

LÊ NGỌC THANH
lntmail@yahoo.com

LẬP TRÌNH WINDOWS VỚI MFC Microsoft Visual C++ 6.0

NHÀ XUẤT BẢN THỐNG KÊ

MỤC LỤC

CHƯƠNG 1 : MỘT SỐ KHÁI NIỆM LẬP TRÌNH TRONG MÔI TRƯỜNG WINDOWS

	<u>Trang</u>
1.1 Chương trình (Program)	1
1.2 Ứng dụng (Application)	1
1.3 Tiến trình (Process)	1
1.4 Tiểu trình (Thread)	1
1.5 Thông điệp (Message)	2
1.5.1 Nguồn gốc của message	2
1.5.2 Các loại message	2
1.5.3 Số hiệu message (Message Identifier – MessageID)	3
1.5.4 Đặc tả message	3
1.6 Cửa sổ giao diện (window) của ứng dụng	3
1.7 Message queue	5
1.8 Kiến trúc xử lý của ứng dụng trong Windows	7
1.9 Resource của ứng dụng	8

CHƯƠNG 2 : THƯ VIỆN MFC CỦA MICROSOFT & ỨNG DỤNG CƠ BẢN TRONG WINDOWS

2.1 Thư viện MFC (Microsoft Foundation Class)	10
2.2 Tiếp □an MFC	10
2.3 Tạo ứng dụng windows với MFC như thế nào ?	10
2.4 Lớp quản lý tiểu trình CwinThread	11
2.5 Lớp quản lý tiểu trình giao diện chính CwinApp	12
2.6 Thực hiện ứng dụng đơn giản	14
2.7 Thực hiện ứng dụng giao tác đơn giản	20
2.8 Tạo mới Icon Resource cho ứng dụng	25
2.9 Lưu trữ chương trình nguồn	27
2.10 Lớp CString của MFC	27

CHƯƠNG 3 : CÁC LỚP GIAO DIỆN ĐỒ HỌA CỦA MFC

3.1 Các công cụ giao diện đồ họa	29
3.2 Device Context	29
3.3 Tọa độ trên giao diện đồ họa	29
3.4 Các lớp MFC hỗ trợ GDI	30
3.4.1 Các lớp đối tượng điểm, hình chữ nhật	30

3.4.2 Lớp Cpen	31
3.4.3 Lớp CBrush	31
3.4.4 Lớp CFont	32
3.4.5 Lớp CBitmap	32
3.4.6 Lớp CPalette	33
3.4.7 Lớp CRgn	34
3.5 Lớp CDC	35
3.6 Lớp CImageList	38

CHƯƠNG 4 : CỬA SỔ GIAO DIỆN LỚP VÀ LỚP CWnd

4.1 Cửa sổ giao diện	40
4.2 Lớp CWnd	40
4.3 Sử dụng đối tượng CWnd	49
4.3.1 Sử dụng CWnd làm giao diện chính của ứng dụng	49
4.3.2 Ứng dụng chỉ chạy một bản (instance) tại mỗi thời điểm	50

CHƯƠNG 5 : XỬ LÝ MESSAGES

5.1 Lớp xử lý message CCmdTarget:	51
5.2 Khai báo mục xử lý message trong MessageMap	52
5.3 Các lớp kế thừa CCmdTarget	55
5.4 MessageMap của lớp kế thừa CWnd trong ứng dụng	55
5.4.1 Cửa sổ của ứng dụng có chức năng hoạt động	55
5.4.2 WM_PAINT và hành vi OnPaint của CWnd	58

CHƯƠNG 6 : ỨNG DỤNG CÔNG CỤ GDI

6.1 DC và BITMAP	60
6.2 Ứng dụng với cửa sổ chính hiển thị ảnh	60
6.3 Sao chép ảnh từ DC đến DC, phóng to & thu nhỏ ảnh	63
6.4 DC trong bộ nhớ (DC ảo) – vùng vẽ đệm lý tưởng	64
6.5 Ảnh chuyển động trong vùng client	65
6.6 CImageList – công cụ quản lý bộ ảnh cùng cỡ	66
6.7 CRgn – Cửa sổ có hình dạng tùy ý	67

CHƯƠNG 7 : MENU – PHÍM TẮT

7.1 Định nghĩa	69
7.2 Menu resoure	69

7.3 Sử dụng menu resource	71
7.4 Mục xử lý command message từ mục chọn của menu	72
7.5 Phím tắt (hot key) cho mục chọn trên menu	73
7.6 Lớp quản lý menu – CMenu	75
7.7 Xử lý điều khiển mục chọn của menu	77

CHƯƠNG 8 : CÁC LỚP ĐỐI TƯỢNG NHẬP LIỆU **(WINDOWS CONTROLS)**

8.1 CStatic	78
8.2 CEdit	80
8.3 CButton	84
8.4 CListBox	85
8.5 CComboBox	88
8.6 CSpinButtonCtrl	91
8.7 CProgressCtrl	93
8.8 CScrollBar	94
8.9 CSliderBar	96

CHƯƠNG 9 : HỘP HỘI THOẠI

9.1 Hộp hội thoại (Dialog)	97
9.2 Lớp CDialog	97
9.3 Tạo và sử dụng dialog trong chương trình	99
9.3.1 Tạo dialog resource	99
9.3.2 Khai báo lớp kế thừa CDialog sử dụng dialog resource	103
9.3.3 Sử dụng dialog trong chương trình	104
9.4 Liên kết giữa dialog và các thành phần khác	104
9.5 Sử dụng dialog làm giao diện chính của ứng dụng	106
9.5.1 Thực hiện ứng dụng với giao diện chính là dialog	106
9.5.2 Dùng MFC wizard tạo ứng dụng với giao diện dialog	107
9.6 Khai báo biến cho control trên dialog	109
9.7 Khai thác các tiện ích hỗ trợ	112

CHƯƠNG 10 : KHUNG CỬA SỔ GIAO DIỆN CHÍNH

10.1 Khung cửa sổ giao diện (Frame Window)	117
10.2 Thanh trạng thái (statusbar) & lớp CStatusBar	117
10.3 Thanh công cụ (toolbar) & lớp CToolBar	119

10.3.1 Thiết kế ToolBar resource	120
10.3.2 Dùng toolbar resource cho CToolBar của FrameWnd	121
10.4 Lớp CFrameWnd	121
10.5 Sử dụng frame window làm giao diện chính	123
10.5.1 Thực hiện ứng dụng với giao diện frame window	123
10.5.2 String Table và CFrameWnd	124
10.5.3 Dùng MFC wizard tạo ứng dụng giao diện framewindow	130

CHƯƠNG 11 : CÁC KIẾN TRÚC DOCUMENT – VIEW

11.1 CDocument	134
11.2 CView	135
11.3 CFrameWnd	136
11.4 CDocTemplate	136
11.5 Hỗ trợ từ phía đối tượng quản lý ứng dụng	137
11.6 Trình tự tạo lập các đối tượng tham gia bộ DVF	138
11.7 Text Document Application	139
11.8 Rich Text Format (rtf) Document Application	143
11.9 HTML Document View Application	146
11.10 Một số lớp view đặc biệt	149
11.10.1 CListView	149
11.10.2 CTreeView	150
11.10.3 CSplitterWnd	152
11.10.4 Sử dụng splitterwnd trong frame window	154
11.10.5 Các ví dụ thực hành	155

CHƯƠNG 12 : MỘT SỐ VẤN ĐỀ TRONG WINDOWS

12.1 Tập tin INI	158
12.2 System Registry	160
12.3 Vùng Status Area trên Taskbar	162
12.4 Ứng dụng ScreenSaver	167
12.4.1 Đặc điểm	168
12.4.2 Tham số dòng lệnh	168
12.4.3 Đặc điểm giao tác với người dùng	170
12.4.4 Thực hiện ứng dụng ScreenSaver đơn giản	171
12.5 Ứng dụng sử dụng nhiều tiểu trình	175
12.5.1 Tiểu trình xử lý nội	175

12.5.2	Tiểu trình giao diện	177
12.5.3	Các hàm hỗ trợ	179
12.6	Lập trình Multimedia với MCI	179
12.7	Án định một số tính năng của Windows	181
12.8	Bẫy (hook) message (Windows Hook)	182
12.8.1	Các kiểu hook (Hook Type)	183
12.8.2	Danh sách hook (Hook Chain)	183
12.8.3	Thủ tục hook (Hook Procedure)	183
12.8.4	Các dịch vụ liên quan hook	184
12.8.5	Ứng dụng hook messages của keyboard	185
12.9	Cài đặt chế độ thực hiện ứng dụng tự động	186
CHƯƠNG 13	MFC VỚI INTERNET	187
13.1	Giao thức truyền thông TCP/IP	187
13.1.1	Giới thiệu	187
13.1.2	Kiến trúc của giao thức TCP/IP trên mô hình DARPA	187
13.1.3	Địa chỉ IP	189
13.1.4	Subnet	190
13.1.5	Subnet Mask	191
13.1.6	Host domain name	192
13.1.7	IP Routing	194
13.2	Lập trình TCP/IP với Winsock	197
13.2.1	Port	197
13.2.2	Socket	198
13.2.3	Một số cấu trúc dữ liệu của Winsock API	198
13.2.4	Một số dịch vụ của Winsock API	199
13.3	MFC với lập trình Winsock	200
13.3.1	Khởi động Winsock	200
13.3.2	Lớp CAsyncSocket	200
13.4	Lập trình Winsock cho giao thức UDP	204
13.5	Lập trình Winsock cho giao thức TCP	207
13.6	TCP với SMTP (Simple Mail Transfer Protocol)	215
13.6.1	Qui ước giữa ứng dụng gửi mail và nhận mail	215
13.6.2	Thiết kế ứng dụng gửi mail	217
13.7	TCP với Pop3 (Post Office Protocol – Version 3)	219
13.7.1	Qui ước giữa ứng dụng mail client và mail server	219

13.7.2	Thiết kế ứng dụng nhận mail	220
13.8	TCP với HTTP và FTP	223
13.8.1	Lớp CInternetSession	223
13.8.2	Lớp CInternetFile	224
13.8.3	Lớp CFtpConnection	225
13.8.4	Lớp CFtpFindFile	227
13.8.5	Lớp CHttpConnection	228
13.8.6	Lớp CHttpFile	229
13.8.7	Thực hiện ứng dụng FTP client đơn giản	231
13.8.8	Thực hiện ứng dụng HTTP client đơn giản	232

<u>Phụ lục A:</u>	MỘT SỐ VẤN ĐỀ	
	LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG	234
A.1	Lập trình hướng đối tượng (OOP)	234
A.2	Các khái niệm	234
A.2.1	Lớp (Class)	234
A.2.2	Đối tượng (Object)	234
A.2.3	Thuộc tính (Attribute)	234
A.2.4	Hành vi (Method)	235
A.2.5	Chương trình (Program)	235
A.3	Đặc điểm lập trình hướng đối tượng	235
A.4	Phân loại thuộc tính và hành vi	236
A.5	Các hành vi đặc biệt	236
A.6	Khai báo lớp, đối tượng trong C++	236
A.6.1	Khai báo lớp	236
A.6.2	Khai báo đối tượng	238
A.6.3	Sử dụng đối tượng trong chương trình	238
A.7	Kế thừa trong C++	239
A.7.1	Kế thừa hành vi tạo lập	239
A.7.2	Kế thừa hành vi hủy bỏ	240
A.7.3	Thực hiện hành vi lớp cơ sở	240
A.8	Khai báo hành vi toán tử số học	241
A.9	Con trỏ this	243
A.10	Hành vi virtual	243
A.11	Thuộc tính và hành vi tĩnh	244

LỜI MỞ ĐẦU



Ngôn ngữ lập trình C++ được biết đến như là một trong những ngôn ngữ lập trình mạnh nhất nhờ khả năng của nó trong việc triển khai phần mềm ở các mức độ khác nhau. Từ mức hệ thống đến mức ứng dụng, từ lập trình cấu trúc đến lập trình hướng đối tượng, từ lập trình dựa trên thuật giải đến lập trình trí tuệ nhân tạo, và từ lập trình cơ sở dữ liệu đến lập trình cơ sở tri thức..., bất cứ đâu, khi mà người lập trình muốn thể hiện ý tưởng khoa học và nghệ thuật của mình trên máy tính thì C++ là một điều nghĩ đến trước tiên.

Nhưng dù ý tưởng có bay bổng thế nào đi nữa thì cũng không thể bỏ qua vấn đề cài đặt mà môi trường cho ứng dụng là điều phải quan tâm. Với xu hướng sử dụng hệ điều hành Microsoft Windows như hiện nay, chúng ta buộc phải nghĩ đến việc cài đặt ứng dụng của mình trong môi trường này và khai thác nó sao cho ứng dụng hoạt động hiệu quả nhất.

Microsoft Visual C++, sản phẩm của Microsoft, với khả năng biên dịch ưu việt và lối khai thác hệ thống rộng mở nhờ tập hợp lớp thư viện MFC cho C++ có đầy đủ các tiện ích giúp chúng ta vét được mọi ngõ ngách của Windows hầu phục vụ cho ứng dụng của mình.

Từ những nhận định nói trên, cuốn sách này được thực hiện để cùng các bạn bắt đầu làm quen lập trình trong Windows áp dụng kỹ thuật lập trình hướng đối tượng với C++, nhằm khai thác hiệu quả thư viện MFC và từng bước du nhập vào thế giới tuyệt vời này thông qua các ứng dụng được sắp xếp theo các cấp độ tiến triển phù hợp.

Trong lần xuất bản đầu tiên, cuốn sách này chắc không tránh khỏi thiếu sót. Chúng tôi rất mong tiếp thu ý kiến đóng góp và trao đổi cùng bạn đọc. Cuối cùng, chúng tôi xin chân thành cảm ơn bạn bè, đồng nghiệp đã cung cấp những nhận xét và kiến thức quý báu để thực hiện cuốn sách này. Xin cảm ơn các bạn học viên-sinh viên, những người đã cùng làm việc với chúng tôi qua nội dung này và đã có những ý kiến khách quan giúp chỉnh sửa cuốn sách kịp thời.

Thành phố Hồ Chí Minh, ngày 19.11.2002

Tác giả

TÀI LIỆU THAM KHẢO



- [1] **Richard Simon**, *Windows 95 - Win32 Programming API-BIBLE*, Waite Group Press 1996.
- [2] **Jeff Prosise**, *Programming Windows 95 with MFC*, Microsoft Press.
- [3] **M. Tracy**, *Professional Visual C++ ISAPI Programming*, Wrox Press.
- [4] **Dr. GUI**, *Microsoft Developer Network - MSDN*, Microsoft Corporation Software.
- [5] **Dino Esposito**, *Visual C++ Windows Shell Programming*, Wrox Press.

PHẦN MỀM CẦN CÀI ĐẶT:

- Microsoft Visual C++ 6.0 hoặc Microsoft Visual C++ .NET.
- MSDN (Microsoft Developer Network), bản tháng 10/2003.

MÃ NGUỒN:

Source Code của các ví dụ minh họa trong cuốn sách này và của một số chương trình trò chơi mà chúng tôi mong muốn chia sẻ cùng bạn đọc được lưu trong đĩa mềm đính kèm, và có thể download từ địa chỉ:

<http://thanh.andisw.com/?id=16&id2=85>

WEB SITE:

Source Code đặc sắc của nhiều tác giả trên thế giới có thể download:

- <http://msdn.microsoft.com>
- <http://www.codeguru.com>
- <http://www.codeproject.com>
- <http://www.softtechsoftware.it>
- <http://www.flipcode.com>
- <http://nps.vnet.ee>

LIÊN HỆ:

- Tác giả: Lê Ngọc Thạnh
- Cơ quan: Khoa Tin Học Quản Lý, Trường ĐHKT TP.HCM
Địa chỉ: 279 Nguyễn Tri Phương Q10, TP.HCM.
- Địa chỉ e-mail: lnmail@yahoo.com
emp@ueh.edu.vn

CHƯƠNG 1:

Một số khái niệm Lập trình Trong môi trường Windows

1.1 CHƯƠNG TRÌNH (PROGRAM):

Chương trình máy tính là tập hợp các chỉ thị điều khiển hoạt động của máy, được bố trí theo một trình tự logic nhằm phối hợp thực hiện một công việc xác định. Các chỉ thị được thể hiện dưới dạng mã nguồn (source code) hay mã máy (machine code). Chương trình mã máy có thể thực hiện được trên máy có bộ lệnh tương thích, với chương trình mã nguồn thì phải sử dụng một ứng dụng chuyên dụng để chuyển sang mã máy trước khi thực hiện.

Việc chuyển các chỉ thị dạng mã nguồn sang chỉ thị mã máy để thực hiện được tiến hành bằng một trong hai cơ chế sau:

- **Thông dịch:** Mỗi chỉ thị mã nguồn được chuyển sang chỉ thị mã máy tương ứng và được thực hiện ngay, sau đó tiếp tục với chỉ thị kế tiếp.
- **Biên dịch:** Tất cả các chỉ thị mã nguồn được chuyển sang các chỉ thị mã máy tương ứng. Tập hợp các chỉ thị mã máy này gọi là chương trình mã máy. Chương trình mã máy được lưu lại trong tập tin chương trình và về sau ta có thể thực hiện chúng một cách độc lập trên máy.

1.2 ỨNG DỤNG (APPLICATION):

Khi một chương trình được cài đặt trên máy tính để sử dụng, ta gọi đó là ứng dụng, ví dụ như ứng dụng NotePad, ứng dụng Microsoft Word,....

Trong môi trường windows, mỗi ứng dụng có thể được thi hành nhiều lần thành nhiều bản khác nhau. Mỗi bản đang thực hiện của một ứng dụng gọi là thể hiện (instance) của ứng dụng đó.

1.3 TIẾN TRÌNH (PROCESS):

Tiến trình là khái niệm chỉ một instance đang hoạt động của ứng dụng. Khi ta double-click trên biểu tượng NotePad để chạy ứng dụng này, ta có một tiến trình của ứng dụng NotePad.

1.4 TIỂU TRÌNH (THREAD):

Tiểu trình là một nhánh xử lý độc lập trong tiến trình. Khi một ứng dụng được thực hiện ta có thêm một tiến trình. Do bản chất chương trình làm nên ứng dụng đó bao gồm chương trình chính (**main** hay **WinMain**) và các chương trình con mà tiến trình ứng với nó có thể tách thành các nhánh xử lý: một nhánh xử lý chính (primary thread), các nhánh xử lý phụ (other threads). Các nhánh xử lý này gọi là các tiểu trình. Có hai loại tiểu trình:

- **Tiểu trình giao diện (user-interface thread):** Có nhiệm vụ xử lý các yêu cầu của người dùng trong quá trình giao tác với họ.
 - **Tiểu trình xử lý nội (worker thread):** Có nhiệm vụ thực hiện các xử lý tính toán bên trong, không trực tiếp nhận yêu cầu của người dùng.
- ☞ Thực ra, có thể xem tiểu trình giao diện như là một tiểu trình xử lý nội nhưng có tính năng giao tác với người sử dụng.

1.5 THÔNG điệp (MESSAGE):

Thông điệp (message) là giá trị phản ánh một nội dung giao tiếp hay yêu cầu xử lý giữa hệ thống (windows) và ứng dụng, giữa các ứng dụng với nhau hoặc giữa các thành phần trong cùng một ứng dụng.

1.5.1 Nguồn gốc message:

Cả windows và ứng dụng đều có thể phát sinh message.

- Windows phát sinh message khi cần thông tin cho ứng dụng các hoạt động nhập-xuất (hoạt động gõ phím, di chuyển hay click chuột, của người dùng), các thay đổi của hệ thống (font chữ, chế độ phân giải màn hình, màu sắc,...) hoặc những biến đổi khác liên quan đến ứng dụng.
- Ứng dụng phát sinh message khi xử lý điều khiển các thành phần bên trong ứng dụng phối hợp thực hiện chức năng giao tiếp với người dùng, hoặc khi ứng dụng thực hiện giao tiếp với windows hay với các ứng dụng khác đang thực hiện trong cùng hệ thống.

1.5.2 Các loại message:

- **Message được định nghĩa bởi hệ thống:** Là các message do hệ điều hành windows tạo ra nhằm phục vụ hoạt động điều khiển toàn bộ hệ thống, xử lý thông tin vào-ra hoặc các thông tin khác cho ứng dụng. Khi có nhu cầu, ứng dụng có thể sử dụng những message này để phát động một chức năng điều khiển nào đó của windows.
- **Message được định nghĩa bởi người dùng:** Là các message do người viết ứng dụng định nghĩa nhằm tạo kênh liên lạc đặc thù giữa các thành phần trong ứng dụng, giữa ứng dụng với windows hoặc với các ứng dụng khác đang thực hiện trong cùng hệ thống.

1.5.3 Số hiệu message (Message Identifier - MessageID):

Có rất nhiều message khác nhau được sử dụng trong môi trường windows. Ứng với mỗi message xác định, windows sử dụng một giá trị nguyên không âm để đặc tả, giá trị này gọi là số hiệu message.

Các message do windows định nghĩa có số hiệu được khai báo sẵn và duy nhất với các hằng số xác định và tên gọi gợi nhớ của chúng có dạng WM_XXX. Các messages do người dùng định nghĩa cũng phải đăng ký số hiệu. Số hiệu đăng ký không được trùng lặp và có giá trị nhỏ nhất bằng WM_USER (một hằng số do windows định nghĩa).

Số hiệu message là cơ sở để phân biệt các message lẫn nhau.

