia vào Clipboard Viewer chain
     hWndNextViewer =
               SetClipboardViewer();
     ......
     return 0;
}
```


[3] … - Clipboard Viewer – Ví dụ

```
//  xử lý thông điệp WM_CHANGECBCHAIN
void CMainWindow::OnChangeCbChain
          (HWND hWndRemove, HWND hWndAfter)
{
   // cập nhật lại handle window kế tiếp
   // trong chain
   if (hWndRemove==hWndNextViewer)
        hWndNextViewer = hWndAfter;
   else if (hWndNextViewer != NULL)
               SendMessage(hWndNextViewer,
                           WM_CHANGECBCHAIN,
                           (WPARAM)hWndRemove,
                           (LPARAM)hWndAfter);
   return 0;
}
```

[3] … - Clipboard Viewer – Ví dụ

```
// xử lý thông điệp WM_DESTROY
void CMainWindow::OnDestroy() {
     // gỡ khỏi chuỗi xích
     ChangeClipboardChain(hWndNextViewer);
     …
     PostQuitMessage(0);
     return 0;
}
```


[3] … - Clipboard Viewer – Ví dụ

```
// xử lý thông điệp WM_DRAWCLIPBOARD
void CMainWindow::OnDrawClipboard() {
     if (hWndNextViewer != NULL )
          SendMessage(hWndNextViewer,
               WM_DRAWCLIPBOARD, 0, 0L);
     // cập nhật lại vùng hiển thị:
     InvalidateRect(NULL, TRUE);
     return 0;
}
```

[3] … - Clipboard Viewer – Ví dụ

```
// xử lý message WM_PAINT với định dạng
// uFormat = CF_TEXT
void CMainWindow::OnPaint() {
  RECT rc;
  CPaintDC dc(this);
  if (OpenClipboard())  {
      HGLOBAL hGlobal = GetClipboardData(uFormat);
      if (hGlobal != NULL)  {
        LPSTR  lpstr= (LPSTR) GlobalLock(hGlobal);
        GetClientRect(&rc);
        dc.DrawText(lpstr, -1, &rc, DT_LEFT);
        GlobalUnlock(hGlobal);
      }
      CloseClipboard();
  }
}
```


[3] … - Clipboard Viewer – Ví dụ

- Xử lý thông điệp **`WM_PAINT`** với định dạng **`CF_OWNERDISPLAY`**:
  - Một ứng dụng khi đưa dữ liệu vào Clipboard có thể dùng định dạng **`CF_OWNERDISPLAY`**

    ```
    SetClipboardData(CF_OWNERDISPLAY, NULL);
    ```

  - Khi đó, chủ Clipboard có trách nhiệm: Xử lý các thông điệp được gửi đến từ Clipboard Viewer để hiển thị và cập nhật cửa sổ Clipboard Viewer

[3] … - Clipboard Viewer – Ví dụ

```
// Xử lý thông điệp WM_PAINT với định dạng
// CF_OWNERDISPLAY
void CMainWindow::OnPaint() {
  CPaintDC dc(this);

  HWND hWndOwner = GetClipboardOwner();

  // chuẩn bị thông tin cấu trúc PAINTSTRUCT
  // để gởi cho cửa sổ chủ Clipboard
  HGLOBAL hGlobal = GlobalAlloc(MEM_MOVEABLE,
                          sizeof(PAINTSTRUCT));
  LPPAINTSTRUCT lpps = GlobalLock(hGlobal);
  CopyMemory(lpps, &dc.m_ps,
                   sizeof(PAINTSTRUCT));
```


[3] … - Clipboard Viewer – Ví dụ

```
  GlobalUnlock(hGlobal);

  // gửi thông điệp cho cửa sổ chủ Clipboard
  // yêu cầu vẽ
  SendMessage(hWndOwner, WM_PAINTCLIPBOARD,
          (WPARAM) m_hWnd, (LPARAM) hGlobal);