1.5.4 Đặc tả message:

Để đối tượng nhận message có thêm thông tin về hoàn cảnh phát sinh và ý nghĩa cụ thể của message, windows cho phép message được nhận thông qua một cấu trúc chứa số hiệu message và các thông số kèm theo. Cấu trúc này được khai báo thành kiểu MSG với nội dung như sau:

```
typedef struct tagMSG {
    HWND    hwnd;    // Giá trị có kích thước 4 bytes (long)
    UINT    message;  // Số hiệu của message
    WPARAM  wParam;   // Giá trị không âm có kích thước 2 bytes
    LPARAM  lParam;   // Giá trị không âm có kích thước 4 bytes
    DWORD   time;     // Thời điểm sinh ra message
    POINT   pt;       // Tọa độ cursor khi message được gửi.
} MSG;
```

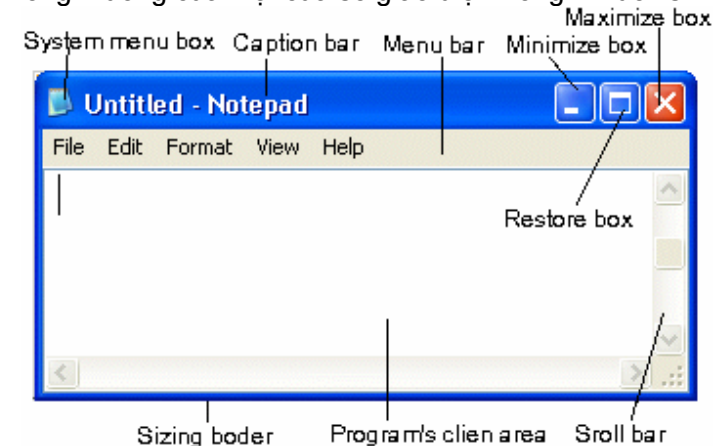
Trường *hwnd* (window handle) của cấu trúc chứa thẻ (handle) quản lý cửa sổ giao diện liên quan đến message. *wParam* và *lParam* là hai tham số gửi kèm theo message làm nhiệm vụ chứa thông tin bổ sung. Hai tham số này được dùng khi message có nhiều ý nghĩa thực tế khác nhau. Windows sử dụng giá trị có kích thước 4 bytes để quản lý các đối tượng của nó. Giá trị này gọi là thẻ quản lý đối tượng (object handle). Ứng với mỗi loại đối tượng cụ thể, windows sử dụng một kiểu handle riêng. HWND là kiểu handle dùng quản lý đối tượng cửa sổ giao diện trong windows.

1.6 CỬA SỔ GIAO DIỆN (WINDOW) CỦA ỨNG DỤNG:

Cửa sổ giao diện là thành phần quan trọng trong việc tạo ra môi trường giao diện đồ họa của các ứng dụng trong windows. Cùng với sự phát triển của hệ điều hành windows, hình ảnh cửa sổ giao diện cũng thay đổi theo với chiều hướng sinh động hơn về hình thức và phong phú hơn về chức năng. Điều đó không chỉ góp phần tăng tính thẩm mỹ mà

còn tạo sự gắn gũi hơn giữa giao diện của ứng dụng trong windows với người dùng.

Dạng thông thường của một cửa sổ giao diện trong windows:



- **System Menu Box:** Chứa biểu tượng của ứng dụng, là nút mở hộp menu hệ thống với các mục di chuyển, thay đổi kích thước hoặc đóng cửa sổ.
- **Caption bar:** Thanh tiêu đề của ứng dụng.
- **Menu bar:** Hệ thống menu với các mục lựa chọn xử lý.
- **Minimize / Maximize Box:** Nút điều khiển thu nhỏ / phóng to cửa sổ.
- **Restore Box:** Nút khôi phục kích thước trước đó của cửa sổ.
- **Border:** Đường viền bao quanh cửa sổ.
- **Client area:** Vùng làm việc của cửa sổ, dùng để hiển thị thông tin.
- **Scroll bar:** Thanh cuộn nội dung vùng làm việc của cửa sổ.
- **Window Procedure:** Ngoài giao diện đồ họa, cửa sổ của windows có khả năng tiếp nhận và xử lý message. Khả năng này được thực hiện thông qua hàm xử lý message mà ta đã gán cho cửa sổ. Hàm xử lý này có khai báo như sau:

```
LRESULT CALLBACK WindowProc (
    HWND hwnd,          // Tham số chứa Handle của cửa sổ liên
    quan
    UINT uMsg,           // Tham số chứa số hiệu message
    WPARAM wParam,       // Tham số bổ sung thứ nhất kiểu WORD
    LPARAM lParam        // Tham số bổ sung thứ hai kiểu LONG
); // Hàm trả về một giá trị có kích thước là 32 bits.
```

Khi một yêu cầu xử lý được chuyển đến cửa sổ dưới dạng message, hàm *WindowProc* gán với cửa sổ sẽ căn cứ trên số hiệu message (*uMsg*) để chọn xử lý phù hợp. Theo nguyên tắc, nếu message được xử lý hoàn tất thì hàm trả về giá trị 0, ngược lại (message không thuộc

khả năng xử lý của cửa sổ) hàm phải chuyển message cho windows xử lý thông qua lời gọi hàm xử lý message mặc nhiên của windows. Hàm xử lý này có tên gọi là *DefWindowProc* với khai báo như sau:

```
LRESULT DefWindowProc( // Default Window Procedure
    HWND hWnd, // Với các tham số có ý nghĩa như trên
    UINT Msg,
    WPARAM wParam,
    LPARAM lParam
);
```

Khi đó, kết quả trả về của *DefWindowProc* được dùng làm kết quả của hàm *WindowProc*. Qui tắc nói trên cần phải được đảm bảo, nếu không, ứng dụng có thể làm rối loạn hoạt động của hệ điều hành windows.

1.7 MESSAGE QUEUE:

Message queue là danh sách thứ tự các message do windows tạo ra và được dùng để chứa các message đang chờ được xử lý. Message queue hoạt động theo nguyên tắc FIFO (First-In, First-Out: vào trước, ra trước). Có hai loại message queue trong windows:

- Message queue của hệ thống (system queue): Được windows dùng riêng cho việc lưu trữ các message đặc tả thông tin nhập-xuất liên quan đến thiết bị phần cứng trong quá trình hệ thống giao tác với người dùng.
- Message queue của ứng dụng (application queue): Được windows tạo ra và cấp cho các thể hiện của ứng dụng. Windows tự động điều phối các message từ system queue sang application queue một cách phù hợp, nhờ đó mỗi ứng dụng có thể tiếp nhận và thực hiện các yêu cầu xử lý của người dùng thông qua hệ thống. Cơ chế này ngăn các ứng dụng trong windows quyền truy cập trực tiếp các thiết bị phần cứng của máy tính.

Việc truy cập message queue của ứng dụng được thực hiện với sự hỗ trợ của các hàm liên quan do windows cung cấp như sau:

- Chờ và lấy một message từ message queue của ứng dụng:

```
BOOL GetMessage (
    LPMSG lpMsg, // Con trỏ đến biến MSG nhận thông tin
    HWND hWnd, // Handle của cửa sổ liên quan
    UINT wMsgFilterMin, // Số hiệu message nhỏ nhất nhận được
    UINT wMsgFilterMax // Số hiệu message lớn nhất nhận được
);
```

Hàm tự động chờ đến khi phát hiện có message cần xử lý trong message queue. Khi đó, message vào trước nhất sẽ được lấy ra khỏi hàng chờ và thông tin của nó được điền vào biến kiểu MSG chỉ bởi

con trỏ tham số *lpMSG*. Khi đã lấy được một message, hàm kết thúc và trả về một giá trị nguyên. Nếu message nhận được là message kết thúc ứng dụng (số hiệu WM_QUIT) thì hàm trả về giá trị 0. Ngược lại, hàm trả về giá trị khác 0.

- Kiểm tra và lấy một message trong message queue của ứng dụng:

```
BOOL PeekMessage(
    LPMSG lpMsg, // . Như GetMessage
    HWND hWnd,
    UINT wMsgFilterMin,
    UINT wMsgFilterMax,
    UINT wRemoveMsg, // Có thực hiện xóa message không ?
                    // PM_NOREMOVE: không xóa
);
```

Hàm trả về giá trị 0 nếu message queue rỗng. Ngược lại, hàm trả về một giá trị khác không và thông tin về message được điền vào biến kiểu MSG được chỉ bởi tham số kiểu con trỏ *lpMSG*.

- Diễn dịch message của bàn phím sang mã phím ASCII:

```
BOOL TranslateMessage(
    CONST MSG *lpMsg, // con trỏ đến biến chứa message
);
```

Tham số *lpMsg* là con trỏ chỉ đến biến kiểu MSG chứa nội dung đặc tả message được lấy từ message queue và cần diễn dịch.

Hàm trả về giá trị khác 0 nếu message nhận được tương ứng với một thao tác trên bàn phím (nhấn phím: WM_KEYDOWN, WM_SYSKEYDOWN; thả phím: WM_KEYUP, WM_SYSKEYUP) hoặc một message có ý nghĩa tương đương mà việc diễn dịch sang mã phím ASCII là thành công. Khi đó hàm tự động tạo message WM_CHAR cho phím diễn dịch được. Trong các trường hợp khác, hàm trả về giá trị 0.

▪ Điều phối message đến cửa sổ giao diện chính:

Cửa sổ giao diện chính của ứng dụng có thể tiếp nhận và xử lý message thông qua hàm **WindowProc** của nó (1.6). Như vậy, ta có thể điều phối message lấy từ message queue của ứng dụng đến cho cửa sổ chính xử lý. Việc điều phối được thực hiện thông qua hàm sau:

```
LRESULT DispatchMessage(
    CONST MSG *lpmsg,    // Con trỏ đến biến chứa message
);
```

Tham số *lpMSG* chỉ đến biến kiểu MSG chứa thông tin đặc tả message được điều phối. Hàm điều phối sẽ chờ đến khi hàm xử lý message WindowProc của cửa sổ chính xử lý xong message, và lấy giá trị kết thúc của hàm này làm giá trị trả về của chính nó.

☞ Quá trình tiếp nhận và điều phối xử lý message từ message queue của ứng dụng được tiến hành liên tục cho đến khi nhận được message kết thúc ứng dụng (WM_QUIT). Quá trình này có tên gọi là vòng lặp nhận và điều phối message (MessageLoop). Đoạn chương trình nhận và điều phối message trong ứng dụng được cài đặt như sau:

```
MSG msg;                // biến chứa nội dung đặc tả message nhận được
while( GetMessage( &msg, NULL, 0, 0 ) != 0 ) {
    // Nếu message nhận được không phải là WM_QUIT
    TranslateMessage(&msg);    // Dịch dịch nếu là phím
    DispatchMessage(&msg);    // Điều phối cho cửa sổ chính.
}
```

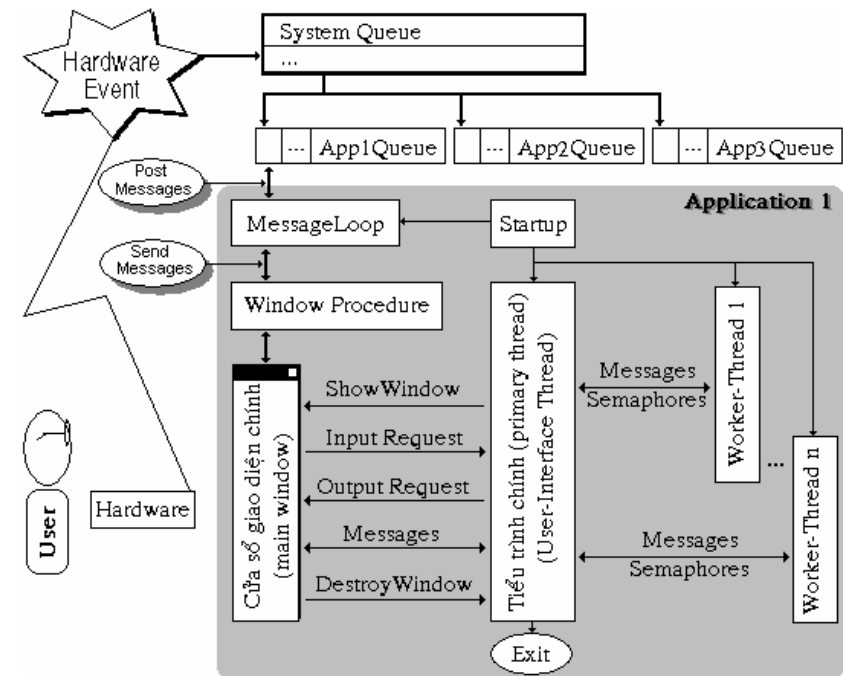
1.8 KIẾN TRÚC XỬ LÝ CỦA ỨNG DỤNG TRONG WINDOWS:

Khi ứng dụng được thực hiện, windows tạo thêm một tiến trình cho thể hiện mới của ứng dụng, đồng thời xây dựng một message queue dùng riêng cho thể hiện này. Tiểu trình chính của tiến trình được tạo ra có nhiệm vụ thực hiện MessageLoop trên message queue dành cho ứng dụng, đồng thời khởi tạo các giao diện và tiểu trình hỗ trợ (nếu cần).

☞ Các cách xử lý của tiểu trình chính khi nắm quyền điều khiển ứng dụng:

- Không thực hiện xử lý nào cả: Ứng dụng kết thúc.
- Thực hiện MessageLoop nhưng không tạo cửa sổ giao diện chính: Ứng dụng chờ nhận message nhưng người dùng không giao tác được.
- Khởi tạo một cửa sổ với hàm xử lý message WindowProc và chọn cửa sổ này làm cửa sổ giao diện chính: Hàm WindowProc của cửa sổ sẽ tiếp nhận và xử lý message được điều phối từ MessageLoop của tiểu trình chính. Người dùng có thể giao tác và kết thúc ứng dụng được.
- Như mục thứ ba, đồng thời tạo ra các tiểu trình phục vụ: Ứng dụng có khả năng tiếp nhận và thực hiện đồng thời nhiều yêu cầu xử lý.

Kiến trúc xử lý chung của ứng dụng trong môi trường windows



1.9 RESOURCE CỦA ỨNG DỤNG:

Đối với một chương trình trong windows, ngoài phần mã lệnh của các hàm xử lý, resource là một thành phần không kém phần quan trọng chứa các nội dung hỗ trợ cho việc trang trí hoặc phục vụ cho một mục đích đặc biệt của ứng dụng. Các nội dung phổ biến trong resource như sau:

- **Cursor:** Ảnh nhỏ đặc tả vị trí làm việc của thiết bị liên quan như mouse, pen, trackball. Khi người dùng tác động lên những thiết bị này thì windows sử dụng cursor để phản ánh hiện tượng đó.
- **Bitmap:** Tập ảnh điểm (pixels) của một ảnh. Các ảnh điểm này bố trí theo các dòng và phối hợp làm nên hình ảnh của đối tượng.
- **Dialog:** Thông tin mô tả khung giao diện với các đối tượng nhập liệu bên trong, là cơ sở để tạo ra các hộp hội thoại trong ứng dụng.
- **Icon:** Ảnh nhỏ được dùng để đặc tả chức năng của một đối tượng, ứng dụng hay một nội dung dữ liệu.
- **HTML (Hypertext Markup Language):** Ngôn ngữ dùng tạo ra những tài liệu dạng văn bản với những ký pháp và kỹ thuật định dạng mà trình duyệt tương ứng có thể thể hiện một cách xúc tích.

- **Menu** : Một danh sách các lựa chọn xử lý mà người dùng có thể chọn nhằm thực hiện một xử lý xác định
- **String Table**: Bảng chứa các chuỗi được đánh dấu phân biệt bởi các số hiệu và được sử dụng như các thông báo trong chương trình. Việc sử dụng *String Table* giúp ứng dụng dễ dàng thay đổi ngôn ngữ giao diện của nó mà không cần phải có sự chỉnh sửa trên phần mã lệnh.
- **ToolbarBitmap**: Tập các ảnh con xác định các nút được cài đặt trên thanh công cụ của cửa sổ hay hộp hội thoại trong ứng dụng. Mỗi nút này là một mục chọn (có thể thay thế mục chọn của menu) giúp tạo ra các message lệnh (WM_COMMAND) với số hiệu phân biệt để có thể ấn định xử lý cần thiết.
- **Version**: Phần khai báo các thông tin liên quan đến ứng dụng, tác giả.
- **Font**: Chứa thông tin về bộ font chữ được lưu trong tập tin **fnt**.
- **Custom Resource**: Bao gồm các nội dung không thuộc các loại nội dung resource chuẩn nói trên. Người dùng có thể tùy ý cài vào resource của ứng dụng, đồng thời phải tự cài đặt xử lý thích hợp cho các resource này trong chương trình.

CHƯƠNG 2:

Thư viện MFC của microsoft & ứng dụng cơ bản trong windows

2.1 THƯ VIỆN MFC (MICROSOFT FOUNDATION CLASS):

Thư viện MFC của Microsoft bao gồm các lớp cơ bản, cài đặt bằng ngôn ngữ C++, hỗ trợ việc lập trình trong môi trường windows. Từ các lớp này, MFC xác lập nền tảng hình thành ứng dụng của windows, bao gồm việc định nghĩa bộ khung ứng dụng, các công cụ chuẩn và phổ biến để bổ sung vào bộ khung nói trên nhằm tạo ra ứng dụng hoàn chỉnh. Với MFC, công việc của người lập trình chỉ còn là việc lựa chọn các thành phần cần thiết, điều chỉnh và phối hợp chúng hợp lý để có được ứng dụng kết quả mong muốn.

Lập trình windows với MFC và MicroSoft Visual C++ 6.0 (VC) đạt được hiệu quả cao bởi không chỉ khai thác được phiên bản mới nhất của MFC mà còn nhận được nhiều tiện nghi lập trình mà VC cung cấp. Đây là con đường ngắn và đơn giản, đặc biệt với người tự học, để viết ứng dụng windows.

2.2 TIẾP CẬN MFC:

MFC là thư viện khổng lồ với khoảng 200 lớp đối tượng mà việc hiểu rõ và vận dụng chúng trong một khoảng thời gian ngắn là không thể được. Mục tiêu của chúng ta là hiểu và vận dụng những thành phần phổ biến nhất của thư viện để xây dựng ứng dụng thông thường. Khi đã có khả năng nhất định về sử dụng MFC thì với tài liệu MSDN, sẽ chẳng khó khăn gì trong việc mở rộng khai thác thư viện để ứng dụng trở nên mạnh mẽ và tinh tế hơn.

Trong những phần trình bày sau, chúng ta sẽ lần lượt tiếp nhận hệ thống nội dung hơi nặng tính lý thuyết để đảm bảo cấu trúc kiến thức, và phần thực hành phối hợp sẽ giúp chúng ta kiểm nghiệm và hiểu rõ vấn đề.

2.3 TẠO ỨNG DỤNG WINDOWS VỚI MFC NHƯ THẾ NÀO ?

Theo mô hình kiến trúc ứng dụng windows ở mục (1.8), việc giải quyết vấn đề trên chính là việc thực hiện trả lời các câu hỏi sau đây:

- Làm thế nào tạo đối tượng tiểu trình chính của ứng dụng ?
 - Làm thế nào tạo đối tượng giao diện của ứng dụng ?
 - Quản lý tương tác giữa đối tượng ứng dụng và đối tượng giao diện ?
- Bằng việc xem xét các lớp MFC liên quan sẽ giúp lần lượt lý giải các câu hỏi được đặt ra. Tiếp theo, chúng ta tìm hiểu xem những lớp nào của MFC giúp khai báo đối tượng tiểu trình trong ứng dụng.

2.4 LỚP QUẢN LÝ TIỂU TRÌNH CWinThread:

CWinThread là một lớp của MFC, lớp đối tượng quản lý tiểu trình được tạo ra trong tiến trình của một ứng dụng đang được thực hiện. Tiểu trình được quản lý có thể là tiểu trình giao diện hoặc tiểu trình xử lý nội. Các dịch vụ cơ bản phục vụ cho quản lý tiểu trình do CWinThread cung cấp thông qua các thuộc tính và hành vi của nó.

- DWORD *m_nThreadID*: Thuộc tính lưu số hiệu của tiểu trình.
- CWnd* *m_pMainWnd*: Lưu con trỏ đối tượng cửa sổ giao diện chính của tiểu trình. Khi cửa sổ giao diện chính chấm dứt hoạt động, tiểu trình liên quan sẽ kết thúc. Nếu tiểu trình thuộc loại tiểu trình xử lý nội thì giá trị này kế thừa từ tiểu trình giao diện cấp cao hơn.
- CWinThread(); Hành vi tạo lập (constructor) đối tượng tiểu trình.
- virtual BOOL *InitInstance*(); Khởi tạo thông số cho đối tượng tiểu trình và đảm nhận các xử lý bổ sung khác của ứng dụng. Đối với tiểu trình giao diện, hành vi này được dùng để khởi tạo đối tượng cửa sổ giao diện và gán địa chỉ của đối tượng cửa sổ cho *m_pMainWnd*.
- virtual int *ExitInstance*(); Hành vi kết thúc của đối tượng tiểu trình. Thông qua hành vi này, đối tượng quản lý tiểu trình thực hiện hoàn trả các tài nguyên của hệ thống mà nó đã đăng ký sử dụng.
- virtual int *Run*(); Hành vi dành riêng cho tiểu trình giao diện, nó thực hiện vòng lặp nhận message, chuyển message cho hành vi *PreTranslateMessage* của lớp. Nếu hành vi này trả về giá trị 0 thì message sẽ tiếp tục được chuyển đến các hàm diễn dịch phím *TranslateMessage* và hàm điều phối message *DispatchMessage*.
- virtual BOOL *PreTranslateMessage*(MSG *pMsg); Hành vi cho phép can thiệp trước trên các message nhận được từ message queue của ứng dụng. Thông qua đó, tiểu trình giao diện có thể thực hiện các tiền xử lý message đặc trưng nhằm đáp ứng yêu cầu đặt ra cho ứng dụng.

☞ Khi CWinThread được dùng để quản lý đối tượng tiểu trình chính thì hàm *WinMain* (cài sẵn bên trong lớp bởi thư viện MFC) tự động thực hiện các hành vi *InitInstance()*, *Run()* và *ExitInstance()* theo thứ tự.

2.5 LỚP QUẢN LÝ TIỂU TRÌNH GIAO DIỆN CHÍNH CWinApp:

CWinApp, kế thừa từ CWinThread, là lớp đối tượng chuyên dùng quản lý tiểu trình giao diện chính của ứng dụng. Ứng dụng windows chỉ được phép sử dụng một đối tượng thuộc lớp này. Ngoài các thuộc tính, hành vi kế thừa public từ CWinThread, CWinApp có các thuộc tính và hành vi bổ sung sau:

- `const char* m_pszAppName`: Lưu chuỗi tên của ứng dụng.
- `LPTSTR m_lpCmdLine`: Lưu nội dung chuỗi tham số dòng lệnh. Tham số dòng lệnh là toàn bộ phần nội dung mà người dùng gõ vào ngay sau chuỗi đường dẫn và tên chương trình ứng dụng khi họ thực hiện ứng dụng. Đối với ứng dụng có nhiều chế độ hoạt động khác nhau thì việc sử dụng tham số dòng lệnh là một cơ chế xác lập các giao ước giữa ứng dụng và người dùng để chọn chế độ sử dụng ứng dụng thích hợp.
- `int m_nCmdShow`: Lưu giá trị thông số đã được dùng để kích hoạt cửa sổ giao diện chính của ứng dụng.
- `CWinApp()`: Tạo lập đối tượng tiểu trình chính.
- `HCURSOR LoadCursor(int nID)`: Nạp cursor từ resource của ứng dụng vào bộ nhớ (nếu cursor chưa được nạp). *nID* là số hiệu của cursor. Hàm trả về giá trị handle quản lý cursor. Giá trị này có kiểu `HCURSOR` (handle of cursor) mà windows dùng để quản lý cursor.
- `HICON LoadIcon (int nID)`: Nạp icon từ resource của ứng dụng vào bộ nhớ. *nID* là số hiệu của icon. Hàm trả về giá trị handle quản lý icon. Giá trị này có kiểu `HICON` (handle of icon).
- Hành vi hiển thị hộp thông báo và chờ nhận ý kiến người dùng:
`virtual int DoMessageBox (`
 `LPCTSTR lpszPrompt, // Nội dung thông báo`
 `UINT nType, // Dạng hộp thông báo`
 `UINT hlpIndex = 0 // Số hiệu mục giúp đỡ (WinHelp)`
`);`

Hành vi này được kích hoạt với tham số tương ứng mỗi khi trong chương trình sử dụng hàm `AfxMessageBox` để hiển thị hộp thông báo.

`int AfxMessageBox (`
 `LPCTSTR lpszPrompt, UINT nType, UINT hlpIndex = 0`
`);`





nType: Ấn định dạng hộp thông báo. Giá trị này là sự kết hợp giữa thông số qui định biểu tượng hiển thị và các nút chọn bố trí trong hộp.

- **Các thông số qui định các nút chọn bố trí trong hộp thông báo.**

Giá trị thông số	Nút chọn bố trí trong hộp
------------------	---------------------------

MB_OK	OK
MB_OKCANCEL	OK - Cancel
MB_YESNO	Yes - No
MB_YESNOCANCEL	Yes - No - Cancel
MB_ABORTRETRYIGNORE	Abort - Retry - Ignore
MB_RETRYCANCEL	Retry - Cancel


- **Các thông số qui định biểu tượng dùng trong nút chọn.**

Giá trị thông số	Biểu tượng
MB_ICONHAND, MB_ICONSTOP, MB_ICONERROR	
MB_ICONQUESTION	
MB_ICONEXCLAMATION, MB_ICONWARNING	
MB_ICONASTERISK, MB_ICONINFORMATION	

- **Số hiệu các nút chọn được sử dụng trong hộp thông báo.**

Số hiệu	Nút chọn	Số hiệu	Nút chọn
IDABORT	Abort	IDOK	OK
IDCANCEL	Cancel	IDRETRY	Retry
IDIGNORE	Ignore	IDYES	Yes
IDNO	No		

- Hành vi `DoMessageBox` chờ người sử dụng trả lời bằng cách chọn một nút chọn xác định trong hộp thông báo. Hành vi kết thúc với giá trị trả về là số hiệu của nút được chọn. Kế thừa hành vi này để chặn và thực hiện xử lý đặc trưng (sử dụng dạng hộp thông báo riêng) cho tất cả các lời gọi `AfxMessageBox` trong ứng dụng.
- Hàm `AfxMessageBox` sử dụng `DoMessageBox` để hiển thị hộp thông báo và lấy giá trị của hành vi này làm kết quả trả về của nó.

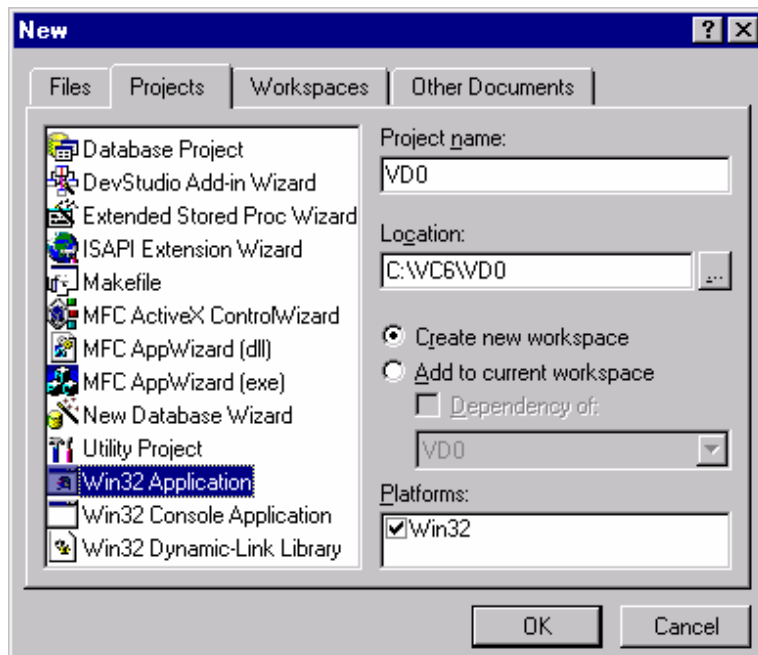
Ví dụ: `AfxMessageBox("Chao Ban", MB_ICONSTOP | MB_OK, 0);`
Hàm trên thực hiện hiển thị hộp thông báo với nội dung là "Chao Ban", biểu tượng  đi kèm, và nút chọn OK kết thúc.

2.6 THỰC HIỆN ỨNG DỤNG ĐƠN GIẢN:

Trong phần này, chúng ta thực hiện ứng dụng với đối tượng thuộc lớp CWinApp quản lý tiểu trình chính. Đặt tên cho dự án của ứng dụng là VD0.

Các bước thực hiện dự án VD0 như sau:

- Khởi động windows với hệ điều hành Win95 hoặc bản mới hơn.
- Tạo mới một thư mục để chứa các dự án. Ví dụ **C:\VC6**.
- Thực hiện ứng dụng Microsoft Visual C++ 6.0 (VC).
- Chọn mục **File / New** từ hệ thống menu của VC.
- Trong hộp hội thoại **New**, chọn trang **Projects**:



- **Win32 Application** : Loại ứng dụng thực hiện.
 - **Location** : Đường dẫn thư mục của dự án.
 - **Project Name** : Tên dự án.
- Sau đó chọn **OK**.

- Tiếp theo, trong hộp hội thoại **Step 1 of 1**.
 - **An empty project** : Tạo dự án rỗng.
 - **Finish** : Hoàn tất việc khởi tạo dự án.

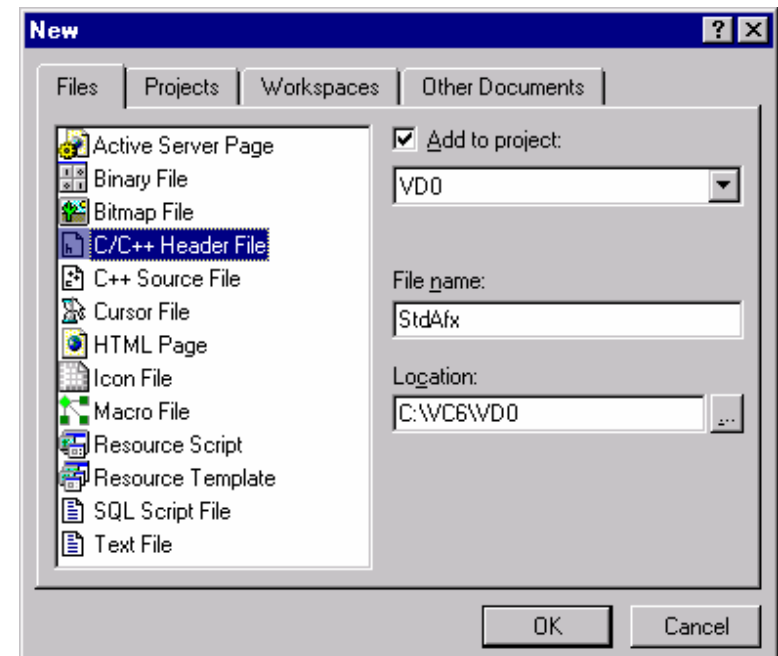
- VC hiển thị hộp hội thoại **New Project Information** để thông báo các thông tin liên quan đến dự án vừa tạo. Chọn **OK**.
 - ☞ Một dự án đã được tạo ra trên đĩa. Với thông tin nhập như trên, dự án mới tạo ra có tên là VD0, toàn bộ phần chương trình nguồn của dự án được lưu trong thư mục VD0 thuộc thư mục C:\VC6.

Tiếp tục thực hiện các bước sau để hoàn tất dự án theo yêu cầu.

- Đăng ký sử dụng lớp CWinApp của thư viện MFC: Lớp CWinApp được khai báo trong **afxwin.h** của MFC. Bổ sung tập tin **stdafx.h** vào dự án và dùng tập tin này đăng ký các thư viện cần thiết của MFC.

Việc bổ sung tập tin **stdafx.h** vào dự án được tiến hành như sau:

- Chọn mục **File / New** từ hệ thống menu của VC.
- Trong hộp hội thoại **New**, chọn trang **Files**:



- **C/C++ Header File** : Loại nội dung tập tin (.h).
- **Add To Project** : Bổ sung tập tin vào dự án VD0.
- **File Name** : Tên tập tin (**StdAfx.h**)

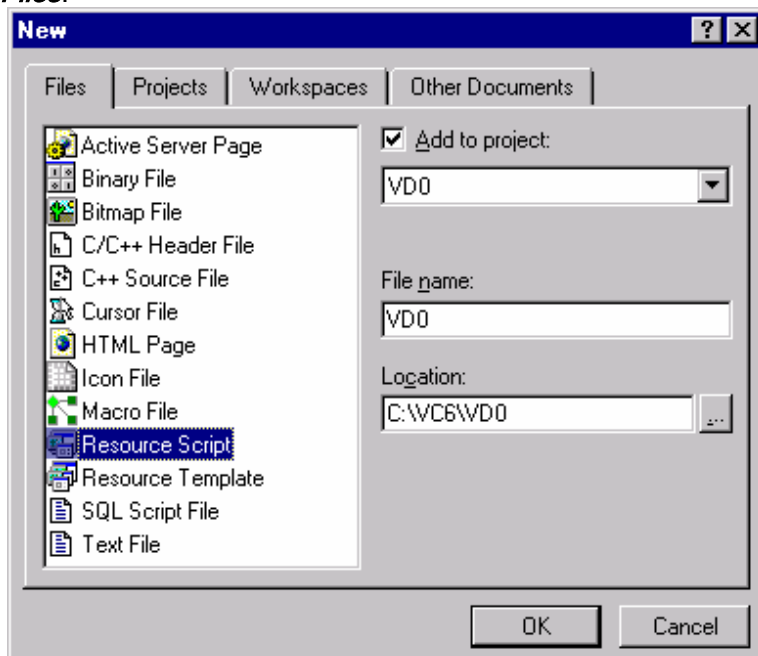
Chọn **OK**, tiếp tục thực hiện các công việc sau.

- Soạn thảo tập tin **stdafx.h**, nhập các định hướng biên dịch và các chỉ thị sử dụng thư viện MFC cần cho dự án:

```
#if !defined( _DU_AN_0_ )
#define _DU_AN_0_
// _DU_AN_0_ giúp trình dịch không thực hiện lặp chỉ thị
#include
#include <afxwin.h> // thư viện chuẩn của MFC
#endif
```

Sau khi nhập xong nội dung tập tin **stdafx.h**, chọn mục **File / Save** (hoặc click biểu tượng  trên thanh công cụ) để lưu tập tin.

- Tạo tập tin **Resource Script** của dự án: Tập tin này chứa khai báo của các resource được sử dụng trong ứng dụng. Khi biên dịch, các resource này sẽ được nhúng vào tập tin chương trình (.EXE). Các bước tạo tập tin **Resource Script** trong dự án như sau:
 - Chọn mục **File / New**. Trong hộp thoại **New**, chọn trang **Files**.



- Resource Script** : Loại nội dung tập tin (.rc).
- Add To Project** : Bổ sung tập tin vào dự án.
- File Name** : Tên tập tin, trùng với tên của dự án.

Sau đó chọn **OK**.

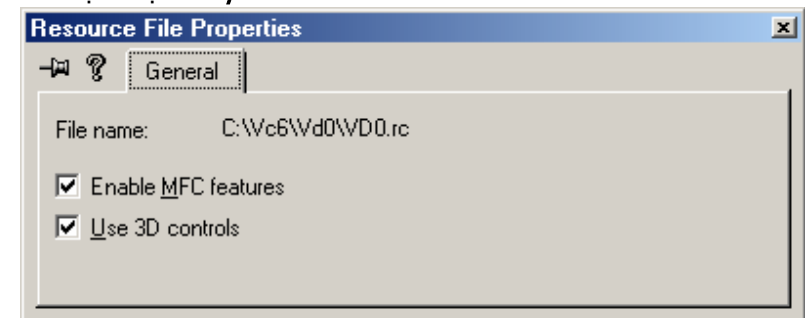
- Đăng ký các hỗ trợ của MFC cho thao tác trên resource: Việc đăng ký này là cơ sở tạo quan hệ giữa các nội dung của

resource và đối tượng lập trình tương ứng của MFC trong dự án. Thực hiện như sau:

- Right-click trên mục tên resource (**VD0 Resource**):



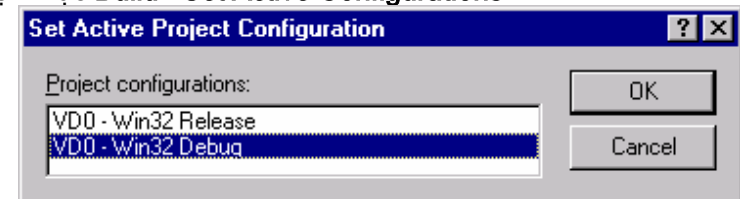
- Chọn mục **Properties**:



Chọn các mục như trên, gõ phím **Enter** để kết thúc.

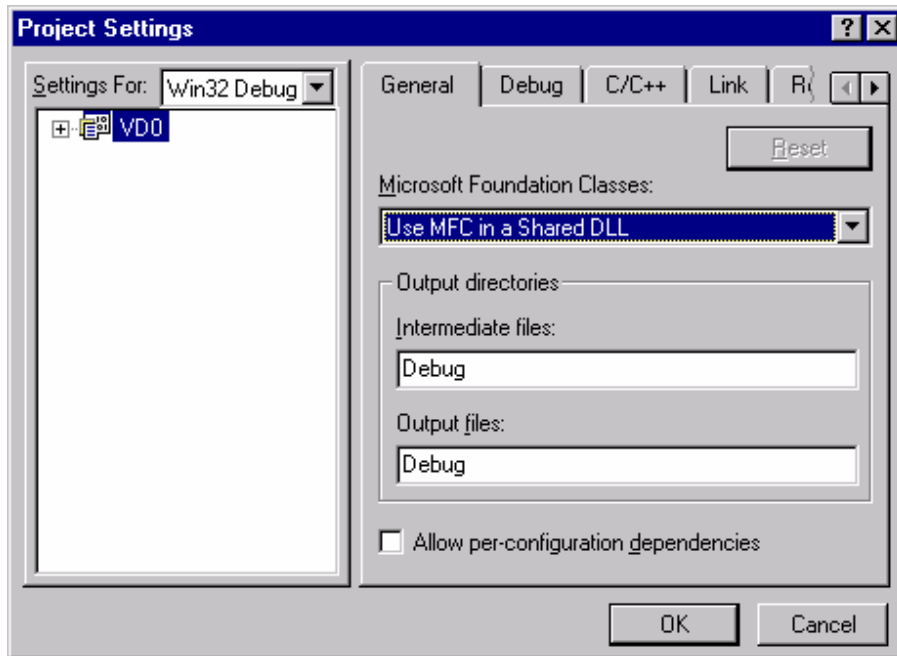
- Soạn thảo resource: Dự án **VD0** chưa cần sử dụng resource, tập tin resource script sẽ tạm thời bỏ trống. Chọn mục **File / Save**, và đóng màn hình soạn thảo resource để kết thúc bước này.
- Chọn phiên bản biên dịch: Có hai phiên bản biên dịch chương trình.
 - Debug version** : Biên dịch chương trình với thông tin debug.
 - Release version** : Phiên bản đem giao, không chứa thông tin debug.

Chọn mục **Build / Set Active Configurations**:



Chọn phiên bản biên dịch (chẳng hạn **Win32 Debug**). Chọn **OK**.

- Ấn định biên dịch với thư viện MFC: Chọn mục **Project / Setting**.



Trong hộp hội thoại **Project Settings**,

- **Setting For** : Chọn phiên bản ấn định (ví dụ: Win32 Debug),
- **General** : Các ấn định chung cho dự án.

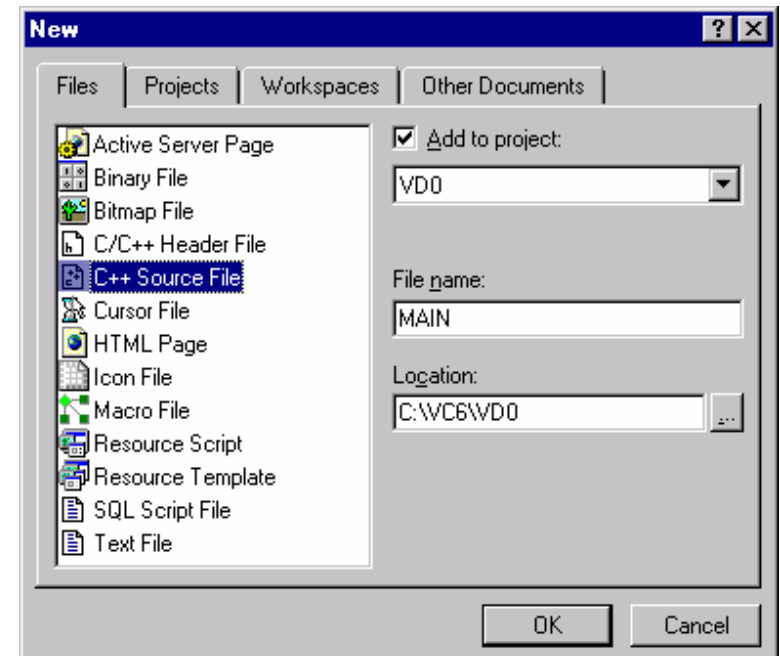
Microsoft Foundation Classes: Cách liên kết thư viện MFC vào ứng dụng. Có thể tùy chọn một trong hai cách sau:

- **Use MFC in Shared DLL**: Chương trình sử dụng thư viện MFC theo cơ chế liên kết động. Tập tin chương trình có kích thước nhỏ nhưng khi thực hiện cần có các tập tin thư viện (DLL) của MFC trong thư mục hệ thống của windows.
- **Use MFC in Static Library**: Thư viện MFC được nhúng vào chương trình trong lúc biên dịch. Tập tin chương trình có kích thước lớn hơn nhưng ứng dụng có thể hoạt động độc lập.

Sau khi ấn định xong, chọn **OK**.

- Khai báo đối tượng thuộc lớp CWinApp: Dùng bất kỳ tập tin CPP nào của dự án để thực hiện công việc này. Vì dự án VD0 đang thực hiện ở đây chưa có tập tin CPP, chúng ta tạo mới tập tin CPP cho dự án. Giả sử tập tin CPP này có tên là MAIN.CPP. Cách thực hiện như sau:



- Chọn mục **File / New**. Trong hộp hội thoại **New**, chọn trang **Files**.



- **C++ Source File** : Loại nội dung tập tin (.cpp).
- **Add To Project** : Bổ sung tập tin vào dự án.
- **File Name** : Tên tập tin (MAIN). Sau đó chọn **OK**.
- Trong màn hình soạn thảo của **main.cpp**, nhập nội dung tập tin:


```
#include "stdafx.h" // Tập tin chứa đăng ký thư viện của MFC
CWinApp theApp, // Đối tượng quản lý tiểu trình chính
```

Sau khi nhập xong, chọn mục **File / Save** để lưu tập tin.

- Biên dịch chương trình: Chọn mục **Build / Build <project name>.exe** hoặc chọn mục  trên thanh công cụ.
Ta đã thực hiện xong một ứng dụng đơn giản trong môi trường windows. Tập tin chương trình được lưu trong thư mục DEBUG (phiên bản biên dịch debug) hoặc RELEASE (phiên bản biên dịch release).
- Chạy chương trình: Có thể chạy chương trình trực tiếp trong VC bằng cách nhấn phím F5 hoặc click chọn  trên thanh công cụ.