  // huỷ khối nhớ toàn cục sau khi đã sử dụng
  GlobalFree(hGlobal);
}
```

Cám ơn - Hỏi & Đáp

Lập trình C trên Windows

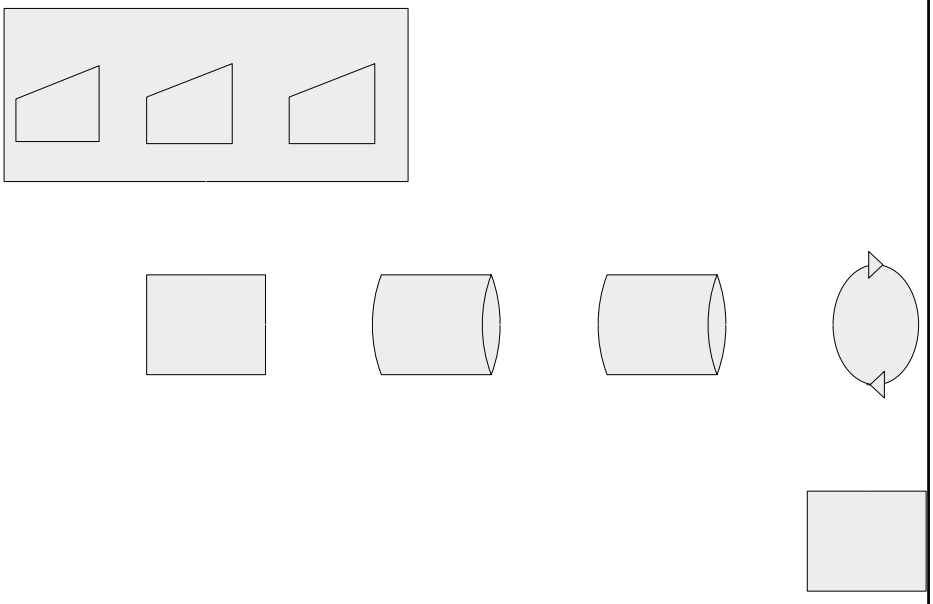
Kỹ thuật lập trình Hook (phụ lục)

Nguyễn Tri Tuấn
Khoa CNTT – ĐH.KHTN.Tp.HCM
Email: nttuan@ fit.hcmuns.edu.vn

Nội dung

- Lập trình sự kiện
- Giới thiệu kỹ thuật Hook
- Minh họa cách lập trình Hook

Lập trình sự kiện



Giới thiệu kỹ thuật Hook

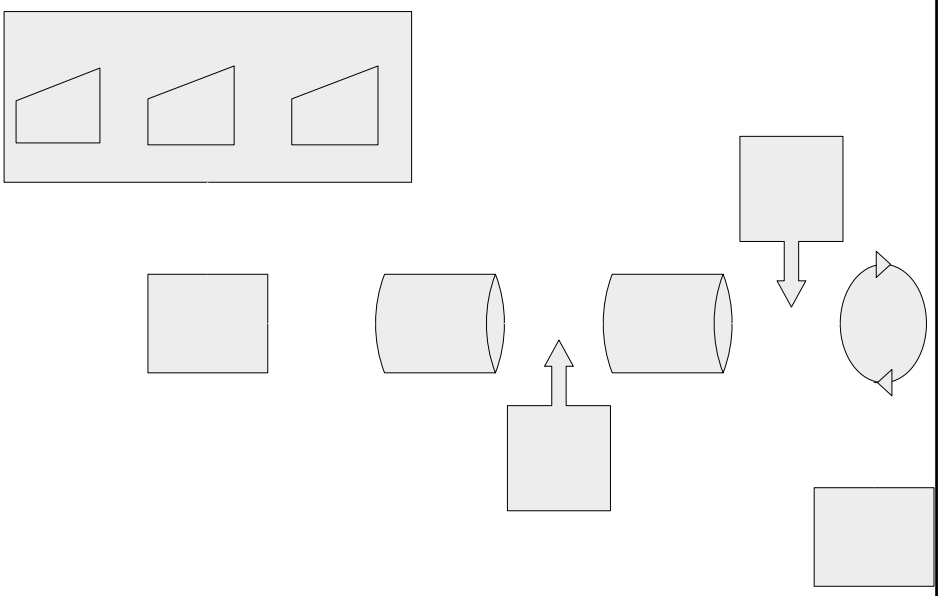
- Hook là gì ?
- Mục tiêu của Hook ?
- Các loại Hook
- Thủ tục Hook (Hook procedure)
- Chuỗi Hook (Hook chain)

Giới thiệu kỹ thuật Hook – Hook là gì ?

- Hook là một cơ chế trong lập trình sự kiện,…
- …cho phép ứng dụng có thể cài đặt một hàm giám sát vào quá trình lưu chuyển các thông điệp

→ ứng dụng có thể chặn và xử lý các thông điệp trước khi nó đến được cửa sổ/ứng dụng đích

Giới thiệu kỹ thuật Hook – Hook là gì ? … (tt)



Giới thiệu kỹ thuật Hook – Hook là gì ? … (tt)

- Có 2 cách cài đặt Hook:
  - Cài đặt cục bộ (Thread Hook): hàm giám sát được cài vào sau Thread message Queue → có tác dụng giám sát tất cả các thông điệp trong một tiểu trình hay một ứng dụng cụ thể
  - Cài đặt toàn cục (Global Hook): hàm giám sát được cài vào sau Systemd message Queue → có tác dụng giám sát tất cả các thông điệp trong toàn hệ thống
- Với Global Hook, hàm cài đặt phải được lưu trong một DLL

Giới thiệu kỹ thuật Hook – Mục tiêu của Hook ?

- Giám sát bàn phím: các ứng dụng gõ tiếng Việt, điều khiển thiết bị bằng bàn phím,…
- Giám sát mouse: Click'n See
- Theo dõi việc sử dụng các ứng dụng, Capture screen
- Ứng dụng dạy học bằng máy tính (CBT – Computer-based Training)
- …

Giới thiệu kỹ thuật Hook – Các loại Hook ?

- WH_KEYBOARD: Hook giám sát thông điệp từ bàn phím: WM_KEYDOWN, WM_KEYUP
- WH_MOUSE: Hook giám sát thông điệp từ chuột
- WH_GETMESSAGE: Hook giám sát thông điệp chung (keyboard, mouse, hay các message khác)
- WH_CBT: Windows gọi hàm hook CBT trước khi tạo lập (create), kích hoạt (active), hủy (destroy), minimize, maximize, di chuyển (move), thay đổi kích thước (size),… của cửa sổ giao diện

Giới thiệu kỹ thuật Hook – Các loại Hook ?...(tt)

- WH_JOURNALPLAYBACK: cho phép đưa message vào System message queue → sử dụng để giả lập hay thực hiện lại 1 dãy các message của bàn phím hay mouse (playback). Đây là một Global Hook
- WH_JOURNALRECORD: giám sát và ghi nhận lại (record) các thông điệp từ chuột và bàn phím. Đây là một Global Hook
- …

Giới thiệu kỹ thuật Hook – Thủ tục Hook (Hook procedure)

- Thủ tục Hook: là hàm dùng để giám sát các thông điệp mà ứng dụng cài vào hệ thống
- Dạng chung của Hook Procudure:

```
LRESULT CALLBACK HookProc(
    int nCode,
    WPARAM wParam,
    LPARAM lParam);
```

  - `nCode`: xác định hành động cần xử lý. Giá trị của nCode tùy thuộc loại Hook
  - `wParam, lParam`: chứa thông tin của message

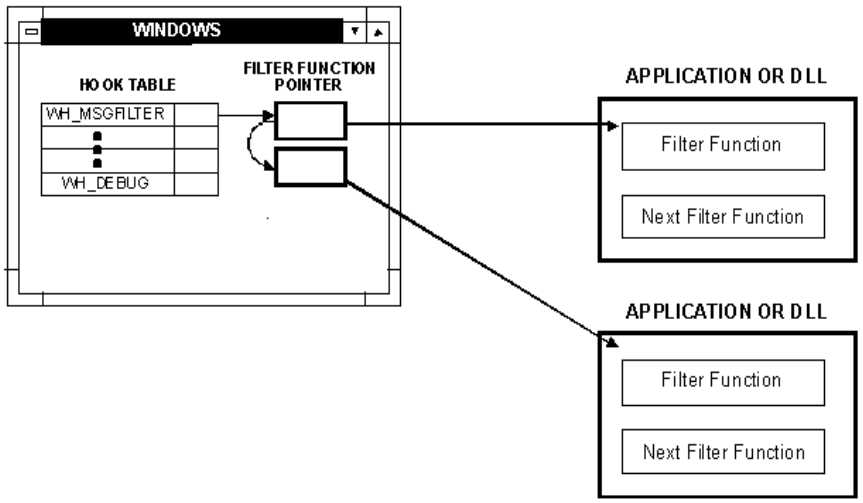
Giới thiệu kỹ thuật Hook – Thủ tục Hook…(tt)

- Mỗi loại Hook cần có cách xử lý khác nhau khi xây dựng Hook Procedure
- Có thể cài đặt nhiều Hook Procedure bằng cách dùng hàm `SetWindowsHook hay SetWindowsHookEx`
- Hook Procedure cài sau sẽ luôn nằm ở vị trí đầu tiên trong dãy thủ tục Hook

Giới thiệu kỹ thuật Hook – Chuỗi Hook (Hook chain)

- Chuỗi Hook: là một dãy các thủ tục Hook được liên kết theo thứ tự độ ưu tiên thực hiện giảm dần
- HĐH Windows quản lý các chuỗi Hook riêng biệt cho từng loại Hook
- Khi có 1 message xảy ra, Windows sẽ gởi message đó đến thủ tục Hook đầu tiên trong chuỗi Hook có loại tương ứng…
- …message sẽ được chuyển lần lượt đến các thủ tục Hook kế tiếp sau đó

Giới thiệu kỹ thuật Hook – Chuỗi Hook…(tt)



Sơ đồ Hook Chain

Minh họa cách lập trình Hook

- Các hàm thao tác với Hook
- Cài đặt thủ tục Hook
- Ví dụ thủ tục Hook
- Chuyển message cho thủ tục Hook kế tiếp
- Hủy bỏ cài đặt Hook

Minh họa cách lập trình Hook - Các hàm thao tác với Hook

- `SetWindowsHookEx`
- `CallNextHookEx`
- `UnhookWindowsHookEx`

Minh họa cách lập trình Hook - Cài đặt thủ tục Hook

- Một ứng dụng cần phải thực hiện việc Cài đặt thủ tục Hook khi muốn giám sát message
- Hàm *SetWindowsHookEx* sẽ cài đặt thủ tục Hook vào điểm bắt đầu của chuỗi Hook