Nhận xét: Ứng dụng VD0 không thực hiện một giao tác hay công việc gì cả bởi nó chỉ là một ứng dụng khung - được xây dựng hoàn toàn từ lớp CWinApp của MFC mà không có một xử lý bổ sung nào.

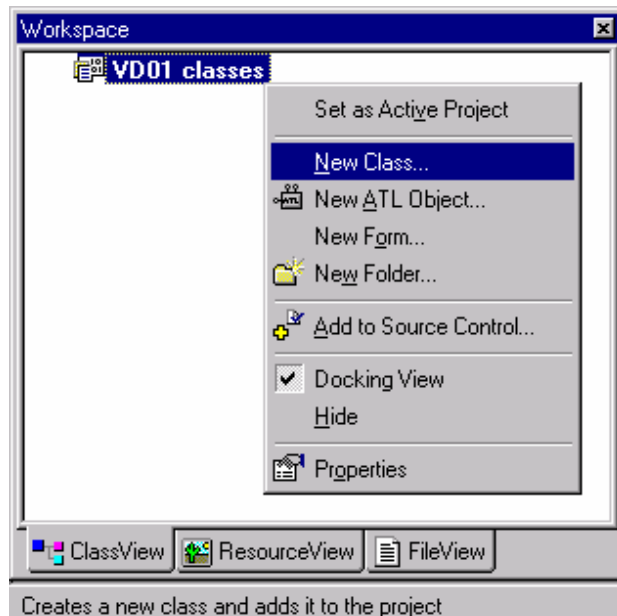
2.7 THỰC HIỆN ỨNG DỤNG GIAO TÁC ĐƠN GIẢN:

Trong phần này ta thiết kế một ứng dụng mà khi thực hiện sẽ hiển thị hộp thông báo "Do You want to Stop" với biểu tượng  và các nút chọn YES- NO. Ứng dụng kết thúc khi người dùng chọn mục YES.

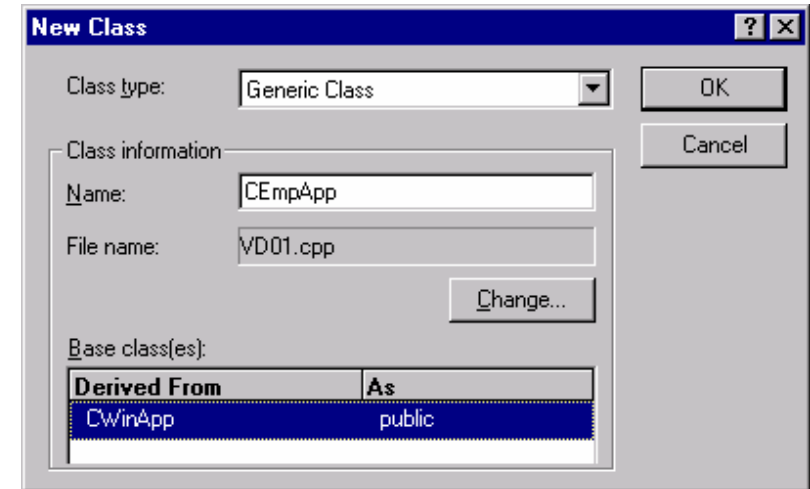
Nhân xét: Ứng dụng chỉ hoàn thành công việc xác định khi đối tượng quản lý tiểu trình chính của ứng dụng tiến hành xử lý thích hợp. Đối tượng này thuộc lớp kế thừa từ lớp CWinApp với xử lý bổ sung nhằm thực hiện công việc mong muốn. Hành vi *InitInstance* của CWinApp là hành vi thích hợp cho việc kế thừa và thực hiện các bổ sung này.

Giả sử dự án của ứng dụng có tên là **VD01**. Các bước thực hiện như sau:

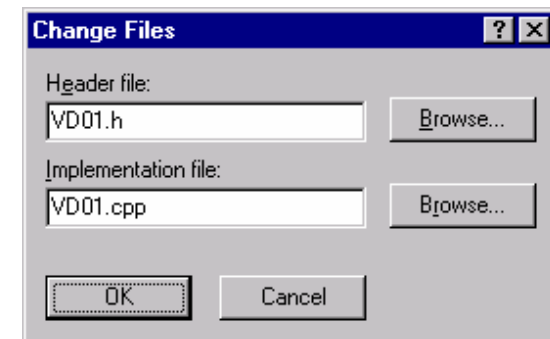
- Tạo dự án VD01 theo các bước như dự án VD0 nhưng dừng lại ở bước "*Khai báo đối tượng thuộc lớp CWinApp*" (không thực hiện bước này). Tiếp tục thực hiện các bước sau đây.
- Tạo mới lớp **CEmpApp** kế thừa từ CWinApp: Mở màn hình **Workspace** của dự án (nếu chưa mở) bằng cách chọn mục menu **View/Workspace** hoặc click chọn biểu tượng  trên thanh công cụ.
- Chọn trang **ClassView**.



- Right-click trên mục **VD01 Classes**, chọn **New Class**.
- Khai báo lớp CEmpApp thông qua hộp hội thoại **New Class**.



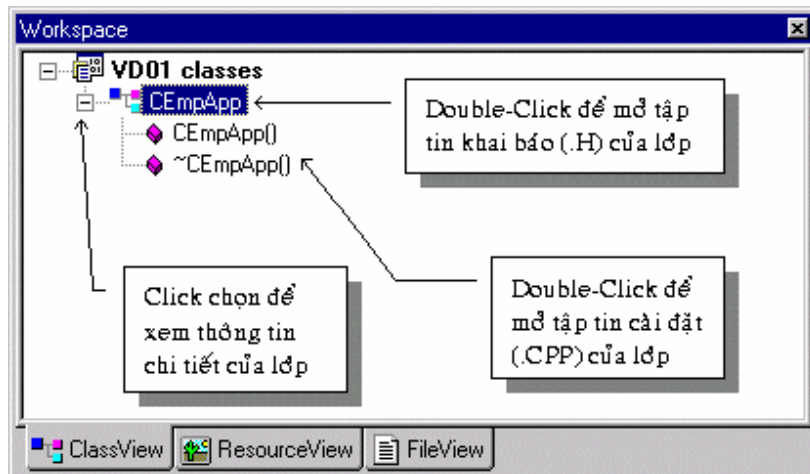
- **Class Type** = *Generic Class*: Chọn loại lớp thông thường vì MFC không hỗ trợ lớp CWinApp trong ClassWizard của nó.
- **Name** = *CEmpApp*: Tên của lớp mới.
- **Change**: Ấn định tên tập tin chứa khai báo (.h) và cài đặt (.cpp) của lớp CEmpApp (nên trùng với tên của dự án : **VD01**):



Sau đó chọn **OK**.

- **Derived From** = *CWinApp*: Chọn CWinApp làm lớp cơ sở.
- **As** = *public*: Kế thừa public. Chọn **OK** để kết thúc.
- Lớp CEmpApp đã được bổ sung vào dự án VD01 cùng với hai tập tin là VD01.H và VD01.CPP:
 - **VD01.H** : Chứa nội dung khai báo (header) của lớp.
 - **VD01.CPP** : Chứa nội dung cài đặt (implement) của lớp.

Có thể mở và chỉnh sửa nội dung các tập tin của lớp bằng cách thao tác trực tiếp trên cấu trúc **ClassView** của màn hình **Workspace**.



- Điều chỉnh lớp **CEmpApp** để nhận được hỗ trợ của ClassWizard:
 - Mở tập tin **VD01.H** chứa khai báo của lớp, bổ sung các nội dung:

```
class CEmpApp : public CWinApp {
public:    CEmpApp();
        virtual ~CEmpApp();

    //{{AFX_VIRTUAL(CEmpApp)
    //}}AFX_VIRTUAL           // Hỗ trợ kế thừa hành vi ảo
    //{{AFX_MSG(CEmpApp)
    //}}AFX_MSG               // Hỗ trợ ấn định xử lý
    message
    DECLARE_MESSAGE_MAP() // Đăng ký MessageMap
}

```

Lưu ý: **//{{** và **//}}** là ký pháp sử dụng của ClassWizard.

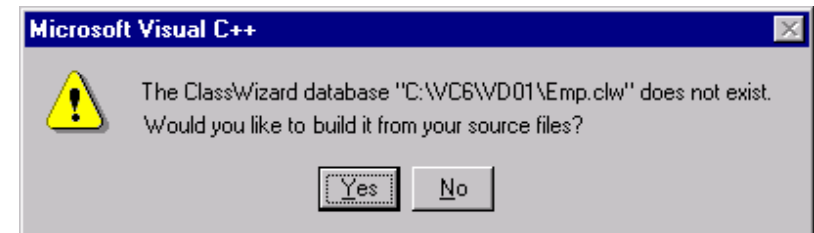
- Mở tập tin **VD01.CPP** chứa cài đặt của lớp, bổ sung nội dung:

```
#include "stdafx.h"
#include "Emp.h"

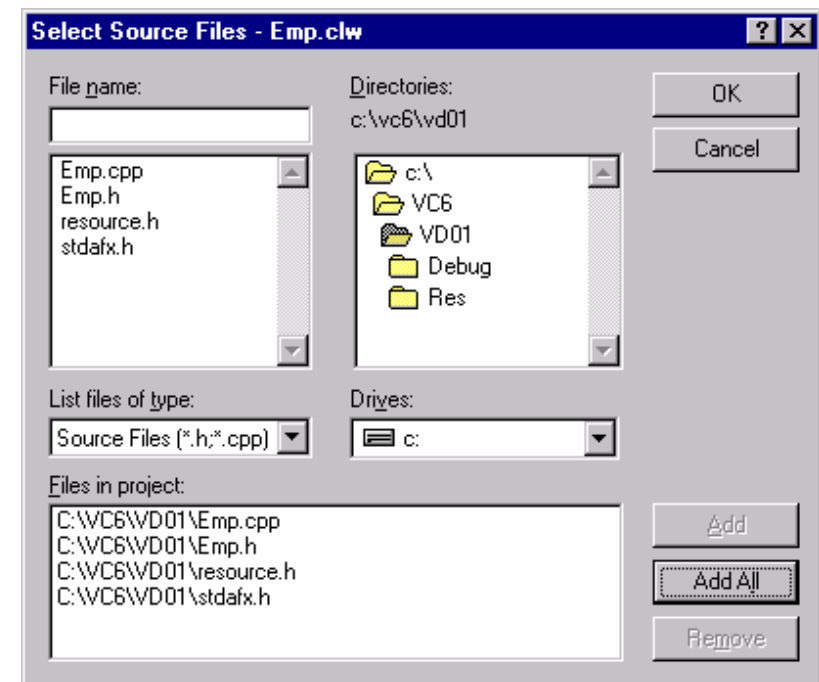
BEGIN_MESSAGE_MAP(CEmpApp, CWinApp)
    //{{AFX_MSG_MAP(CEmpApp) // Bảng message map,
    //}}AFX_MSG_MAP         // sẽ đề cập đến ở các
    END_MESSAGE_MAP()      // phần sau

```

- Khởi tạo thông tin **ClassWizard**. Thao tác này là cơ sở để khai thác tiện ích của ClassWizard trong việc định nghĩa lớp, khai báo thông tin kế thừa, ấn định xử lý message, định nghĩa biến,.
- Chọn menu **View / Classwizard**:



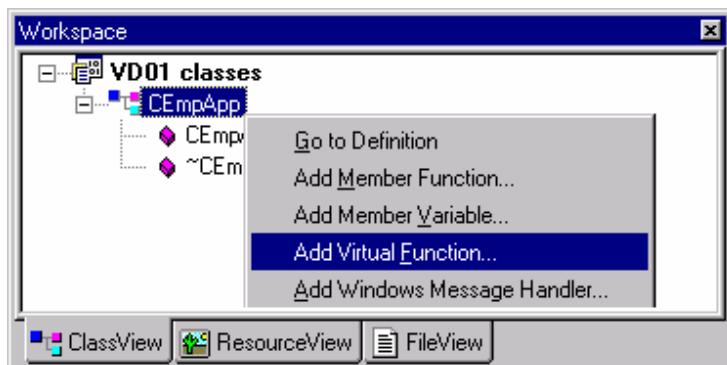
- Chọn **Yes**.



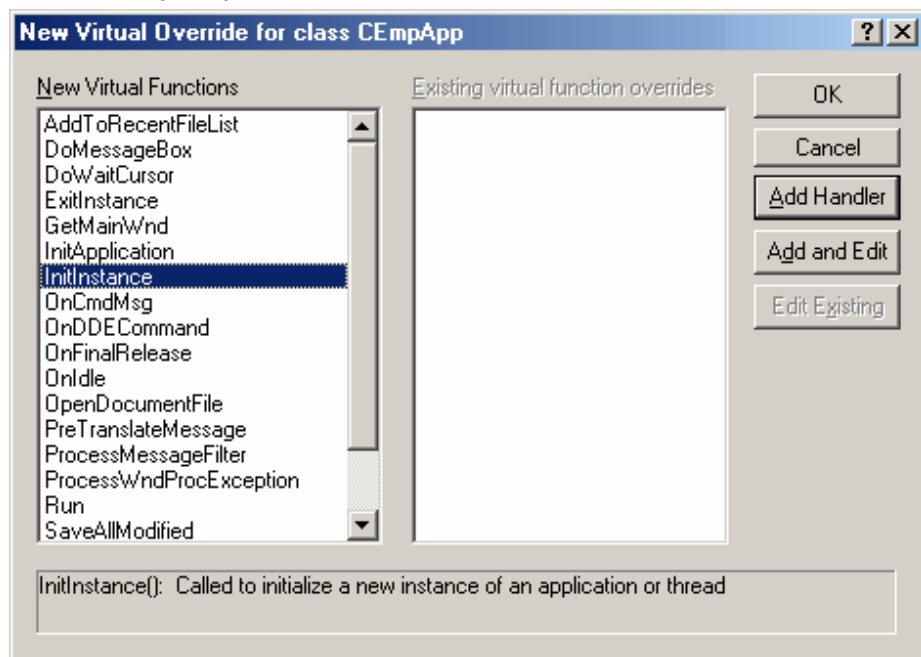
Chọn **Add All**, sau đó chọn **OK**.

- Trong hộp hội thoại **MFC Class Wizard**, Chọn **OK** để kết thúc. Thông tin **ClassWizard** được lưu trong tập tin có cùng tên với tên của dự án và phần mở rộng là **.CLW**.
- Kế thừa hành vi **InitInstance** của lớp CWinApp cho lớp **CEmpApp**. Dùng hành vi này cài đặt xử lý bổ sung như yêu cầu của ứng dụng:

- Trong màn hình **Workspace** của dự án, chọn **ClassView**, right-click trên tiêu đề của lớp **CEmpApp**.



- Chọn mục **Add Virtual Function**.



- Chọn hành vi **InitInstance**, sau đó chọn mục **Add and Edit**.
- Trong phần soạn thảo nội dung của hành vi **InitInstance** thuộc lớp **CEmpApp**, ta cài đặt đoạn chương trình xử lý sau:

```
BOOL CEmpApp::InitInstance()
{
```

```
UINT stop;                                // Biến kiểm tra đồng ý dừng?

do {
    stop = AfxMessageBox( "Do You want to stop",
                           MB_YESNO | MB_ICONQUESTION, 0 );
} while (stop == IDNO );                  // Tiếp tục lặp nếu chọn NO
return TRUE;                             // Xử lý tiến hành bình thường
}
```

- Dùng lớp **CEmpApp** khai báo đối tượng quản lý tiểu trình chính của ứng dụng: Mở tập tin **VD01.CPP** của lớp **CEmpApp**, bổ sung nội dung:

```
#include "stdafx.h"
#include "Emp.h"
CEmpApp theApp; // Đối tượng kiểu CEmpApp, dùng quản lý
                // tiểu trình chính của ứng dụng.
                // Không chỉnh sửa các nội dung khác!
```

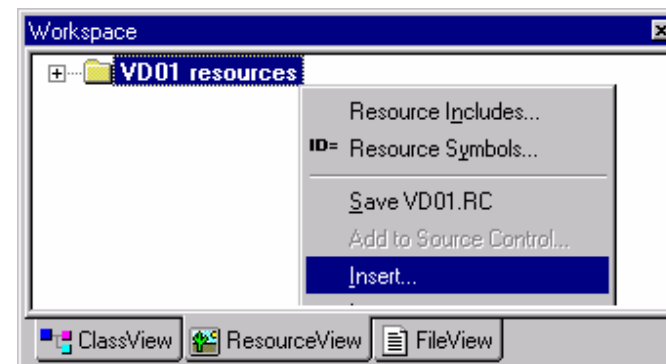
- Biên dịch và chạy chương trình.

2.8 TẠO MỐI ICON RESOURCE CHO ỨNG DỤNG:

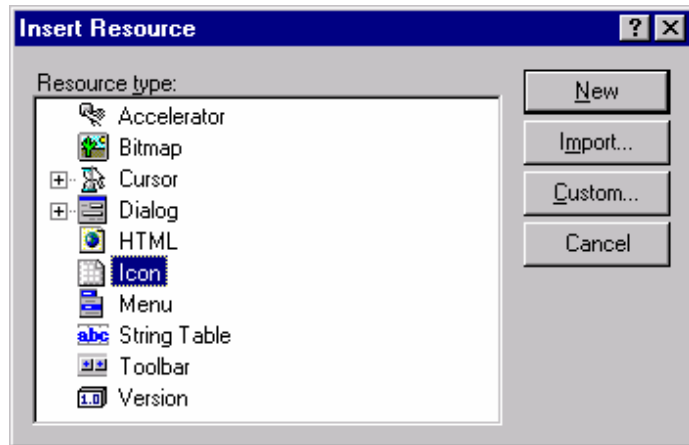
Đối với ứng dụng trong windows, icon không chỉ là hình ảnh trang trí đơn thuần mà còn là yếu tố gợi nhớ về ứng dụng tốt nhất. Windows sử dụng icon của ứng dụng để đại diện cho ứng dụng ở tất cả những nơi nào mà người dùng có thể nhìn thấy và khai thác ứng dụng.

Việc bổ sung icon resource vào dự án của ứng dụng được thực hiện thông qua các bước sau:

- Mở dự án trong VC.
- Mở màn hình **Workspace** của dự án, chọn trang **ResourceView**, right-click trên tiêu đề resource của dự án.

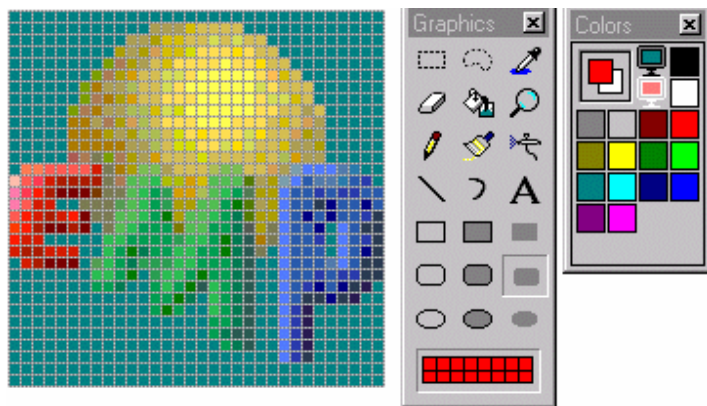


- Chon mục **Insert...**



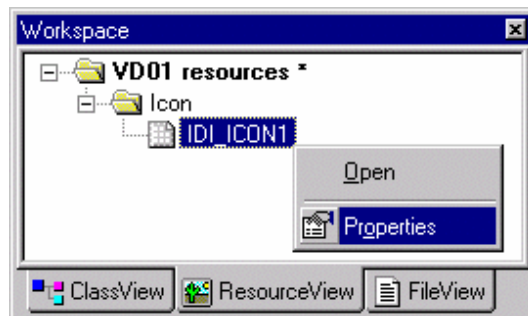
- Chon mục **Icon**, sau đó chon **New**.

Ta nhận được màn hình thiết kế icon. Vẽ icon có nội dung tùy ý.

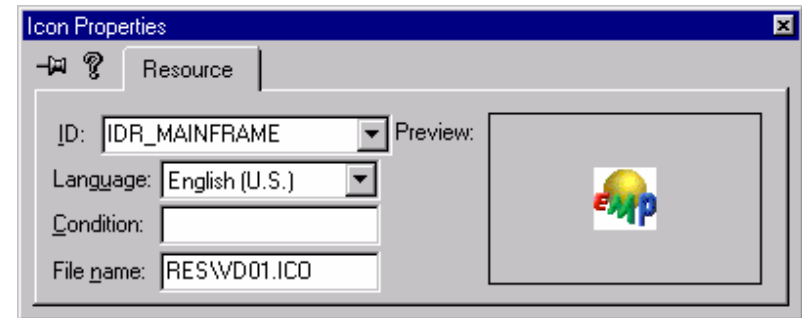


- Ấn định các thông số thuộc tính của icon (số hiệu icon, tên thư mục và tên tập tin chứa icon) trong dự án: Chon **ResourceView** trong **Workspace** của dự án, right-click trên icon mới thêm.

Chon mục **Properties**.



- Thực hiện ấn định thông số của icon thông qua hộp hội thoại sau:



- **ID** : Số hiệu icon. **IDR_MAINFRAME** là số hiệu qui ước dùng cho icon chính của ứng dụng.
- **File name** : Tên tập tin chứa icon. **RES** là thư mục con sẽ được tạo trong thư mục dự án để chứa các tập tin resource.

Sau khi ấn định xong, gõ phím **Enter** để kết thúc.

- Đóng màn hình thiết kế icon resource.
- Biên dịch dự án. Dùng **Windows Explorer** xem tập tin VD01.exe trong thư mục debug của dự án. Lúc này, ứng dụng VD01 đã có icon riêng.

2.9 LƯU TRỮ CHƯƠNG TRÌNH NGUỒN:

Sau khi hoàn tất dự án, một việc rất thường xuyên là lưu giữ lại chương trình nguồn của dự án. Công việc này đòi hỏi phải loại bỏ khỏi thư mục dự án các tập tin không cần thiết. Đó là các tập tin sau:

- Thư mục dự án : Các tập tin .NCB, .PLG, .OPT, .APS.
- Thư mục Debug : Tất cả các tập tin, trừ tập tin .exe cần giữ lại.
- Thư mục Release : Tất cả các tập tin, trừ tập tin .exe cần giữ lại.

2.10 LỚP CString CỦA MFC:

CString là lớp đối tượng của MFC cho phép quản lý một nội dung chuỗi trong bộ nhớ với những đặc tính ưu việt sau:

- Nội dung chuỗi lưu trữ có độ dài lớn với cơ chế sử dụng bộ nhớ tối ưu.
- Xây dựng dựa trên kiểu TCHAR nên thích nghi với bộ mã UNICODE.
- Các hành vi xử lý sẵn sàng cho các tác vụ chuỗi thông thường.

Một số hành vi được sử dụng phổ biến của CString như sau:

- CString()**; Hành vi tạo lập đối tượng chuỗi.
- int GetLength()** const; Trả về số đo chiều dài chuỗi.

- void **Empty**(); Xóa rỗng nội dung đối tượng chuỗi.
- BOOL **IsEmpty**() const; Trả về TRUE nếu nội dung chuỗi là rỗng.
- TCHAR operator **[]**(int *nIndex*) const; Trả về ký tự ở vị trí *nIndex*.
- void **SetAt**(int *nIndex*, TCHAR *ch*); Đặt ký tự *ch* vào vị trí *nIndex*.
- operator **LPCTSTR** () const; Chuyển về kiểu chuỗi của C.
- CString::operator = ; Phép toán gán chuỗi cho đối tượng.
- CString::operator + ; Phép toán cộng chuỗi.
- CString::operator += ; Phép toán nối chuỗi mới vào đối tượng.
- CString Comparison **Operators** ; Trong đó **Operators** là phép toán quan hệ thông thường như: == , < , <= , > , >= , !=.
- CString **Mid**(int *nFirst* [, int *nCount*]) const; Trích chuỗi con giữa.
- CString **Left**(int *nCount*) const; Trích chuỗi con bên trái.
- CString **Right**(int *nCount*) const; Trích chuỗi con bên phải.
☞ *nCount* là chiều dài (số ký tự) của chuỗi con được trích ra.
- void **MakeUpper**() / void **MakeLower**(); Viết hoa / thường nội dung.
- void **Format**(LPCTSTR *lpzFormat*, ...); Tạo nội dung cho chuỗi.
- void **TrimLeft**() / void **TrimRight**(); Hủy khoảng trắng trái / phải.
- BOOL **LoadString**(UINT *nID*); Gán nội dung cho chuỗi bởi một hằng chuỗi trong string table resource. *nID* là số hiệu của hằng chuỗi.

THỰC HÀNH:

1. Viết ứng dụng cho phép hiển thị đường dẫn đến tập tin chương trình và chuỗi tham số dòng lệnh thực thi nó.

2. Viết ứng dụng có cài đặt DoMessageBox sao cho hàm AfxMessageBox sử dụng trong ứng dụng hiển thị hộp thông báo với icon riêng của ứng dụng.

HD: Sử dụng hàm: int **MessageBoxIndirect**(LPMSGBOXPARAMS *pMsg*);

```
typedef struct {
    UINT        cbSize;        // Kích thước, = sizeof( MSGBOXPARAMS )
    HWND        hwndOwner;    // Handle của cửa sổ cha, có thể = NULL
    HINSTANCE    hInstance;    // Handle ứng dụng =
    AfxGetInstanceHandle()
    LPCTSTR     lpzText;        // Nội dung chuỗi thông báo
    LPCTSTR     lpzCaption;    // Nội dung chuỗi tiêu đề
    DWORD       dwStyle;        // Dạng hộp, chứa thông số MB_USERICON
    LPCTSTR     lpzIcon;        // Chuỗi tên icon được hiển thị trong hộp
    DWORD_PTR   dwContextHelpId; // Số hiệu giúp đỡ
    MSGBOXCALLBACK lpfnMsgBoxCallback; // Hàm xử lý
    DWORD       dwLanguageId; // Số hiệu ngôn ngữ sử dụng
} MSGBOXPARAMS, *PMSGBOXPARAMS;
```

CHƯƠNG 3:

Các lớp GIAO DIỆN đồ họa của MFC

3.1 CÁC CÔNG CỤ GIAO DIỆN ĐỒ HỌA:

Để hỗ trợ ứng dụng trong việc trang trí giao diện đồ họa (Graphic Device Interface - GDI) như màn hình, máy in., windows cung cấp một hệ thống các công cụ đồ họa cơ bản như sau:

- *Pen* : Công cụ vẽ điểm hoặc đường thẳng.
- *Brush* : Công cụ tô màu.
- *Font* : Công cụ qui định kiểu ký tự cho nội dung văn bản.
- *Bitmap*: Công cụ quản lý vùng ảnh điểm.
- *Palette*: Công cụ quản lý bộ màu hiển thị.
- *Region*: Công cụ quản lý vùng hiển thị có dạng bất kỳ.

Thông qua các công cụ nói trên, ứng dụng trong windows có thể thực hiện các tác vụ đồ họa cơ bản một cách dễ dàng mà không phải bận tâm đến cấu trúc vật lý của thiết bị hiển thị. Các công cụ này gọi là công cụ GDI.

3.2 DEVICE CONTEXT:

Việc khai thác khả năng của các công cụ GDI được thực hiện thông qua một cấu trúc chứa thông tin quản lý thiết bị hiển thị: Device Context (DC). Tại mỗi thời điểm, mỗi DC được phép gắn với một thiết bị hiển thị đồ họa và sử dụng một bộ các công cụ GDI duy nhất. Muốn sử dụng một công cụ GDI mới thì DC phải chấm dứt sử dụng công cụ tương ứng mà nó đang dùng.

Mỗi khi một DC được tạo mới, hệ thống tự động chuẩn bị một bộ công cụ GDI (trừ bitmap) mặc nhiên cho nó. Ứng dụng có thể tùy nghi thay đổi các công cụ này cho phù hợp với nhu cầu của ứng dụng.

3.3. TỌA ĐỘ TRÊN GIAO DIỆN ĐỒ HỌA:

Việc định vị trên giao diện đồ họa được thực hiện dựa trên hệ trục tọa độ có gốc (0,0) là góc trái-trên (top-left corner) của giao diện đồ họa, chiều dương của trục hoành (trục x) hướng từ trái sang phải, chiều dương của trục tung (trục y) hướng từ trên xuống dưới. Trên mặt phẳng giao diện đồ họa đó:

- Điểm: Biểu diễn bằng cặp tọa độ (x,y).
- Hình chữ nhật: Biểu diễn bằng cặp đỉnh góc trái trên và góc phải dưới.

Thông tin về điểm, vùng hình chữ nhật và kích thước của các đối tượng trong mặt phẳng giao diện đồ họa được đặc tả thông qua các cấu trúc sau:

- Cấu trúc quản lý 1 điểm:

```
typedef struct _POINT {  
    int x, y           // Tọa độ ( hoành độ, tung độ ) của điểm  
} POINT ;
```
- Cấu trúc quản lý 1 vùng hình chữ nhật:

```
typedef struct _RECT {  
    int top, left,      // Tọa độ góc trái trên hình chữ nhật.  
    int right, bottom; // Tọa độ góc phải dưới hình chữ nhật.  
} RECT ;
```
- Cấu trúc quản lý chiều rộng và chiều cao vùng hình chữ nhật:

```
typedef struct _SIZE {  
    int cx, cy,         // Chiều rộng, chiều cao vùng hình chữ  
                        // nhật.  
} SIZE ;
```

3.4 CÁC LỚP MFC HỖ TRỢ GDI:

MFC cung cấp một hệ thống phong phú các lớp với đầy đủ chức năng nhằm hỗ trợ người dùng tối đa trong lập trình GDI. Với việc vận dụng kỹ thuật OOP trên các lớp GDI của MFC, người dùng dễ dàng tạo ra ứng dụng mong muốn mà không phải viết quá nhiều mã lệnh trong chương trình.

3.4.1 Các lớp đối tượng điểm, hình chữ nhật:

- **CPoint** : Lớp đối tượng điểm, tương ứng với cấu trúc POINT.
- **CRect** : Lớp đối tượng vùng hình chữ nhật trên giao diện đồ họa, tương ứng với cấu trúc RECT. CRect có các hành vi đặc trưng sau:
 - **CRect()**; Tạo lập đối tượng vùng hình chữ nhật rỗng.
 - **CRect(int *left*, int *top*, int *right*, int *bottom*);** Tạo lập đối tượng vùng hình chữ nhật có thông số xác định.
 - **int Width() const**; Trả về độ rộng vùng hình chữ nhật.
 - **int Height() const**; Trả về độ cao vùng hình chữ nhật.
 - **void NormalizeRect()**; Hợp lý hóa các thành phần tọa độ của rect mà không làm thay đổi vị trí và kích thước của nó.

VD: Giả sử khởi tạo một rect không hợp lệ như sau:

```
CRect rect( 100, 0, 0, 50 ); // rect.left = 100 > rect.right = 0  
rect.NormalizeRect();  
rect trở nên hợp lệ với: rect ( 0, 0, 100, 50 );
```
- **CSize** : Lớp đối tượng kích thước, tương ứng với cấu trúc SIZE.

3.4.2 Lớp CPen:

CPen là lớp đối tượng quản lý *pen*, một công cụ GDI quan trọng của DC. Thông qua đối tượng này, DC có thể thay đổi màu sắc, nét vẽ của

điểm hay đường thẳng mà DC sẽ thực hiện. CPen có các hành vi đặc trưng sau:

- **CPen();** Tạo lập đối tượng pen rỗng. Chúng ta phải khởi tạo tham số cho đối tượng này trước khi sử dụng.
- **CPen (** *// Tạo lập đối tượng pen với tham số*
 int nPenStyle, *// Kiểu nét vẽ*
 int nWidth, *// Độ rộng nét vẽ (=1: mặc nhiên)*
 COLORREF crColor *// Màu sắc của nét vẽ*
);
 nPenStyle có thể nhận một trong các giá trị sau:
 PS_SOLID : PS_DASH :
 PS_DOT : PS_DASHDOT :
 crColor có thể nhận giá trị từ macro phối màu RGB() như sau:
 RGB (int *màu_đỏ* , int *màu_xanh_lá_cây* , int *màu_xanh_dương*
) -----
 Mỗi màu được đặc trưng bằng một giá trị trong đoạn 0255 phần ánh độ sáng của nó. Bộ phối màu theo qui tắc phối màu tự nhiên.
- **BOOL CreatePen(int nPenStyle, int nWidth, COLORREF crColor**); Khởi tạo thông số cho đối tượng pen. Ý nghĩa tham số như trên.
- operator HPEN() const; Toán tử chuyển kiểu, trả về handle của pen (HPEN) quản lý bởi đối tượng.

3.4.3 Lớp CBrush:

CBrush là lớp đối tượng quản lý brush, công cụ GDI của DC. Thông qua đối tượng này, DC có thể thay đổi màu sắc, dạng của nét tô trong một vùng hình chữ nhật mà DC sẽ thực hiện. CBrush có các hành vi đặc trưng sau:

- **CBrush();** Tạo lập đối tượng brush rỗng.
- **CBrush(COLORREF crColor);** Tạo lập đối tượng brush có màu tô tương ứng với màu qui định bởi tham số *crColor*.
- **CBrush(int nIndex, COLORREF crColor);** Tạo lập đối tượng brush có màu tô *crColor*, và nét tô *nIndex*.
 Giá trị nét tô có thể là: HS_VERTICAL (đường kẻ dọc đứng), HS_HORIZONTAL (đường kẻ ngang).
- **BOOL CreateSolidBrush(COLORREF crColor);** Tạo đặc tính tô đặc với màu *crColor* cho đối tượng brush chưa có thông số.
- operator HBRUSH() const; Toán tử chuyển kiểu, trả về handle của brush (HBRUSH) quản lý bởi đối tượng.

3.4.4 Lớp CFont:

CFont là lớp đối tượng quản lý font, công cụ GDI của DC. Thông qua đối tượng này, DC thực hiện ấn định font, kiểu dáng, kích thước của bộ ký

tự được sử dụng cho việc hiển thị các nội dung văn bản. Các hành vi đặc trưng:

- **CFont();** Tạo lập đối tượng font rỗng.
- **CFont(const LOGFONT* lpLogFont);** Tạo lập đối tượng font với thông số đầy đủ. Giá trị thông số được ấn định bởi tham số kiểu cấu trúc LOGFONT chứa thông tin. LOGFONT được khai báo như sau:

```
typedef struct tagLOGFONT {  
    LONG lfHeight;                // chiều cao của ký tự  
    LONG lfWidth;                // chiều rộng trung bình các ký tự  
    LONG lfEscapement;           // góc (1/10) giữa hướng in và trục X  
    LONG lfOrientation;          // góc (1/10) giữa ký tự và trục X (No 9x)  
    LONG lfWeight;               // mức độ đậm của font chữ (0 - 1000)  
    BYTE lfItalic;                // = TRUE : Chữ nghiêng  
    BYTE lfUnderline;            // = TRUE : Chữ gạch dưới  
    BYTE lfStrikeOut;            // = TRUE : Chữ gạch ngang thân  
    BYTE lfCharSet;              // bộ ký tự (=DEFAULT_CHARSET )  
    BYTE lfOutPrecision;         // = OUT_DEFAULT_PRECIS  
    BYTE lfClipPrecision;        // = CLIP_DEFAULT_PRECIS  
    BYTE lfQuality;              // = DEFAULT_QUALITY  
    BYTE lfPitchAndFamily;       // =DEFAULT_PITCH|FF_DONTCARE  
    TCHAR lfFaceName[LF_FACESIZE]; // Chuỗi tên của font  
} LOGFONT ;
```

☞ Win9x chỉ dùng *lfEscapement*.

- **int GetLogFont(LOGFONT * pLogFont);** Lấy thông tin của font quản lý bởi đối tượng font chữ. Kết quả được điền vào biến kiểu LOGFONT chỉ bởi tham số kiểu con trỏ LOGFONT* : *pLogFont*.
- operator HFONT(); Toán tử chuyển kiểu, trả về handle của font được quản lý bởi đối tượng.
- **BOOL CreateFontIndirect(LOGFONT *lpLogFont);** Khởi tạo thông số cho đối tượng font từ thông tin lưu trong cấu trúc chỉ bởi *lpLogFont*.

3.4.5 Lớp CBitmap:

CBitmap là lớp đối tượng quản lý bitmap, một công cụ GDI quan trọng giúp quản lý vùng ảnh điểm (pixels) của DC. Thông qua đối tượng này, DC có thể dễ dàng tạo nội dung trang trí trên giao diện đồ họa từ nội dung ảnh có sẵn được lưu trong tập tin, đồng thời thực hiện các tác vụ xử lý ảnh cơ bản trên nội dung đó. CBitmap cung cấp các hành vi đặc trưng sau:

- **CBitmap();** Tạo lập đối tượng bitmap rỗng.

- **BOOL LoadBitmap(UINT *nIDResource*);** Tạo nội dung cho đối tượng bitmap với thông tin được lấy từ một ảnh trong resource. *nIDResource* : số hiệu của ảnh bitmap trong resource của chương trình.
- **BOOL CreateCompatibleBitmap (CDC* *pDC*, // Con trỏ đối tượng DC tương ứng
int *nWidth*, // Chiều rộng,
int *nHeight* // chiều cao tính bằng pixel của bitmap được tạo.**); Tạo nội dung cho đối tượng bitmap với thông tin về màu sắc, độ phân giải tương ứng với DC chỉ bởi con trỏ tham số *pDC*.
- **int GetBitmap(BITMAP* *pBitMap*);** Lấy thông tin về bitmap được quản lý bởi đối tượng. Kết quả nhận được sẽ được điền vào biến có kiểu BITMAP chỉ bởi tham số kiểu con trỏ BITMAP* : *pBitmap*.
Cấu trúc **BITMAP** bao gồm các trường có ý nghĩa như sau:

```
typedef struct tagBITMAP { /* bm */
    int bmType;           // Kiểu bitmap,
    int bmWidth;          // Chiều rộng bitmap tính bằng pixel
    int bmHeight;         // Chiều cao bitmap tính bằng pixel
    int bmWidthBytes;     // Kích thước 1 dòng pixel trong bitmap
    BYTE bmPlanes;        // Số màu.
    BYTE bmBitsPixel;     // Số bit màu của 1 pixel
    LPVOID bmBits;        // Địa chỉ vùng nhớ chứa pixel của
    bitmap
} BITMAP;
```

- operator **HBITMAP()** const; Toán tử chuyển kiểu, trả về handle của bitmap được quản lý bởi đối tượng.

3.4.6 Lớp CPalette:

CPalette là lớp đối tượng quản lý palette, một công cụ GDI của DC. Thông qua đối tượng này, DC có thể tạo ra các hiệu ứng màu trên giao diện đồ họa bằng cách thay đổi các bộ màu một cách phù hợp.

- **CPalette()** ; Tạo lập đối tượng palette rỗng.
- **CreatePalette(LPLOGPALETTE *lpLogPalette*);** Khởi tạo thông số cho đối tượng palette rỗng. Con trỏ tham số *lpLogPalette* chỉ đến biến kiểu cấu trúc LOGPALETTE chứa thông tin bộ màu dùng khởi tạo.

```
typedef struct tagLOGPALETTE {
    WORD palVersion;      // Số hiệu palette hệ thống
    WORD palNumEntries;    // Số màu sử dụng
    PALETTEENTRY palPalEntry[1]; // Chứa các giá trị màu,
                                // có số phần tử mảng bằng palNumEntries
} LOGPALETTE ;
```

☞ Mỗi phần tử của palPalEntry có kiểu PALETTEENTRY phản ánh giá trị màu dùng hiển thị màu chỉ mục (color index) tương ứng với vị trí thứ tự của nó. Cấu trúc PALETTEENTRY có nội dung như sau:

```
typedef struct tagPALETTEENTRY {
    BYTE peRed;           // Giá trị thành phần màu đỏ
    BYTE peGreen;         // Giá trị thành phần màu xanh lá cây
    BYTE peBlue;          // Giá trị thành phần màu xanh dương
    BYTE peFlags;         // Vai trò màu trong hệ thống.
} PALETTEENTRY;
```

- operator **HPALETTE()** const; Toán tử chuyển kiểu, trả về handle của palette quản lý bởi đối tượng.

3.4.7 Lớp CRgn:

CRgn là lớp đối tượng quản lý region, một công cụ GDI quan trọng giúp DC ấn định vùng ảnh xử lý có hình dạng tùy ý trên giao diện đồ họa của nó.

- **CRgn()** ; Tạo lập đối tượng vùng ảnh rỗng.
- **BOOL CreateRectRgn(int *x1*, int *y1*, int *x2*, int *y2*);** Khởi tạo thông số cho đối tượng vùng ảnh bằng 1 hình chữ nhật với tọa độ góc trái trên và góc phải dưới lần lượt là (*x1*, *y1*) và (*x2*, *y2*).
- **BOOL CreateEllipticRgn(int *x1*, int *y1*, int *x2*, int *y2*);** Khởi tạo thông số cho đối tượng vùng ảnh bởi hình ellipse nội tiếp hình chữ nhật có góc trái trên (*x1*, *y1*) và góc phải dưới (*x2*, *y2*).
- **BOOL CreatePolygonRgn (LPPOINT *lpPoints*, // Mảng chứa tọa độ các điểm ziczac
int *nCount*, // Số phần tử POINT trong mảng trên
int *nMode* // = WINDING**

);
Khởi tạo thông số cho đối tượng vùng ảnh bởi một đường ziczac khép kín qua các điểm có tọa độ xác định và được lưu trong một mảng.

**int CombineRgn (CRgn* *pRgn1*, // Con trỏ đối tượng quản lý vùng ảnh thứ nhất
CRgn* *pRgn2*, // Con trỏ đối tượng quản lý vùng ảnh thứ hai
int *nCombineMode* // Cách kết hợp hai vùng ảnh**); Tạo thông số cho đối tượng vùng ảnh trên cơ sở kết hợp hai vùng ảnh đã có. Tham số *nCombineMode* có thể là:
RGN_AND : Vùng ảnh kết quả là vùng giao nhau giữa hai vùng ảnh.
RGN_OR : Vùng ảnh kết quả là vùng hợp giữa hai vùng ảnh.
RGN_DIFF : Vùng ảnh kết quả là vùng bù giữa hai vùng ảnh.

- operator HRGN() const; Toán tử chuyển kiểu, trả về handle của vùng ảnh quản lý bởi đối tượng.