```
HHOOK SetWindowsHookEx(
    int hookMsg, HOOKPROC hookProc,
    HINSTANCE hIns, DWORD  threadId);
```

  - *hookMsg:* loại Hook
  - *hookProc*: con trỏ đến thủ tục Hook. Trường hợp Global Hook, thủ tục Hook phải lưu trong DLL; với Thread Hook, thủ tục Hook có thể chứa trong chính thread tương ứng
  - *hIns*: handle của module chứa thủ tục Hook
  - *threadId:* ID của thread. Nếu là 0, Hook sẽ là Global

Minh họa cách lập trình Hook - Cài đặt thủ tục Hook…(tt)

Ví dụ 1: cài đặt Keyboard Hook toàn cục (load-time)

```
SetWindowsHookEx(WH_KEYBOARD,
      (HOOKPROC)fnKeyboardProc, hInstDLL, 0);
```

Ví dụ 2: cài đặt Keyboard Hook toàn cục (run-time)

```
HOOKPROC fnKeyboardProc;
static HINSTANCE hInstDLL;
static HHOOK hHook;
hInstDLL = LoadLibrary((LPCTSTR) "myKBDLL.dll");
fnKeyboardProc = (HOOKPROC)GetProcAddress(hInstDLL,
                                "KeyboardProc");
hHook = SetWindowsHookEx(WH_KEYBOARD,
             fnKeyboardProc, hInstDLL, 0);
```

Ví dụ 3: cài đặt Keyboard Hook cục bộ

```
SetWindowsHookEx(WH_KEYBOARD, (HOOKPROC)fnKeyboardProc,
  NULL, GetCurrentThread());
```

Minh họa cách lập trình Hook - Ví dụ thủ tục Hook

Ví dụ 4: Thủ tục hook cho Keyboard

```
LRESULT CALLBACK KeyboardProc(int nCode,
     WPARAM wParam, LPARAM lParam)
{
  if (nCode >= 0 && nCode == HC_ACTION) {
     pMsg = (MSG *)lParam;
     if (pMsg->message == WM_KEYDOWN) {
          char s[] = {LOBYTE(wParam),'\0'};
          MessageBox(NULL, s, "Hook", 0);
     }
  }

  return CallNextHookEx(hHook, nCode,
                        wParam, lParam);
}
```


Lập trình Hook - Chuyển message cho thủ tục Hook kế tiếp

- Sau khi thực hiện xong, thủ tục Hook sẽ gọi hàm `CallNextHookEx` để chuyển message đến thủ tục Hook kế tiếp trong chuỗi Hook

```
LRESULT CallNextHookEx(
    HHOOK hHook, int code,
    WPARAM wParam, LPARAM lParam);
```

  - `hHook`: handle của Hook (hiện hành) nhận về từ hàm `SetWindowsHookEx`
  - `code, wParam, lParam`: các giá trị của thủ tục Hook hiện tại truyền cho thủ tục Hook kế tiếp trong chuỗi Hook

Lập trình Hook - Chuyển message cho thủ tục Hook kế tiếp…

- Thủ tục Hook có thể không chuyển thông điệp đến thủ tục Hook kế tiếp trong chuỗi Hook
- *Lưu ý*: việc không chuyển message có thể gây ra lỗi nghiêm trọng cho hệ thống

Lập trình Hook - Hủy bỏ cài đặt Hook

- Sử dụng kỹ thuật Hook sẽ làm giảm khả năng thực thi của hệ thống
- Do đó, khi không sử dụng Hook nữa nên hủy bỏ Hook khỏi hệ thống