3.5 LỚP CDC:

CDC là lớp đối tượng quản lý DC. Thông qua đối tượng DC, khả năng của các công cụ đồ họa được khai thác cho việc trang trí giao diện đồ họa quản lý bởi DC. CDC có các hành vi đặc trưng như sau:

- **CDC()**; Tạo lập đối tượng DC rỗng.
- virtual BOOL **CreateCompatibleDC**(CDC* *pDC*); Khởi tạo thông số cho đối tượng DC một cách tương thích với một DC có sẵn được chỉ bởi con trỏ tham số *pDC*.
- virtual BOOL **DeleteDC**(); Hủy bỏ đối tượng DC.
- CPen* **SelectObject**(CPen* *pPen*); Chọn công cụ vẽ mới cho DC. *pPen*: con trỏ đến đối tượng pen sẽ được dùng cho DC. Hàm trả về con trỏ chỉ đến đối tượng pen mà DC đang sử dụng.
☞ Có thể sử dụng hành vi này cho các công cụ trang trí khác của DC. Kết quả trả về là con trỏ đến đối tượng đang dùng tương ứng.
- CPent* **GetCurrentPen**(); Trả về giá trị con trỏ của đối tượng pen đang được sử dụng bởi DC.
☞ Một cách tương tự cho các công cụ trang trí khác.
- virtual COLORREF **SetBkColor**(COLORREF *crColor*); Đặt màu nền cho DC. *crColor* là giá trị màu đặt.
- COLORREF **GetBkColor**(); Trả về giá trị màu nền của DC.
- virtual COLORREF **SetTextColor**(COLORREF *crColor*); Ấn định màu được sử dụng để hiển thị các nội dung văn bản trên DC.
- COLORREF **GetTextColor**(); Trả về giá trị màu hiện dùng để hiển thị các nội dung văn bản trên DC.
- int **SetBkMode**(int *nBkMode*); Ấn định chế độ hiển thị nền ký tự biểu diễn nội dung văn bản. *nBkMode* chứa thông số ấn định:
TRANSPARENT : Nền chữ hiển thị trong suốt.
OPAQUE : Chữ hiển thị có màu nền.
- int **GetBkMode**(); Lấy chế độ hiển thị chữ của DC.
- virtual BOOL **TextOut**(
int *x*, *y*, // Tọa độ bắt đầu hiển thị
LPCTSTR *lpszString*, // Nội dung chuỗi hiển thị.
int *nCount* // Chiều dài chuỗi.
); Hiển thị nội dung chuỗi văn bản lên giao diện đồ họa của DC.
- virtual int **DrawText**(
LPCTSTR *lpszString*, // Nội dung chuỗi hiển thị
int *nCount*, // Chiều dài chuỗi
LPRECT *lpRect*, // Con trỏ đến biến kiểu RECT

UINT *nFormat* // Chứa thông tin canh chỉnh chuỗi hiển thị.
); In nội dung chuỗi lên giao diện đồ họa của DC với các canh chỉnh.

lpRect: Chỉ đến biến kiểu RECT chứa thông tin giới hạn vùng hiển thị.

nFormat: Cho phép kết hợp một cách hợp lý các canh chỉnh trong vùng hình chữ nhật giới hạn. Các thông số canh chỉnh có thể là:

DT_CENTER : Canh giữa theo chiều ngang
DT_VCENTER : Canh giữa theo chiều dọc
DT_RIGHT : Canh phải.

- CPoint **MoveTo**(int *x*, int *y*); Ấn định điểm vẽ hiện hành.
- BOOL **LineTo**(int *x*, int *y*); Vẽ đường thẳng từ điểm vẽ hiện hành đến điểm có tọa độ (*x*, *y*).
- void **FillRect**(
LPRECT *lpRect*, // Con trỏ chỉ đến biến kiểu RECT.
CBrush* *pBrush* // Con trỏ đến đối tượng brush dùng tô màu.
); Tô màu vùng hình chữ nhật được xác định bởi các giá trị chứa trong biến kiểu RECT do tham số *lpRect* chỉ đến.
- void **Draw3dRect**(// Vẽ hình chữ nhật 3 chiều
LPRECT *lpRect*, // Con trỏ đến biến kiểu RECT
COLORREF *clrTopLeft*, // Màu vẽ cạnh trái và cạnh trên.
COLORREF *clrBottomRight* // Màu vẽ cạnh phải và cạnh dưới
);
- BOOL **DrawEdge**(
LPRECT *lpRect*, // Con trỏ đến biến kiểu RECT.
UINT *nEdge*, // Cách vẽ trên các gờ (trong, ngoài) cạnh.
UINT *nFlags* // Các cạnh được vẽ.
); Vẽ khung chữ nhật với hiệu ứng 3 chiều.
nEdge có thể kết hợp các giá trị sau:
BDR_RAISEDINNER : Vẽ nổi gờ trong
BDR_SUNKENINNER : Vẽ chìm gờ trong
BDR_RAISEDOUTER : Vẽ nổi gờ ngoài
BDR_SUNKENOUTER : Vẽ chìm gờ ngoài
nFlags có thể kết hợp các giá trị sau:
BF_RECT : Vẽ tất cả các cạnh
BF_TOPLEFT : Vẽ cạnh trái và cạnh trên
BF_BOTTOMRIGHT : Vẽ cạnh phải và cạnh dưới
- BOOL **DrawState**(
CPoint *pt*, // Điểm đặt góc trái trên của ảnh trên DC
CSize *size*, // Kích thước vùng hiển thị ảnh
CBitmap* *pBitmap*, // Con trỏ đối tượng bitmap được vẽ


```

UINT nFlags,           // = DST_BITMAP (vẽ bitmap)
CBrush* pBrush        // Con trỏ đối tượng Brush, sử dụng khi vẽ
                        // Bitmap ẩn: nFlags |= DSS_DISABLED
); Vẽ bitmap hoặc icon lên DC.
▪ BOOL BitBlt (
    int x, int y,           // Tọa độ góc trái trên và
    int nWidth, int nHeight, // Kích thước vùng nhận ảnh.
    CDC* pSrcDC,           // Con trỏ đối tượng quản lý DC nguồn.
    int xSrc, int ySrc,     // Góc trái trên phần ảnh nguồn được
chép.
    DWORD dwRop           // Cách chụp pixel từ ảnh nguồn.
); Chụp nội dung phần ảnh bắt đầu từ vị trí (xSrc, ySrc) trong DC
nguồn sang vùng nhận ảnh bắt đầu từ vị trí (x, y), với kích thước
(nWidth, nHeight) trong DC quản lý bởi đối tượng.
Giá trị pixel được chuyển vào vùng nhận ảnh tùy thuộc vào giá trị
tham số dwRop. Một số giá trị dùng cho tham số này có thể như
sau:
• SRCCOPY : Giá trị pixel lấy trực tiếp từ pixel của ảnh nguồn.
• SRCPAINT : Là kết quả OR của pixel ảnh nguồn và ảnh nhận.
• SRCAND : Là kết quả AND của pixel ảnh nguồn và ảnh
nhận.
▪ BOOL MaskBlt (
    int x, int y,           // Tọa độ góc trái trên và
    int nWidth, int nHeight, // Kích thước vùng nhận ảnh
    CDC* pSrcDC,           // Con trỏ đối tượng quản lý DC nguồn
    int xSrc, int ySrc,     // Góc trái trên phần ảnh được chép
    CBitmap& maskBitmap,   // Con trỏ đối tượng bitmap mặt nạ
    int xMask, int yMask,   // Góc trái trên vùng ảnh làm mặt nạ
    DWORD dwRop           // Cách chụp pixel từ ảnh nguồn.
); Tương tự BitBlt nhưng sử dụng mặt nạ lọc pixel. Đối tượng
maskBitmap sử dụng ảnh trắng đen (monochrome bmp) mà mỗi
pixel "đen" sẽ ngăn việc chuyển pixel ở vị trí tương ứng từ DC
nguồn sang DC quản lý bởi đối tượng, các vị trí khác được chuyển
bình thường.
▪ BOOL StretchBlt (
    int x, int y,           // Tọa độ góc trái trên và
    int nWidth, int nHeight, // Kích thước vùng nhận ảnh.
    CDC* pSrcDC,           // Con trỏ đối tượng DC nguồn.
    int xSrc, int ySrc,     // Tọa độ góc trái trên và
    int nSrcWidth, int nSrcHeight, // Kích thước phần ảnh được
chép.

```

```

    DWORD dwRop           // Cách chụp pixel từ ảnh nguồn.
); Tương tự BitBlt nhưng ảnh đích và ảnh nguồn có thể có kích
thước khác nhau nên tạo hiệu ứng co giãn ảnh chép được so với
ảnh nguồn.
▪ BOOL DrawIcon (
    int x, int y,           // Tọa độ góc trái trên nơi đặt icon trên DC
    HICON hIcon           // Handle của icon
); Vẽ icon hIcon lên vị trí (x, y) của DC quản lý bởi đối tượng.
3.6 LỚP CImageList:
CImageList là lớp đối tượng imagelist. Mỗi imagelist cho phép quản lý
danh sách ảnh có cùng kích thước và hỗ trợ nhiều tiện ích trên chúng.
▪ CImageList ( ); Tạo lập đối tượng imagelist rỗng.
▪ BOOL Create (
    UINT nBitmapID,        // Số hiệu của resource bitmap chứa các
ảnh
    int cx,                // Độ rộng mỗi ảnh trong bitmap nói trên
    int nGrow,             // Số ảnh trong bitmap
    COLORREF crMask // Màu mặt nạ (không hiển thị)
); Khởi tạo nội dung cho đối tượng từ một bitmap trong resource.
▪ int GetImageCount ( ); Số ảnh của imagelist quản lý bởi đối tượng.
▪ int Add (
    CBitmap* pbmImage,     // Con trỏ đối tượng bitmap của ảnh
mới.
    CBitmap* pbmMask |     // Đối tượng bitmap mặt nạ hoặc
[ COLORREF crMask ] // màu mặt nạ của ảnh mới.
); Thêm một ảnh (bitmap) vào imagelist.
▪ BOOL BeginDrag( int nImage, CPoint ptHotSpot ); Chuẩn bị
chuyển ảnh thứ nImage trong imagelist với vị trí bắt đầu ptHotSpot.
▪ static BOOL DragEnter( CWnd* pWndLock, CPoint point ); Cấm
cửa sổ pWndLock, nơi mà imagelist đang thực hiện chuyển ảnh.
▪ static BOOL DragMove(CPoint pt); Chuyển ảnh nImage đến vị trí
pt.
▪ static BOOL DragLeave( CWnd* pWndLock ); Chấm dứt tình trạng
bị cấm của cửa sổ pWndLock.
▪ static void EndDrag ( ); Chấm dứt tác vụ chuyển ảnh.
▪ static BOOL DragShowNolock( BOOL bShow ); Hiển thị hoặc che
ảnh trong quá trình chuyển hình.
▪ BOOL Draw (
    CDC* cdc,              // Đối tượng DC dùng vẽ hình
    int nImage,            // Số thứ tự hình được vẽ trong imagelist
    POINT pt,             // Tọa độ góc trái trên nơi vẽ hình

```

```

    UINT nStyle // Kiểu vẽ = ILD_NORMAL
); Vẽ hình nImage của imagelist quản lý bởi đối tượng lên DC.
▪ BOOL DrawIndirect (
    CDC* pDC, int nImage, POINT pt, // Tương tự Draw
    SIZE sz, // Kích thước vùng nhận ảnh
    POINT ptOrigin, // Góc trái trên phần ảnh được vẽ
    UINT fStyle, // Kiểu ảnh ( = ILD_NORMAL )
    DWORD dwRop, // Cách chép pixel ( = SRCCOPY )
    COLORREF rgbBack, // Màu vùng bị lợp = CLR_DEFAULT
    COLORREF rgbFore // Màu phối hợp cho fStyle có thông số
    // ILD_BLEND25 hoặc ILD_BLEND50 ( = CLR_DEFAULT )
); Vẽ hình nImage.
☞ Đối tượng GDI được khởi tạo trong chương trình bởi hành vi Createxxx
cần được hủy bỏ khi chấm dứt sử dụng để tránh lãng phí bộ nhớ hệ
thống:
- Các đối tượng công cụ GDI, sử dụng hành vi: DeleteObject( );
- Các đối tượng quản lý thiết bị đồ họa, sử dụng hành vi: DeleteDC(
);

```

CHƯƠNG 4:

Cửa sổ giao diện và lớp CWnd

4.1 CỬA SỔ GIAO DIỆN:

Cửa sổ giao diện là thành phần quan trọng của ứng dụng. Nó không chỉ đóng vai trò trung gian trong trao đổi thông tin giữa ứng dụng với người dùng bởi giao diện đồ họa dễ nhìn mà còn là công cụ xử lý message hiệu quả và không thể thiếu cho cơ chế điều phối message của ứng dụng windows.

Bên cạnh đó, cửa sổ giao diện còn làm chức năng nhận diện ứng dụng, là thẻ thông hành cho ứng dụng trong hành trình tồn tại, hoạt động độc lập cũng như phối hợp trao đổi dữ liệu với các ứng dụng khác trong windows.

4.2 LỚP CWnd:

CWnd là lớp đối tượng quản lý cửa sổ của windows. Thông qua các thuộc tính và hành vi của lớp CWnd, MFC cung cấp các dịch vụ cần thiết cho phép tạo lập và khai thác các tính năng của cửa sổ windows một cách dễ dàng.

- HWND *m_hWnd*: Thuộc tính lưu handle của cửa sổ.
- CWnd(); Hành vi tạo lập đối tượng cửa sổ.
- virtual BOOL Create (
LPCTSTR *lpzClassName*, // Tên đăng ký của lớp cửa sổ
LPCTSTR *lpzWindowName*, // Tên cửa sổ.
DWORD *dwStyle*, // Các thông số về dạng cửa sổ
const RECT& *rect*, // Qui định vị trí, kích thước cửa sổ
CWnd* *pParentWnd*, // Con trỏ đối tượng cửa sổ cha
UINT *nID*, // Số hiệu cửa sổ
CCreateContext* *pContex* = NULL
); Khởi tạo thông số cho cửa sổ quản lý bởi đối tượng.
Tham số *dwStyle* qui định đặc điểm và kiểu dáng cửa sổ. Giá trị dùng cho tham số này có thể kết hợp một số trong các giá trị sau:
WS_POPUP : Cửa sổ được tạo là cửa sổ chính.
WS_CHILD : Cửa sổ được tạo là cửa sổ con.
WS_TABSTOP : Cửa sổ con, chuyển được bằng phím tab.
WS_OVERLAPPED : Cửa sổ chính.
WS_SYSMENU : Cửa sổ có hộp menu hệ thống.
WS_BORDER : Cửa sổ có viền.
WS_CAPTION : Cửa sổ có tiêu đề (caption)
WS_DISABLED : Cửa sổ bị cấm.
WS_DLGFRAME : Cửa sổ có viền đậm kiểu hộp thoại,

WS_HSCROLL : Cửa sổ có thanh trượt ngang ở biên.
WS_VSCROLL : Cửa sổ có thanh trượt dọc ở biên.
WS_MAXIMIZEBOX : Có hộp phóng to trên caption của cửa sổ.

WS_MINIMIZEBOX : Có hộp thu nhỏ trên caption của cửa sổ.
WS_THICKFRAME : Viền cho phép thay đổi kích thước cửa sổ.

WS_VISIBLE : Cửa sổ nhìn thấy được (hiển thị).

Ví dụ: WS_POPUP | WS_CAPTION : *Cửa sổ chính có tiêu đề.*

lpzClassName là một tên đã đăng ký cho lớp cửa sổ. Ngoài các tên mà windows đã đăng ký như STATIC, BUTTON, EDIT, ... (chương 8), ta có thể đăng ký tên lớp cửa sổ riêng của mình một cách tùy ý. Việc đăng ký tên lớp cửa sổ có thể thực hiện bằng một trong hai cách sau:

- *Đăng ký trực tiếp:*

```
LPCTSTR AFXAPI AfxRegisterWndClass (  
    UINT nClassStyle, // Thông số dạng của cửa sổ  
    HCURSOR hCursor = 0, // Cursor hiển thị trong cửa sổ  
    HBRUSH hbrBackground = 0, // Brush dùng tô nền cửa sổ  
    HICON hIcon = 0 // Icon trên tiêu đề của cửa sổ  
);
```

Trả về chuỗi tên lớp cửa sổ được đăng ký. Các lần đăng ký tên lớp cửa sổ có tham số giống nhau sẽ nhận được một tên duy nhất.

Tham số *nClassStyle* có thể kết hợp từ các giá trị sau:

CS_HREDRAW : Cửa sổ được vẽ lại khi chiều rộng thay đổi.
CS_VREDRAW : Cửa sổ được vẽ lại khi chiều cao thay đổi.
CS_NOCLOSE : Cấm hộp đóng [] trên tiêu đề của cửa sổ.

Ví dụ: Đăng ký lớp cửa sổ có nền màu xanh dương.

```
CString myClassName;  
CBrush myBrush;  
myBrush.CreateSolidBrush( RGB ( 0, 0, 255 ) );  
myClassName = AfxRegisterWndClass (  
    CS_VREDRAW | CS_HREDRAW,  
    NULL, myBrush, NULL );
```

- *Đăng ký qua cấu trúc chứa các thông số:*

```
BOOL AFXAPI AfxRegisterClass( WNDCLASS* lpWndClass );
```

Hàm trả về giá trị TRUE nếu tác vụ đăng ký thành công. Thực hiện đăng ký theo cách này tránh được sự dùng chung tên lớp cửa sổ ở hai ứng dụng khác nhau khi hai ứng dụng này tình cờ đăng ký các tên lớp cửa sổ giống nhau về thông số.

lpWndClass là con trỏ chỉ đến biến có kiểu cấu trúc **WNDCLASS**.

```
typedef struct _WNDCLASS {
    UINT style;           // Dạng của lớp đăng ký
    WNDPROC lpfnWndProc;  // Con trỏ hàm WindowProc của
                          // cửa sổ. Có thể lấy hàm do
    MFC
                          // khai báo sẵn: AfxWndProc
    int cbClsExtra;       // Dành riêng của hệ thống
    int cbWndExtra;       // Dành riêng của hệ thống
    HINSTANCE hInstance;  // Instance handle của ứng
    dụng
    HICON hIcon;          // Handle của icon
    HCURSOR hCursor;      // Handle của cursor
    HBRUSH hbrBackground; // Handle của brush vẽ nền
    LPCTSTR lpszMenuName; // Chuỗi tên menu trong
    resource
    LPCTSTR lpszClassName; // Tên lớp cửa sổ đăng ký
} WNDCLASS;
```

- **BOOL CreateEx** (
 - DWORD *dwExStyle*, // Các thông số dạng mở rộng
 - LPCTSTR *lpszClassName*, // Tên lớp
 - LPCTSTR *lpszWindowName*, // Tên cửa sổ
 - DWORD *dwStyle*, // Dạng cửa sổ
 - int *x*, int *y*, // Tọa độ góc trái trên của cửa sổ
 - int *nWidth*, int *nHeight*, // Chiều rộng và cao của cửa sổ
 - HWND *hWndParent*, // Handle của cửa sổ cha
 - HMENU *nIDorHMenu*, // Handle của menu gắn với cửa sổ
 - LPVOID *lpParam* = NULL
); Khởi tạo cửa sổ với việc sử dụng các thông số mở rộng về dạng. Tham số *dwExStyle* qui định dạng mở rộng của cửa sổ có thể kết hợp từ các giá trị sau:
 - WS_EX_TOPMOST : Cửa sổ không bị che khuất.
 - WS_EX_TOOLWINDOW : Cửa sổ không hiển thị trên taskbar.
 - WS_EX_TRANSPARENT : Cửa sổ có nền trong suốt.
 - WS_EX_CLIENTEDGE : Cửa sổ có gờ quanh vùng client.
- **virtual BOOL PreCreateWindow**(CREATESTRUCT& *cs*); Hành vi được thực hiện trước khi windows khởi tạo thông số cho cửa sổ. Tham biến *cs* kiểu **CREATESTRUCT** chứa thông số khởi tạo.

```
typedef struct tagCREATESTRUCT {
```

```
LPVOID lpCreateParams; // Con trỏ vùng chứa thông số của
sổ
HANDLE hInstance;      // Handle của ứng dụng
HMENU hMenu;           // Handle của menu gắn với cửa sổ
HWND hWndParent;       // Handle của cửa sổ cha
int cy; int cx;         // Chiều rộng và cao của cửa sổ
int y; int x;           // Tọa độ góc trái trên của cửa sổ
LONG style;            // Thông số ấn định dạng cửa sổ
LPCSTR lpszName;        // Tên cửa sổ được tạo
LPCSTR lpszClass;       // Tên lớp cửa sổ dùng cho cửa sổ
DWORD dwExStyle;       // Thông số ấn định dạng mở rộng
} CREATESTRUCT;
```

☞ Trong các lớp kế thừa CWnd, hành vi này được dùng để can thiệp cài đặt các ấn định riêng trên cấu trúc thông số **cs** của cửa sổ.

- **BOOL EnableWindow**(BOOL *bEnable* = TRUE); Cho phép hoặc cấm hoạt động của cửa sổ.
- **BOOL ShowWindow**(int *nCmdShow*); Ấn định trạng thái hiển thị của cửa sổ trên màn hình. Giá trị cho tham số *nCmdShow* có thể là:
 - SW_HIDE : Dấu cửa sổ
 - SW_MINIMIZE : Thu nhỏ cửa sổ
 - SW_RESTORE : Đưa cửa sổ về trạng thái trước đó
 - SW_SHOW : Hiển thị cửa sổ
 - SW_SHOWNA : Hiển thị nhưng không kích hoạt cửa sổ
 - SW_SHOWMAXIMIZED : Hiển thị và phóng to cửa sổ
 - SW_SHOWMINIMIZED : Hiển thị và thu nhỏ cửa sổ
- **BOOL SetWindowPos** (
 - const CWnd* *pWndInsertAfter*, // Con trỏ cửa sổ làm mốc
 - int *x*, int *y*, // Tọa độ góc trái trên của cửa sổ
 - int *cx*, int *cy*, // Kích thước cửa sổ
 - UINT *nFlags* // Thông số trạng thái
); Ấn định vị trí cửa sổ trên màn hình. Giá trị *pWndInsertAfter* qui định vị trí đặt cửa sổ theo chiều thứ 3 (z-order). Giá trị này có thể như sau:
 - wndBottom : Cửa sổ được đặt dưới mọi cửa sổ.
 - wndTop : Cửa sổ được đặt trên các cửa sổ thông thường.
 - wndTopMost : Cửa sổ được đặt trên mọi cửa sổ.
 Tham số *nFlags* qui định trạng thái mới của cửa sổ:
 - SWP_SHOWWINDOW : Hiển thị cửa sổ.