```
BOOL UnhookWindowsHooks(
    HHOOK hHook);
```

- `hHook`: handle của hook cần hủy bỏ

Cám ơn - Hỏi & Đáp

Lập trình C trên Windows

Thư viện lập trình Multi-Media

Nguyễn Tri Tuấn
Khoa CNTT – ĐH.KHTN.Tp.HCM
Email: nttuan@ fit.hcmuns.edu.vn

Nội dung

- Mục tiêu
- Các kiến thức cơ bản về Multi-Media
- Các thư viện lập trình Multi-Media trên Windows
- PlaySound
- Thư viện MCI

Mục tiêu

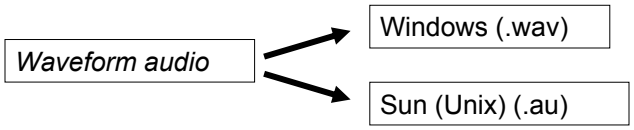
- Giới thiệu các kiến thức khái quát về Multi-Media
- Giới thiệu các thư viện lập trình điều khiển thiết bị Multi-Media trên HĐH Windows
- Trình bày chi tiết thư viện PlaySound, MCI
- Có thể xây dựng các ứng dụng Multi-Media đơn giản
  - Play WAV file
  - Play MIDI file
  - Play CD Audio
- Có kiến thức cơ bản để tìm hiểu sâu hơn kỹ thuật lập trình Multi-Media trên Windows

Các kiến thức khái quát về Multi-Media

- Có rất nhiều định dạng media khác nhau tuỳ thuộc vào sự phát triển cụ thể của các công ty phần mềm (.aiff, .wav, .midi, .au, .rmf, .avi, .mpeg,…)
- Hai dạng chuẩn thông dụng nhất của Audio là: waveform audio (sampled sound) và Musical Instrument Digital Interface (MIDI)
- Dạng chuẩn thông dụng nhất của Video là avi

Các kiến thức khái quát về Multi-Media…(tt)

- WAV (Waveform Audio):
  - Được tạo ra bằng cách thu âm trực tiếp
  - Quá trình thu được thực hiện bằng cách lấy mẫu rời rạc sau mỗi chu kỳ thời gian…
  - … và được mã hoá bằng các giải thuật PCM (Pulse Code Modulation) hay ADPCM (Adaptive Pulse Code Modulation)
  - Mật độ lấy mẫu, lượng thông tin lưu trên mẫu (8,16, hay 32 bits) sẽ quyết định chất lượng âm thanh



Các kiến thức khái quát về Multi-Media…(tt)

- MIDI:
  - Dùng để ghi, phát, soạn thảo âm thanh được tạo bởi các nhạc cụ điện tử
  - Ứng với mỗi nốt nhạc và các dụng cụ nhạc khí tạo ra nó, MIDI sẻ gán cho một số tương ứng, cũng giống như cách những kí tự được tượng trưng bởi một số trong mã ASCII
  - Ngoài ra, còn có các tiêu chuẩn cho các đặc tính khác như trường độ hay nhịp độ của bản nhạc
  - MIDI thường có kích thước nhỏ
  - Không có cách nào để thực hiện ghi hay phát lại những "real" sounds như tiếng nói hay hiệu ứng đặc biệt

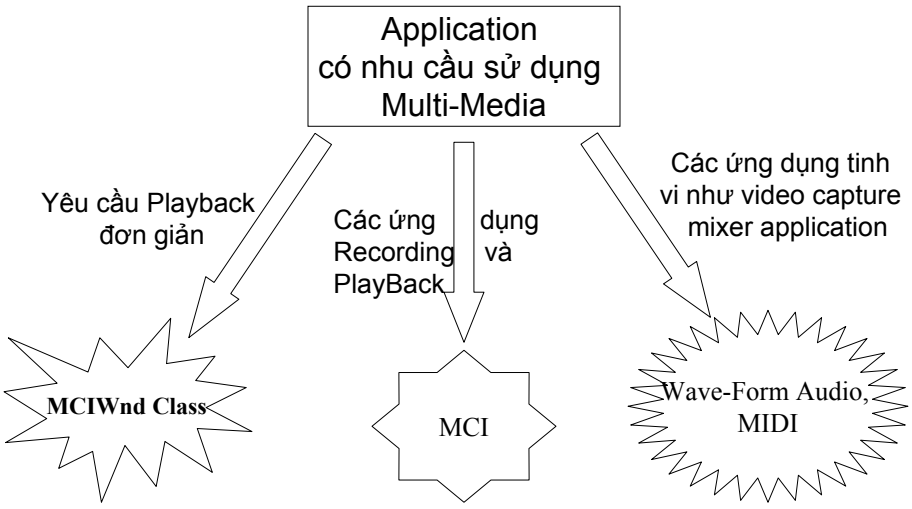
Các kiến thức khái quát về Multi-Media…(tt)

- MDI…(tt)
  - Chuẩn MIDI là phương thức được chọn để ghi và phát âm thanh và cũng là phương pháp duy nhất để kết nối các dụng cụ âm nhạc như thiết bị chơi nhạc điện tử đa năng, trống ... với máy tính và với nhau
  - Các studio âm thanh thương mại thường thực hiện việc hoà trộn giữa waveform audio và Midi file để tạo ra thành phẩm âm thanh cuối cùng

Các thư viện lập trình Multi-Media trên Windows

- **PlaySound**
- **Thư viện MCI (Media Control Interface)**
- MCIWnd Window class
  (MSDN / Platform SDK / Graphics and Multi-media services / Windows Multi-media / Multi-media Audio / MCIWnd Window class)
- Thư viện MIDI (Musical Instrument Digital Interface)
  (MSDN / Platform SDK / Graphics and Multi-media services / Windows Multi-media / Multi-media Audio / MIDI)
- Wave-form Audio
  (MSDN / Platform SDK / Graphics and Multi-media services / Windows Multi-media / Multi-media Audio / Wave-form Audio)
- Windows Media Player
- DirectX
  (MSDN / Platform SDK / Graphics and Multi-media services / DirectX)

Các thư viện lập trình Multi-Media trên Windows…(tt)



Chọn lựa thư viện cho ứng dụng

PlaySound

- Hàm đơn giản, dùng để playback nhanh một file audio
- **`BOOL PlaySound(LPCSTR pszSound, HMODULE hmod, DWORD fdwSound);`**
  - **`pszSound`**: Chuỗi chỉ định audio cần play. Audio có thể là file, resource hay alias (được đăng ký sẵn trong registry hay WIN.INI)
  - **`hmod`**: Handle của ứng dụng chứa audio resource
  - **`fdwSound`***:* Các option dùng cho việc play (**`SND_ALIAS, SND_FILENAME, SND_RESOURCE, SND_ASYNC,…`**)

PlaySound…(tt)

VD1. Play file Wave trên đĩa

```
PlaySound("C:/CHORD.WAV", 0, SND_FILENAME);
```

VD2. Play nhạc theo kiểu Background

```
PlaySound("C:/CHORD.WAV", 0, SND_ASYNC |
                             SND_FILENAME);
```

VD3. Play nhạc theo dạng lặp lại

```
PlaySound("C:/CHORD.WAV", 0, SND_LOOP |
                  SND_ASYNC | SND_FILENAME);
```

VD4. Ngừng play

```
PlaySound(0, 0, 0);
```


PlaySound…(tt)

VD5. Play Waveform theo các khoá trong section [Sound] của WIN.INI

```
// WIN.INI
// [Sound]
// MouseClick = C:\CHORD.WAV
PlaySound("MouseClick", 0, SND_ALIAS |
                           SND_NODEFAULT);
```

VD6. Play file WAVE được add sẵn trong resources

```
// resource
// IDR_WAVE1 WAVE DISCARDABLE "c:/chord.wav"
// IDR_WAVE1 129
PlaySound("#129", 0, SND_RESOURCE |
                     SND_NODEFAULT);
```

PlaySound…(tt)

Ưu điểm

- Dễ dàng play Audio
- Cho phép play ở dạng background.
- Quản lí hoàn toàn việc mở, loading, và đóng file

Khuyết điểm

- Chỉ play dạng chuẩn dành cho audio kĩ thuật số trên nền tảng PC của Intel.
- File audio phải có kích thước thích hợp với bộ nhớ có sẵn.
- Không thể được sử dụng một cách đồng thời bởi nhiều tiểu trình trong cùng một tiến trình.
- Cung cấp ít khả năng điều khiển.

Thư viện MCI

- Giới thiệu
- MCI Command String
- MCI Command Message
- Xử lý lỗi
- Nhận xét

Thư viện MCI - Giới thiệu

- MCI (Media Control Interface) là thư viện cấp trung (mid-level interface) bao gồm các chỉ thị hay các thủ tục giao tiếp, điều khiển file và các thiết bị multimedia
- Tập lệnh MCI cung cấp không phụ thuộc thiết bị
- Cho phép ứng dụng điều khiển các file audio/video và các thiết bị ngoại vi một cách độc lập
- Cho phép điều khiển hầu hết các thiết bị Multi-Media bao gồm: wave audio, CD audio, MIDI sequencers, video số,…

Thư viện MCI - Giới thiệu…(tt)

- Trong Windows, các MCI drivers phổ biến sẽ được tích hợp sẵn
- Mỗi thiết bị multimedia mới, như một sound card hay một video card, sẽ có MCI drivers đi kèm để điều khiển nó

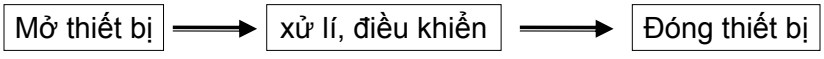
Thư viện MCI - Giới thiệu…(tt)

Tên thiết bị	
animation	Animation device
cdaudio	Audio CD player
Dat	Digital audio tape player
Digitalvideo	Digital video in a window (not GDI based)
Overlay	Overlay device (analog video in a window)
Scanner	Image scanner
Sequencer	MIDI sequencer
vcr	Videotape recorder or player
Videodisc	Videodisc player
Waveaudio	Audio device that plays digitized waveform files
Other	Undefined MCI device

Các thiết bị được nhận biết bởi phiên bản hiện thời của MCI

Thư viện MCI - Giới thiệu…(tt)

- Có 2 phương pháp lập trình MCI:
  - Command string (**mciSendString**)
  - Command message (**mciSendCommand**)
- Tất cả MCI devices đều hỗ trợ một tập chuẩn các MCI commands và messages. Nhiều devices hỗ trợ thêm các lệnh đặc biệt bổ sung mở rộng
- Qui trình xử lý chung của MCI



Thư viện MCI - MCI Command String

- Điều khiển thiết bị Multi-Media bằng câu lệnh ở dạng chuỗi
- Các câu lệnh bắt đầu bằng từ khóa **`open, play, stop,`**…
- Ứng với mỗi tác vụ khác nhau chúng ta sẽ có câu lệnh với các định dạng khác nhau nhưng nhìn chung chúng có cùng một cấu trúc tổng quát (ngoại trừ tác vụ open).
- Câu lệnh được gởi đến thiết bị thông qua hàm **`mciSendString,`** định nghĩa trong thư viện **`<Winmm.lib>`**

Thư viện MCI - MCI Command String…(tt)

- **`MCIERROR mciSendString(LPCTSTR lpszCommand,`**