- SWP_DRAWFRAME : Vẽ lại frame của cửa sổ.
- SWP_NOREDRAW : Không cập nhật lại thông tin của cửa sổ.
- void **MoveWindow** (
 - int x, int y, // Tọa độ mới cho góc trái trên
 - int nWidth, int nHeight, // Chiều rộng và chiều cao của cửa sổ
 - BOOL bRepaint= TRUE // Yêu cầu windows vẽ lại cửa sổ
); Thay đổi vị trí và kích thước của cửa sổ.
- void **GetWindowRect**(LPRECT lpRect); Lấy thông tin tọa độ, kích thước của cửa sổ, lpRect chỉ đến biến kiểu RECT chứa kết quả.
- void **GetClientRect**(LPRECT lpRect); Lấy thông tin tọa độ, kích thước vùng client của cửa sổ, lpRect chỉ đến biến RECT chứa kết quả.
- int **GetWindowRgn**(HRGN hRgn); Xác định vùng hiển thị của cửa sổ.
- int **SetWindowRgn** (
 - HRGN hRgn, // Handle của region quản lý vùng ẩn định
 - BOOL bRedraw // Vẽ lại cửa sổ (TRUE) hay không (FALSE)
); Ẩn định vùng hiển thị của cửa sổ theo dạng của region.
- void **GetWindowText**(CString rString); Lấy nội dung chuỗi tiêu đề của cửa sổ và lưu vào biến đối tượng chuỗi rString.
- int **GetWindowTextLength**(); Trả về chiều dài của chuỗi tiêu đề.
- void **ClientToScreen**(LPPOINT lpPoint / LPRECT lpRect); Chuyển tọa độ điểm hay vùng hình chữ nhật trong client của cửa sổ sang hệ trục tọa độ của màn hình.
- void **ScreenToClient**(LPPOINT lpPoint / LPRECT lpRect); Chuyển tọa độ điểm hay vùng hình chữ nhật trên màn hình sang hệ trục tọa độ của vùng client trong cửa sổ.
- HICON **GetIcon**(BOOL bBigIcon); Trả về handle của icon mà cửa sổ đang sử dụng. Giá trị tham số bBigIcon có ý nghĩa như sau:
 - TRUE : Handle của icon hiển thị trên taskbar (big Icon)
 - FALSE : Handle của icon hiển thị trên caption (small Icon)
- HICON **SetIcon** (
 - HICON hIcon, // handle của icon
 - BOOL bBigIcon // TRUE (đặt bigIcon) , FALSE (đặt smallIcon)
); Đặt icon mới cho cửa sổ.
- static CWnd* PASCAL **GetFocus**(); Trả về con trỏ chỉ đến đối tượng CWnd đang được phép nhận thông tin nhập từ bàn phím.
- CWnd* **SetFocus**(); Kích hoạt cửa sổ. Hàm trả về con trỏ của đối tượng CWnd đã được kích hoạt trước đó.
- CFont* **GetFont**(); Trả về đối tượng font chữ của cửa sổ.
- void **SetFont** (
 - CFont* pFont, // Con trỏ đến đối tượng font chữ

- BOOL bRedraw= TRUE // Vẽ lại cửa sổ sau tác vụ đặt font ?
); Ẩn định font chữ cho cửa sổ.
- CMenu* **GetMenu**(); Trả về con trỏ đối tượng menu gắn với cửa sổ.
- BOOL **SetMenu**(CMenu* pMenu); Gắn menu cho cửa sổ.
- CWnd* **GetParent**(); Trả về con trỏ đến đối tượng của cửa sổ cha.
- int **GetScrollPos**(int nBar); Trả về vị trí hiện hành của nút cuộn trên thanh cuộn. nBar chứa số hiệu thanh cuộn quan tâm. nBar có thể là:
 - SB_HORZ : Thanh cuộn ngang.
 - SB_VERT : Thanh cuộn dọc.
- int **SetScrollPos** (
 - int nBar, // Thanh cuộn được chọn
 - int nPos, // Vị trí đặt
 - BOOL bRedraw= TRUE // Vẽ lại thanh cuộn sau tác vụ đặt
); Đặt vị trí nút cuộn cho thanh cuộn tương ứng.
- UINT **SetTimer** (
 - UINT nIDEvent, // Số hiệu của timer, phân biệt duy nhất
 - UINT nElapse, // Chu kỳ timer (tính bằng mili-second)
 - NULL // Sử dụng hành vi **OnTimer** xử lý timer
); Đặt biến cố định thời (timer) cho cửa sổ quản lý bởi đối tượng. Mỗi khi hết một chu kỳ của timer, hệ thống gửi WM_TIMER kèm theo số hiệu của timer đó đến cho cửa sổ.
- BOOL **KillTimer**(int nIDEvent); Hủy bỏ timer có số hiệu nIDEvent.
- afx_msg void **OnTimer**(UINT nIDEvent); Hành vi xử lý WM_TIMER của cửa sổ. Tham số nIDEvent chứa số hiệu của timer liên quan.
- void **Invalidate**(BOOL bErase = TRUE); Kích hoạt cơ chế vẽ lại vùng client của cửa sổ. Nếu bErase = FALSE, windows không thực hiện xóa thông tin trong vùng cần vẽ lại.
- void **InvalidateRect**(LPCRECT lpRect, BOOL bErase = TRUE); Kích hoạt cơ chế vẽ lại một vùng trong client của cửa sổ. Thông tin về vị trí và kích thước của vùng cần vẽ lại được lưu trong biến kiểu RECT chỉ bởi lpRect. Tham số bErase có ý nghĩa như **Invalidate**.
- int **MessageBox** (
 - LPCTSTR lpszText, // Nội dung thông báo
 - LPCTSTR lpszCaption= NULL // Tiêu đề hộp thông báo
 - UINT nType= MB_OK // Dạng hộp thông báo
); Hiển thị hộp thông báo và trả về số hiệu của nút được chọn.
- LRESULT **SendMessage** (
 - UINT message, // Số hiệu message
 - WPARAM wParam= 0, // Tham số kiểu WORD
 - LPARAM lParam= 0 // và kiểu LONG kèm theo message

); Gửi message và tham số kèm theo đến hàm **WindowProc** của cửa sổ quản lý bởi đối tượng, và chờ đến khi hàm **WindowProc** xử lý xong.

- **BOOL PostMessage(** *UINT message*,
 WPARAM wParam = 0, LPARAM lParam=0
); Đặt message và các tham số kèm theo vào message queue của ứng dụng. Hành vi kết thúc mà không chờ message đó được xử lý.
- **afx_msg void OnSize(** *UINT nType*, *int cx*, *int cy*); Hành vi xử lý **WM_SIZE**, message do windows gửi đến cửa sổ khi một tác vụ thay đổi kích thước của sổ hoàn tất. *cx*, *cy* chứa kích thước mới của cửa sổ.
- **afx_msg int OnCreate(** *LPCREATESTRUCT lpCreateStruct*); Hành vi xử lý **WM_CREATE**, message do windows gửi đến cửa sổ khi tác vụ khởi tạo thông số cho cửa sổ được thực hiện xong.
- **afx_msg void OnClose(**); Hành vi xử lý **WM_CLOSE**, message do windows gửi đến cửa sổ khi tác vụ đóng cửa sổ đang xảy ra.
- **afx_msg void OnDestroy(**); Hành vi xử lý **WM_DESTROY**, message do windows gửi đến cửa sổ khi tác vụ hủy bỏ cửa sổ đang xảy ra.
- **afx_msg void OnKeyDown(**
 UINT nChar, // Mã phím
 UINT nRepCnt, // Số lần gõ phím
 UINT nFlags // Trạng thái các phím kèm theo
); Hành vi xử lý **WM_KEYDOWN**, message do windows gửi đến cửa sổ khi cửa sổ đang được kích hoạt, đồng thời có phím vừa được ấn xuống mà không có sự sử dụng phím **Alt** kèm theo.
- **afx_msg void OnKeyUp(** // Các tham số tương tự như trên
 UINT nChar, *UINT nRepCnt*, *UINT nFlags*
); Hành vi xử lý **WM_KEYUP**. Một cách tương tự **WM_KEYDOWN**.
- **afx_msg void OnChar(**
 UINT nChar, // Mã ASCII
 UINT nRepCnt, // Số lần gõ
 UINT nFlags // Trạng thái các phím kèm theo
); Hành vi xử lý **WM_CHAR**, message do windows gửi đến cửa sổ khi một phím ký tự được gõ.
- **afx_msg void OnLButtonDblClk(**
 UINT nFlags, // Chứa giá trị phím được nhấn kèm
 CPoint point // Vị trí double-click chuột
); Hành vi xử lý **WM_LBUTTONDOWNDBLCLK**, message do windows gửi đến cửa sổ khi người dùng double-click vào nút chuột trái.
Tham số *nFlag* có thể là kết hợp của các giá trị sau:
 MK_CONTROL : Phím CTRL được nhấn kèm theo

MK_SHIFT : Phím SHIFT được nhấn kèm theo

- **afx_msg void OnLButtonDown(** *UINT nFlags*, *CPoint point*); Hành vi xử lý **WM_LBUTTONDOWN**, message do windows gửi đến cửa sổ khi người dùng ấn nút chuột trái. Các thông tin như trên.
- **afx_msg void OnLButtonUp(** *UINT nFlags*, *CPoint point*); Hành vi xử lý **WM_LBUTTONUP**, message do windows gửi đến cửa sổ khi người dùng nhả nút chuột trái. Các thông tin như trên.
- Một cách tương tự cho các hành vi xử lý message của nút chuột phải.
- **afx_msg void OnMouseMove(** *UINT nFlags*, *CPoint point*); Hành vi xử lý **WM_MOUSEMOVE**, message do windows gửi đến cửa sổ khi người dùng di chuyển chuột trong cửa sổ. Các thông tin như trên.
- **int GetDlgCtrlID(**); Trả về số hiệu của đối tượng cửa sổ con.
- **afx_msg void OnPaint(**); Hành vi xử lý **WM_PAINT**, message do windows gửi đến cửa sổ khi hệ thống hoặc ứng dụng có nhu cầu trang trí lại một phần hay toàn bộ giao diện của cửa sổ.

Công việc thông thường của **OnPaint** là vẽ lại các nội dung cần duy trì trên bề mặt giao diện của cửa sổ. Để thực hiện việc này, **OnPaint** sử dụng một đối tượng CDC và dùng nó cho các thao tác đồ họa cần thiết nhằm hoàn thành yêu cầu nói trên.

Bố cục xử lý thông thường của hành vi **OnPaint** như sau:

PAINTSTRUCT ps ;	// Biến chứa thông tin trang trí
CDC* pDC = BeginPaint(&ps);	// Lấy DC của giao diện cửa sổ
.	// Xử lý trang trí giao diện đồ họa
EndPaint(&ps);	// Chấm dứt.

- **afx_msg void OnHScroll(**
 UINT nSBCode, // Số hiệu ghi nhận đặc điểm tác động
 UINT nPos, // Vị trí nút cuộn / nút trượt trên mục
 CScrollBar pScrollBar* // Con trỏ đối tượng quản lý mục
); Hành vi xử lý **WM_HSCROLL**, message do windows gửi đến cửa sổ khi có một mục là thanh cuộn hay thanh trượt đặt ngang (horizontal scrollbar hoặc horizontal sliderCtrl) trong cửa sổ bị tác động.
nSBCode ghi nhận đặc điểm tác động lên nút cuộn / trượt như sau:
 SB_LEFT : Giảm nút về vị trí thấp nhất
 SB_ENDSCROLL : Chấm dứt tác vụ chuyển nút
 SB_LINELEFT : Giảm nút một vị trí
 SB_LINERIGHT : Tăng nút một vị trí
 SB_PAGELF : Giảm nút một đoạn

SB_PAGERIGHT : Tăng nút một đoạn
 SB_RIGHT : Tăng nút đến vị trí cao nhất
 SB_THUMBPOSITION: Chuyển nút bằng chuột
 SB_THUMBTRACK : Đang chuyển nút bằng chuột .

nPos được sử dụng trong các tác vụ định vị nút tuyệt đối ().

☞ Dùng hàm vi **GetDlgCtrlID** của đối tượng chỉ bởi *pScrollbar* để xác định số hiệu của mục phát sinh message WM_HSCROLL. Đây là cơ sở giúp phân biệt mục này với các mục khác trong cùng cửa sổ giao diện nhằm lựa chọn xử lý thích hợp cho WM_HSCROLL.

- **afx_msg void OnVScroll (**
 UINT *nSBCode*, // Số hiệu ghi nhận đặc điểm tác động
 UINT *nPos*, // Vị trí nút cuộn / nút trượt trên mục
 CScrollbar* *pScrollbar*// Con trỏ đối tượng quản lý mục
); Hàm vi xử lý WM_VSCROLL, message do windows gửi đến cửa sổ khi có một mục là thanh cuộn hay thanh trượt đặt thẳng đứng (vertical scrollbar hoặc vertical sliderCtrl) trong cửa sổ bị tác động.
 ☞ Xử lý của hàm vi này được cài đặt tương tự hàm vi **OnHScroll**.
- **afx_msg BOOL OnSetCursor (**
 CWnd* *pWnd*, // Con trỏ đến đối tượng cửa sổ chứa cursor
 UINT *nHitTest*, // Thông tin về vị trí cursor
 UINT *message* // Chứa các số hiệu message có liên quan đến
 // trạng thái hiện thời của các nút con chuột
); Hàm vi xử lý WM_SETCURSOR, message do windows gửi đến cửa sổ khi windows cần ấn định lại hình dạng cursor cho phù hợp với vị trí hiện thời của nó trên cửa sổ.
nHitTest chứa thông tin vị trí hiện thời của cursor:
 HTBORDER : Cursor hiện nằm trên biên cửa sổ
 HTCLIENT : Cursor hiện nằm trong vùng client
 HTCAPTION : Cursor hiện nằm trên tiêu đề của cửa sổ
- **virtual LRESULT WindowProc(** UINT *message*,
 WPARAM *wParam*, LPARAM *lParam*);
 Hàm vi xử lý các message gửi đến cửa sổ. Mặc nhiên, hàm vi này dựa vào bảng **MessageMap** để chuyển message đến hàm vi xử lý message tương ứng của đối tượng quản lý cửa sổ.


4.3 SỬ DỤNG ĐỐI TƯỢNG CWnd:

4.3.1 Sử dụng CWnd làm giao diện chính của ứng dụng:

- Tạo dự án VD02 như dự án VD01. Thực hiện các bổ sung sau:
- Tạo icon có số hiệu là IDC_MAINFRAME. Tham khảo (2.8).
 - Tạo cursor có số hiệu là IDC_MAINFRAME:
 - Tạo mới cursor: Thực hiện tương tự như tạo mới icon, (2.8).

- Đặt điểm chi (hotpot) của cursor: Trong màn hình thiết kế cursor:



- Click chọn biểu tượng  trên thanh công cụ.
 - Click tại vị trí hotpot của cursor trên màn hình thiết kế.
- Dùng đối tượng CWnd làm cửa sổ giao diện chính của ứng dụng: Được thực hiện bởi hàm vi **InitInstance** (xem 2.4) của đối tượng CEmpApp quản lý tiểu trình chính. Kế thừa hàm vi này từ CWinApp cho lớp CEmpApp (xem 2.7). Nội dung cài đặt của hàm vi như sau:

```
CWnd* main = new CWnd(); // Con trỏ đối tượng CWnd.
HICON myIcon; // Khai báo biến quản lý
HCURSOR myCursor; // handle của cursor và icon
CBrush myBrush;

// Nạp cursor và icon từ resource vào bộ nhớ.
myIcon = LoadIcon ( IDR_MAINFRAME );
myCursor = LoadCursor ( IDR_MAINFRAME );
// Tạo brush tô nền cửa sổ với màu RGB(190, 190, 0)
myBrush.CreateSolidBrush (RGB(190, 190, 0) );

// Khởi tạo thông số cho đối tượng cửa sổ main
main->CreateEx( WS_EX_TOPMOST,
               AfxRegisterWndClass(
                   CS_HREDRAW|CS_VREDRAW,
                               myCursor, myBrush, myIcon),
               "Emp.Example 2",
               WS_SYSMENU | WS_VISIBLE | WS_MINIMIZEBOX,
               100, 100, 300, 200, NULL, NULL );
// Dùng main làm cửa sổ giao diện chính
m_pMainWnd = main;
main->ShowWindow ( SW_SHOW ); // Kích hoạt cửa sổ main
```

☞ Khi đối tượng cửa sổ main ngừng hoạt động thì ứng dụng cũng kết thúc.

☞ **Xem VD02.** Cửa sổ main với ExStyle là WS_EX_TOPMOST có thể nổi trên mọi cửa sổ khác ngay cả khi nó không phải là cửa sổ kích hoạt. Với ExStyle là WS_EX_TOOLWINDOW, cửa sổ sẽ không hiển thị trên taskbar.

4.3.2 Ứng dụng chỉ chạy một bản (instance) tại mỗi thời điểm:

Để ứng dụng chỉ được thực hiện với 1 bản duy nhất, ta cài đặt cơ chế đánh dấu và kiểm tra. Trong chương trình của ứng dụng, ta qui ước đăng

ký và sử dụng một tên duy nhất cho cửa sổ chính. Khi chương trình được thực hiện, nó kiểm tra xem tên đó đã được đăng ký chưa thông qua hàm sau:

```
HANDLE CreateMutex( NULL, FALSE, LPCTSTR Tên );
```

Hàm trả về giá trị ERROR_ALREADY_EXISTS nếu *Tên* đã được đăng ký. Trong trường hợp này ta có thể khẳng định một instance của ứng dụng đã được thực hiện, chương trình kết thúc để chỉ cho phép một instance duy nhất.

☞ Hãy cài đặt cơ chế này cho ứng dụng VD02 (Tham khảo VD03).

THỰC HÀNH:

1. Viết ứng dụng windows chỉ cho phép thực hiện tối đa hai bản (instance).
2. Cài đặt hành vi *PreCreateWindow* cho lớp kế thừa CWnd của ứng dụng để cửa sổ giao diện luôn có kích thước 100100 và tiêu đề là "Hello !" bất chấp giá trị kích thước và tiêu đề dùng cho khởi tạo thông số của đối tượng cửa sổ.

Xử lý message

5.1 LỚP XỬ LÝ MESSAGE CCmdTarget:

Windows là môi trường mà phần lớn giao tác giữa các bộ phận dựa trên cơ chế gửi-nhận message. Việc tạo ra đối tượng có khả năng xử lý và điều phối messages là rất cần thiết không chỉ đối với hệ thống mà với cả ứng dụng. Trên quan điểm đó, MFC cung cấp lớp đối tượng CCmdTarget phục vụ xử lý và điều phối messages trong phạm vi ứng dụng, giữa ứng dụng với hệ thống và với các ứng dụng khác. Các hành vi đặc trưng của lớp như sau:

- void **BeginWaitCursor**(); Hiển thị cursor chờ xử lý (đồng hồ cát).
- void **EndWaitCursor**(); Chấm dứt hiển thị cursor chờ xử lý.
- Định hướng xử lý message: Cơ chế định hướng xử lý message do MFC cung cấp cho phép bổ sung mục xử lý message cho các lớp đối tượng kế thừa lớp CCmdTarget. Các macro giúp cài đặt cơ chế này như sau:
 - **DECLARE_MESSAGE_MAP**(): Ấn định đặc tính xử lý message cho lớp đối tượng xử lý message thông qua các cài đặt bổ sung sau:
 - Thuộc tính **private** kiểu cấu trúc mảng chứa các phần tử có kiểu AFX_MSGMAP_ENTRY. Mỗi phần tử của mảng được dùng lưu trữ một mục xử lý message mà lớp kế thừa khai báo bổ sung.
 - Thuộc tính **protected** kiểu cấu trúc AFX_MSGMAP với tên là **MessageMap** chỉ đến bảng các mục xử lý message nói trên.
 - Hành vi **protected**: virtual AFX_MSGMAP GetMessageMap(); trả về địa chỉ của bảng **MessageMap** chứa các mục xử lý.

DECLARE_MESSAGE_MAP được đặt cuối phần khai báo lớp:

```
class MyClass : public CCmdTarget {      // Tập tin .H của lớp
...                                     // Các nội dung khai báo của lớp
    DECLARE_MESSAGE_MAP()
};
```

- **BEGIN_MESSAGE_MAP**(Tên_lớp_kế_thừa, Tên_lớp_cơ_sở): Bắt đầu nội dung khai báo các mục xử lý của bảng **MessageMap**.
- **END_MESSAGE_MAP**(): Kết thúc khai báo bảng **MessageMap**. Toàn bộ nội dung khai báo của bảng **MessageMap** được đặt trong tập tin cài đặt (.CPP) của lớp, nên đặt đầu tập tin để tiện theo dõi.
- virtual BOOL **OnCmdMsg**(

```
UINT nID,           // Số hiệu command message
int nCode,
void* pExtra,
AFX_CMDHANDLERINFO* pHandlerInfo
```

); Điều phối command message. Nếu bản thân đối tượng là cửa sổ giao diện chính thì các WM_COMMAND được ưu tiên gọi đến cho đối tượng. Thông qua hành vi này, đối tượng có thể điều phối command message cho các đối tượng khác (ứng dụng, các control, view...). Lưu ý là đối tượng được điều phối có thể không có chức năng xử lý command message gửi đến, do đó cần kiểm tra kết quả của hành vi **OnCmdMsg** trên đối tượng được điều phối. Bố cục xử lý điều phối như sau:

```
if (Đối_tượng->OnCmdMsg(...) != 0) {
    return;      // Đối tượng được điều phối đã xử lý message
}
...             // Chủ thể phải xử lý message
```

5.2 KHAI BÁO MỤC XỬ LÝ MESSAGE TRONG MESSAGE MAP:

Mục xử lý message trong bảng **MessageMap** cho phép ấn định một xử lý duy nhất cho một message. Các loại message khác nhau có kiểu mục xử lý message khác nhau. Các kiểu mục xử lý message phổ biến như sau:

- Các message của hệ thống, được biểu diễn bởi các hằng số bắt đầu bằng WM_*, mục xử lý message tương ứng có dạng ON_WM_*().