  `LPTSTR lpszReturnString,`
  `UINT cchReturn,`
  `HANDLE hwndCallback);`
  - **`lpszCommand:`** Chuỗi câu lệnh gởi tới các thiết bị multimedia để yêu cầu thực hiện các thao tác: **`close, play, stop`**…
    - **`wsprintf(lpszCommand, "Tên tác vụ %s [%s] %s", lpszDeviceID, <Các cờ đặc biệt dành riêng có thể có hoặc không>, lpszFlags);`**
    - **`lpszDeviceID`**: Là số ID dùng xác định thiết bị MCI. ID này cũng chính là Alias được gán trong câu lệnh mở (open) thiết bị
    - **`lpszFlags:`** có thể là "wait“, “notify”, “test”.

Thư viện MCI - MCI Command String…(tt)

- **mciSendString…(tt)**
  - **lpszReturnString:** Chuỗi chứa thông tin phản hồi từ thiết bị media (=NULL nếu ta không quan tâm đến thông tin phản hồi)
  - **cchReturn:** Độ dài tính theo số kí tự của **lpszReturnString**
  - **hwndCallback:** Handle của cửa sổ để xử lí các message do thiết bị multimedia gởi mỗi khi có sự kiện nào đó (như chơi hết file, đóng file,…).
    - Tham số náy chỉ có ý nghĩa khi trong câu lệnh gởi tới thiết bị có chứa cờ notify.
    - Message được gởi là **MM_MCINOTIFY**
    - = NULL nếu không quan tâm đến việc nhận message phản hồi

Thư viện MCI - MCI Command String…(tt)

```
VD1.Mở thiết bị
char  szMCIString[128];
char  szDevType[30];
char  szFileName[128];
MCIERROR    dwReturn;
strcpy(szFileName,"C:/Windows/Windows_Start.wav");
strcpy(szDevType, "waveaudio");
sprintf(szMCIString, "open %s type %s alias MyWAV",
      szFileName, szDevType);
dwReturn = mciSendString(szMCIString, NULL, 0, NULL);

VD2. Play
mciSendString("play MyWAV notify", NULL, 0, m_hWnd);

VD3. Ngừng play
mciSendString("stop MyWAV", NULL, 0, NULL);

VD4. Đóng thiết bị
mciSendString("close MyWAV", NULL, 0, NULL);
```

Thư viện MCI - MCI Command String…(tt)

Loại thiết bị	Từ khóa
CD Audio	**`type cdaudio`**
Digital-video playback	**`type digitalvideo`**
MIDI	**`type sequencer`**
Video-cassete recorder	**`type vcr`**
Video-Disc player	**`type videodisc`**
Wave audio	**`type waveaudio`**

Chọn loại thiết bị trong cú pháp lệnh Command String

Thư viện MCI - MCI Command String…(tt)

- Ưu điểm
  - Lệnh bằng ngôn ngữ tự nhiên dễ hiểu
- Khuyết điểm
  - Không xử lý được các file có tên chứa khoảng trắng

Thư viện MCI - MCI Command Message

- Điều khiển các thiết bị Multi-Media bằng cách sử dụng các thông điệp (message)
- Mỗi lệnh điều khiển tương ứng với 1 message khác nhau
- Message được gởi đến thiết bị thông qua hàm **`mciSendCommand <Winmm.lib>`**

Thư viện MCI - MCI Command Message…(tt)

- **`MCIERROR mciSendCommand(MCIDEVICEID IDDevice,`**
  `UINT uMsg,`
  `DWORD fdwCommand,`
  `DWORD dwParam);`
  - **`IDDevice:`** ID của thiết bị cần điều khiển, thông số này không dùng cho thông điệp **`MCI_OPEN`**, giá trị này sẽ có được sau khi thực hiện **`MCI_OPEN`**
  - **`uMsg:`** Thông điệp hay chỉ thị được gởi tới thiết bị, nhằm điều khiển theo những mục đích cụ thể, Đó là những thông điệp được biểu diễn dưới dạng những hằng số được Windows định nghĩa trước **`MCI_xxx`**
  - fdwCommand: là giá trị được xem như những tham số bổ sung tương ứng với từng thông điệp.
  - dwParam: con trỏ chỉ tới cấu trúc lưu thông tin ứng với từng thông điệp do mỗi thông điệp có những đặc thù riêng. Ví dụ ứng với thông điệp **`MCI_PLAY`**, chúng ta có cấu trúc tương ứng là **`MCI_PLAY_PARMS`**

Thư viện MCI - MCI Command Message…(tt)

VD1. Mở thiết bị

```
MCI_OPEN_PARMS    mciOpenParams;
mciOpenParams.lpstrDeviceType = "waveaudio";
mciOpenParams.lpstrElementName = "C:/MySound.wav";
DWORD mciError = mciSendCommand(NULL, MCI_OPEN,
                  MCI_OPEN_TYPE | MCI_OPEN_ELEMENT,
                 (DWORD)(LPVOID) &mciOpenParams);
mciDevID = mciOpenParams.wDeviceID;

VD2. Play
MCI_PLAY_PARMS mciPlayParms;
mciPlayParms.dwCallback = (unsigned long) m_hWnd;
DWORD mciError = mciSendCommand(mciDevID, MCI_PLAY,
            MCI_NOTIFY, (DWORD)(LPVOID) &mciPlayParms);
```


Thư viện MCI - MCI Command Message…(tt)

```
VD3. Ngừng play
DWORD mciError =
     mciSendCommand(mciDevID, MCI_STOP, 0, 0);

VD4. Đóng thiết bị
DWORD mciError =
     mciSendCommand(mciDevID, MCI_CLOSE, 0, NULL);
```

Thư viện MCI - MCI Command Message…(tt)

- Ưu điểm
  - Sử dụng message → quen thuộc với các lập trình Windows
  - Lệnh có cú pháp rõ ràng, dễ nhớ
  - Xử lý được file có tên chứa khoảng trắng
- Khuyết điểm
  - Cần nắm rõ các cấu trúc dữ liệu tương ứng với mỗi thông điệp

Thư viện MCI - Xử lý lỗi

- **`mciSendString`** và **`mciSendCommand`** có giá trị trả về là **`MCIERROR`**
  - 0: lệnh được thực hiện thành công;
  - Khác 0: là mã lỗi ở dạng số
- **MCI** cung cấp cho chúng ta hàm **`mciGetErrorString`** để nhận thông báo lỗi tương ứng ở dạng chuỗi
- **`BOOL mciGetErrorString(DWORD fdwError,`**
  `LPTSTR lpszErrorText,`
  `UINT cchErrorText);`
  - **`fdwError`**: Đây chính là mã lỗi nhận được từ các hàm **`mciSendString`** và **`mciSendCommand`**
  - **`lpszErrorText`**: Chuỗi chứa nội dung thông báo lỗi (dài tối đa 128 ký tự)
  - **`cchErrorText`**: Độ dài của chuỗi **`lpszErrorText`**

Thư viện MCI - Xử lý lỗi…(tt)

```
VD. Xác định chuỗi thông báo lỗi
if (dwReturn = mciSendCommand(mciDevID, MCI_PLAY,
     MCI_NOTIFY, (DWORD)(LPVOID) &mciPlayParams))
{
  char szError[128];
  mciSendCommand(mciDevID, MCI_CLOSE, 0, 0L);
  mciGetErrorString(dwReturn, szError, 128);
  MessageBox(szError, "Error", MB_OK);
  return;
}
```


Thư viện MCI - Nhận xét

- Ưu điểm
  - Dễ sử dụng
  - Không cần hiểu biết nhiều về các thiết bị Multi-Media, các cấu trúc file
  - Sử dụng cùng một cơ chế cho các thiết bị: Wave audio, MIDI, CD Audio, AVI,…
  - Độc lập thiết bị
- Khuyết điểm
  - Do xử lý ở cấp cao nên phụ thuộc vào giao diện hàm MCI có sẵn, không thể can thiệp vào việc mở rộng khả năng ứng dụng Multimedia như hiệu ứng âm thanh, graphics equalizer,…
  - Đối với xử lý âm thanh wave audio thì MCI chỉ xử lý được file wave chuẩn dạng PCM

Cám ơn - Hỏi & Đáp