Ví dụ: WM_PAINT → ON_WM_PAINT()
WM_SIZE → ON_WM_SIZE()

- Các message của người dùng: Số hiệu message được chọn tùy ý trong đoạn WM_USER ÷ WM_USER+0x7FFF. Mục xử lý message cho các message của người dùng có dạng như sau:

ON_MESSAGE(userMessageID , UserFuncName)

Trong đó:

- *userMessageID* : Số hiệu message do người dùng chọn trước
- *UserFuncName* : Hàm xử lý message, có khai báo như sau:

```
afx_msg LRESULT UserFuncName (
    WPARAM wParam,    // Tham số kiểu WORD và
    LPARAM lParam      // Tham số kiểu LONG kèm theo message
);
```

- Các message có đăng ký của người dùng: Ngoài các message tự định nghĩa và sử dụng theo qui ước trong một ứng dụng, windows cho phép ứng dụng đăng ký message để message đó có thể sử dụng trên nhiều ứng dụng khác nhau. Việc đăng ký được thực hiện thông qua hàm sau:

UINT RegisterWindowMessage (LPCSTR *Chuỗi_tên_message*);

Hàm trả về số hiệu đăng ký được của message. Giá trị này nằm trong đoạn 0xC000÷0xFFFF. Các ứng dụng đang chạy trên một hệ thống có thể chia sẻ message dùng riêng với điều kiện chúng phải thực hiện thao tác đăng ký cùng một chuỗi tên message để lấy số hiệu message. Mục xử lý message cho các message có đăng ký của người có dạng:

ON_REGISTERED_MESSAGE(*UserRegMessageID*, *UserFuncName*)

Sau đây là một ví dụ:

```
// Đăng ký message với tên là "MY_MESS"
const UINT myMess = RegisterWindowMessage("MY_MESS");
// Khai báo mục xử lý cho message được đăng ký
BEGIN_MESSAGE_MAP ( CMyWnd, CMyBasedWndClass )
//{{AFX_MSG_MAP ( CMyWnd )
ON_REGISTERED_MESSAGE ( myMess, myFunc )
// ...
//}}AFX_MSG_MAP
END_MESSAGE_MAP ( )
```

- Message WM_COMMAND: Khi WM_COMMAND được gửi đến đối tượng xử lý message thì tham số **wParam** kèm theo chứa số hiệu (CommandID) của đối tượng phát sinh message. Mục xử lý message WM_COMMAND ấn định xử lý tương ứng, và có dạng như sau:

ON_COMMAND (*CommandID*, *FunctionName*)

☞ Có thể cài đặt xử lý điều khiển đối với đối tượng làm phát sinh WM_COMMAND thông qua mục xử lý điều khiển message như sau:

ON_UPDATE_COMMAND_UI(*CommandID*, *PreFunctionName*)

PreFunctionName là hành vi thực hiện xử lý điều khiển trên đối tượng phát sinh WM_COMMAND, tham số nhận được là giá trị con trỏ đến đối tượng CCmdUI*. Hành vi **Enable** (BOOL *isEnabled*) của đối tượng này được dùng để cấm hoặc cho phép hoạt động đối với đối tượng phát sinh WM_COMMAND. Xử lý của hành vi này có thể là:

```
void [ClassName::]PreFunctionName (CCmdUI* pCmdUI ) {
    pCmdUI->Enabled (FALSE );    // Cấm đối tượng hoạt động
}
```

- Các message do đối tượng con (controls) gửi đến cửa sổ cha: Tham số **wParam** chứa số hiệu control, giá trị WORD cao của tham số **lParam** chứa thông tin về trạng thái control ở thời điểm gửi message đến cửa sổ cha (ví dụ **BN_CLICKED** là một trạng thái của button control,...). Mục xử lý message cho message gửi từ control có dạng như sau:

ON_CONTROL (*Trạng_thái_control*, *Số_hiệu_Control*, *Hàm_xử_ly*)

Ví dụ: Ta có ví dụ minh họa định hướng xử lý message như sau:

* Khai báo lớp (tệp tin .H):

```
class CMyClass: public CBasedWnd {
public: CMyClass();
    void myProc(void);
    void mySerach();
    void myWork();
    void PremyWork();
    void OnExit(void);
    DECLARE_MESSAGE_MAP()
}
```

* Phần cài đặt của lớp (tệp tin .CPP):

```
#define MY_MESSAGE WM_USER + 1
static UINT NEAR MY_MESS = RegisterWindowMessage("MY_MESS");
BEGIN_MESSAGE_MAP(CMyClass, CDerivedWnd)
//{{AFX_MSG_MAP(CMyClass)
    ON_WM_PAINT()                // WindowsMessage
    ON_MESSAGE (MY_MESSAGE, myProc) // UserMessage
    ON_REGISTERED_MESSAGE(MY_MESS, OnSearch)
                                    // UserRegMessage
    ON_COMMAND(ID_DO, myWork)     // CommandMessage
                                    // Command Preprocess
    ON_UPDATE_COMMAND_UI(ID_DO, PremyWork)
                                    //Control IDC_EXIT
    ON_CONTROL(BN_CLICKED, IDC_EXIT, OnExit)
//}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

Các mục xử lý message do Classwizard quản lý đặt giữa `//{ {` và `//}`

5.3 CÁC LỚP KẾ THỪA CCmdTarget:

Các lớp đối tượng của MFC kế thừa từ CCmdTarget có chức năng xử lý message là CWnd, CWinApp, CDocument. Ứng dụng có thể dựa trên những lớp này để xây dựng các lớp kế thừa đảm nhận chức năng xử lý message phù hợp với yêu cầu của ứng dụng.

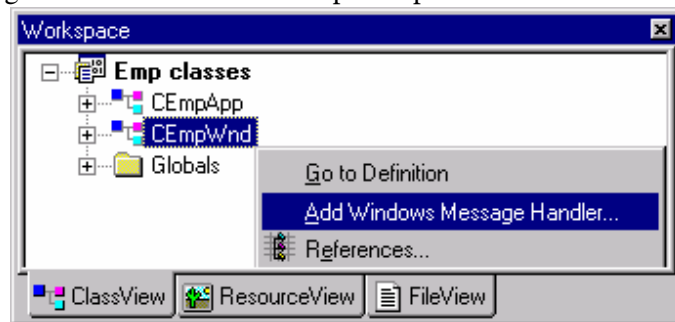
5.4 MESSAGE MAP CỦA LỚP KẾ THỪA CWnd TRONG ỨNG DỤNG:

5.4.1 Cửa sổ của ứng dụng có chức năng hoạt động:

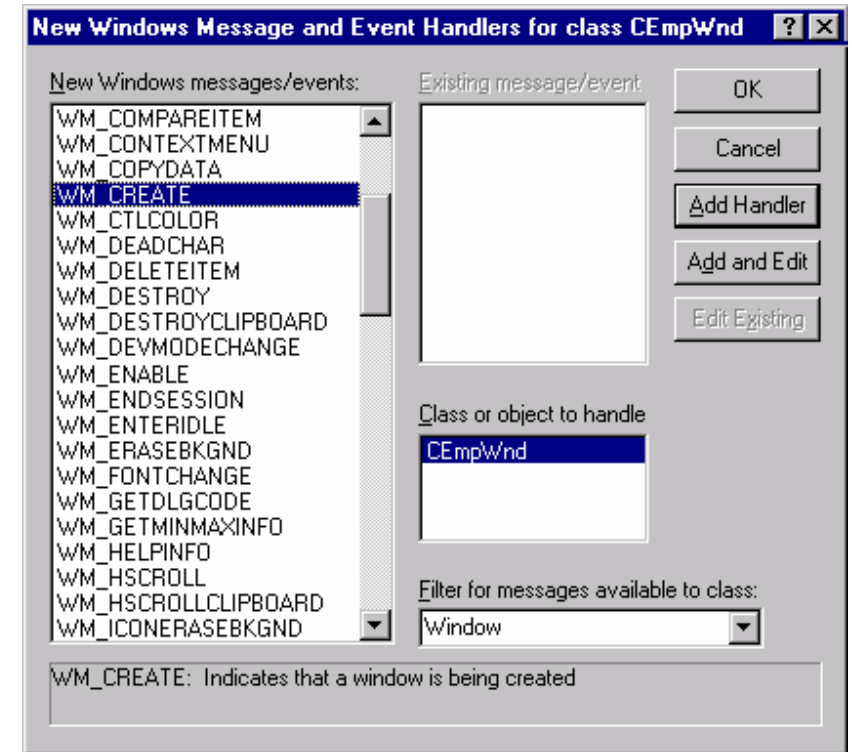
Trong phần này, ta thực hiện ứng dụng với cửa sổ giao diện chính có tiêu đề chứa nội dung chữ chạy theo kiểu bảng chữ điện tử.

Lớp CWnd của MFC không cung cấp tiện ích này. Chúng ta cần xây dựng lớp cửa sổ mới với những khả năng phù hợp; có các chức năng như CWnd để làm giao diện, đồng thời có khả năng tự thay đổi nội dung tiêu đề (caption) theo thời gian (timer). Lớp cửa sổ này kế thừa từ lớp CWnd, tự cài đặt timer (SetTimer) khi bắt đầu (OnCreate) hoạt động, xử lý thay đổi nội dung tiêu đề ở mỗi chu kỳ Timer (OnTimer) và hủy bỏ Timer (KillTimer) khi chấm dứt hoạt động (OnDestroy). Sau đây là các bước thực hiện dự án:

- Tạo dự án VD04 tương tự dự án VD03.
- Bổ sung lớp CEmpWnd (tên lớp cửa sổ mới) kế thừa từ CWnd: Thực hiện như bổ sung lớp CEmpApp trong mục (2.7). Lưu ý trong hộp hội thoại **New Class**: chọn **Class Type** = *MFC Class*; **BaseClass** = *CWnd*.
- Cài đặt các hành vi xử lý message cần thiết cho lớp CEmpWnd trên cơ sở kế thừa từ lớp CWnd của MFC:
 - Hành vi **OnCreate** thực hiện các ấn định cần thiết cho **CEmpWnd** trước khi đi vào hoạt động. Bổ sung và cài đặt hành vi như sau:
 - Trong màn hình **Workspace** của dự án, chọn trang **Class View**.
 - Right-click trên tiêu đề của lớp CEmpWnd:



- Chọn mục **Add Windows Message Handler...** :



- Chọn WM_CREATE. Sau đó chọn **Add and Edit**.
- Hành vi OnCreate với tham số thích hợp được bổ sung vào lớp CEmpWnd, đồng thời mục xử lý ON_WM_CREATE() được đặt vào bảng **MessageMap**. Cài đặt nội dung của OnCreate như sau:

```
int CEmpWnd::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CWnd::OnCreate(lpCreateStruct) == -1) return -1;
    // Đặt timer số hiệu 100, chu kỳ 250 ms
    SetTimer(100, 250, NULL);
    return 0;
}
```

- Hành vi **OnTimer** xử lý WM_TIMER, cho phép xử lý yêu cầu ở mỗi chu kỳ của timer. Việc bổ sung hành vi này được thực hiện tương tự OnCreate. Nội dung cài đặt của hành vi như sau:

```
void CEmpWnd::OnTimer( UINT nIDEvent )
{
}
```

```

if (nIDEvent == 100) {           // Timer do chúng ta cài đặt
    char s[200], ch;
    GetWindowText(s, 200);      // Lấy tiêu đề cửa sổ
    ch = s[0];
    for (UINT i=0; i<strlen(s)-1; i++)
        s[i] = s[i+1];         // Dịch nội dung chuỗi
    s[i] = ch;
    SetWindowText(s);           // Đặt tiêu đề cửa sổ
}
CWnd::OnTimer(nIDEvent);        // Thực hiện hành vi lớp cơ sở
}

```

- Hành vi **OnDestroy** xử lý WM_DESTROY:

```

void CEmpWnd::OnDestroy()
{
    KillTimer (100 );           // Số hiệu timer (TimerID)
    CWnd::OnDestroy();          // Gọi hành vi lớp cơ sở.
}

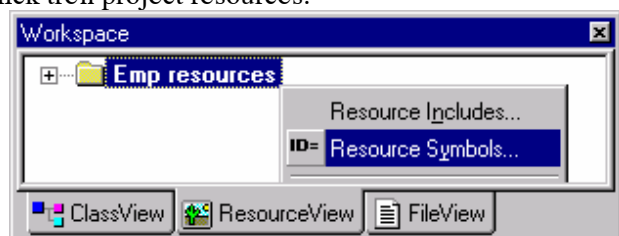
```

- Dùng lớp **CEmpWnd** cho đối tượng cửa sổ chính của ứng dụng: Mở hành vi **InitInstance** của CApp, thực hiện các chỉnh sửa sau:
 - Thực hiện chỉ thị sau ở đầu tập tin chương trình:


```
#include "EmpWnd.h" // Tập tin khai báo của lớp CEmpWnd
```
 - Dùng CEmpWnd làm kiểu cho biến con trỏ đối tượng **main**.
- Biên dịch dự án và chạy thử ứng dụng.

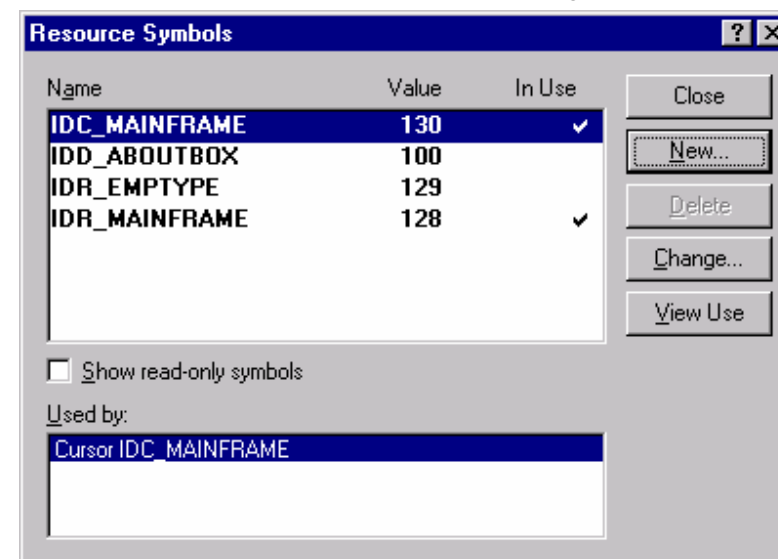
Lưu ý: Dự án VD04 sử dụng 100 làm số hiệu timer. Việc sử dụng giá trị hằng như thế không gợi nhớ và kém linh hoạt trong sử dụng. Ta nên khai báo một tên riêng cho hằng để tránh các hạn chế trên. Cách thực hiện như sau:

- Chọn trang **ResourceView** trong màn hình **Workspace**.
- Right-click trên project resources:

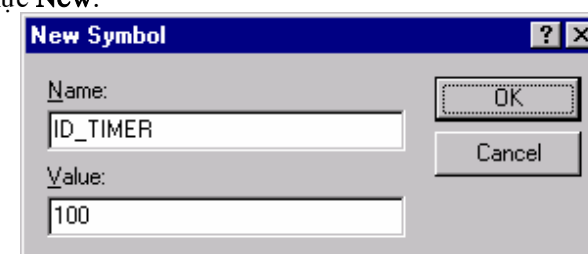


- Chọn **Resource Symbols...**:

Ta nhận được hộp hội thoại Resource Symbols chứa danh sách các giá trị đã khai báo. Có thể thực hiện thêm, xóa các giá trị khai báo này.



- Chọn mục **New**:



- Nhập tên của giá trị khai báo trong hộp **Name**, nhập giá trị khai báo trong hộp **Value**. Sau đó chọn OK
 - Đóng hộp hội thoại **Resource Symbol** để kết thúc.
- Thông tin khai báo lưu trong tập tin resource.h của dự án. Khi đó, trong chương trình, thay vì viết giá trị hằng cụ thể cho số hiệu của Timer (chẳng hạn 100), ta sử dụng tên khai báo của nó (theo ví dụ là ID_TIMER).

5.4.2 WM_PAINT và hành vi OnPaint của CWnd:

Để duy trì thông tin hiển thị trên bề mặt cửa sổ, hệ thống thường xuyên gửi WM_PAINT đến cho cửa sổ mỗi khi có hiện tượng xâm phạm đến nội dung hiển thị của nó. Ứng dụng cũng có thể kích hoạt hệ thống phát sinh message này thông qua một trong các hành vi sau:

- void **Invalidate**(BOOL *bErase* = TRUE); Yêu cầu cập nhật toàn bộ vùng client của cửa sổ. Nếu tham số *bErase* = FALSE thì hệ thống sẽ không tự động xóa nội dung cũ trong cửa sổ.
- void **InvalidateRect** (

LPCRECT *lpRect*, // Con trỏ đến biến kiểu RECT chứa
 // thông tin vùng được cập nhật
 BOOL *bErase* = TRUE // Có ý nghĩa như **Invalidate** ()

); Yêu cầu cập nhật một vùng giới hạn trong client của cửa sổ.

☞ Hành vi **OnPaint** của CWnd dùng xử lý WM_PAINT. Việc sử dụng hành vi này trong các lớp kế thừa CWnd nhằm thực hiện các trang trí riêng theo bố cục ở mục OnPaint trong (4.2). Toàn bộ thao tác xử lý này được MFC thực hiện thông qua lớp CPaintDC như sau:

```
CPaintDC dc(this); // Device context để vẽ lên
...                // Thực hiện các tác vụ vẽ trên dc
```

THỰC HÀNH:

1. Tương tự VD04. Khi người dùng kết thúc ứng dụng, chương trình hiển thị hộp thông báo "Are you sure to exit this program ?" với hai mục YES-NO. Nếu người dùng chọn YES thì kết thúc:

HD: Cài đặt hành vi OnClose xử lý message WM_CLOSE cho CEmpWnd. Dùng hành vi MessageBox của CWnd để hiển thị câu thông báo. Nếu người dùng đồng ý thì thực hiện hành vi OnClose của CWnd để kết thúc, ngược lại không thực hiện xử lý gì cả (xem VD05)

2. Tương tự VD04 với phần demo là ảnh viên bi chạy trong client của cửa sổ.

HD: Dùng timer để liên tục phát WM_PAINT bằng hành vi Invalidate theo mỗi chu kỳ. Hành vi OnPaint thực hiện vẽ vào vùng client của cửa sổ chính một dòng chữ có nội dung chạy kiểu bồng chữ điện tử. (xem VD06).

3. Thực hiện ứng dụng cho phép hiển thị một vật thể có hình dạng bất kỳ trong vùng client. Các phím ←, ↑, →, ↓ cho phép dịch chuyển vật thể này.

HD: Như bài tập 2 nhưng không sử dụng timer. Dùng hành vi OnKeyDown xử lý message WM_KEYDOWN. Hành vi này kiểm tra giá trị phím nhận được **nChar** với các giá trị hằng phím VK_LEFT (phím ←), VK_UP (phím ↑), VK_RIGHT (phím →), VK_DOWN (phím ↓) để thay đổi tọa độ vật thể cho phù hợp. Sau cùng phát sinh message WM_PAINT để vẽ lại vật thể.

Ứng dụng công cụ GDI

6.1 DC VÀ BITMAP:

Vấn đề trang trí thiết bị đồ họa được tiến hành thông qua đối tượng DC quản lý thiết bị, trên cơ sở khai thác chức năng các công cụ GDI liên quan. Kết quả trang trí trên DC được quản lý bởi đối tượng Bitmap mà DC đang sử dụng. Bitmap là công cụ làm nền không thể thiếu cho các DC.

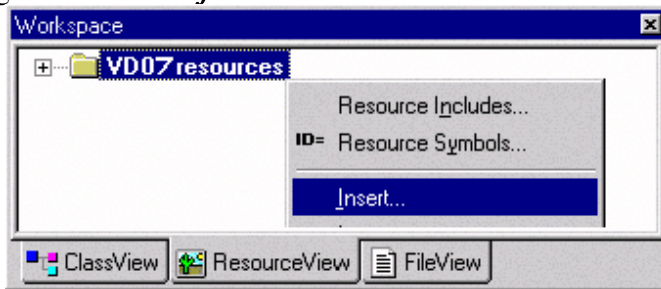
Việc khởi tạo nội dung cho đối tượng bitmap trong ứng dụng có thể được thực hiện bằng cách lấy ảnh bitmap từ resource (LoadBitmap) hay tạo mới nội dung cho bitmap dựa trên một DC xác định (CreateCompatibleBitmap).

Thông thường, ứng dụng đồ họa phải chuẩn bị sẵn các ảnh cần thiết trong resource của ứng dụng. Ở phần xử lý, các resource này được tải vào bộ nhớ làm nội dung cho các đối tượng bitmap. Từ các đối tượng bitmap này, ảnh sẽ được vẽ lên các thiết bị hiển thị đồ họa thông qua đối tượng DC tương ứng.

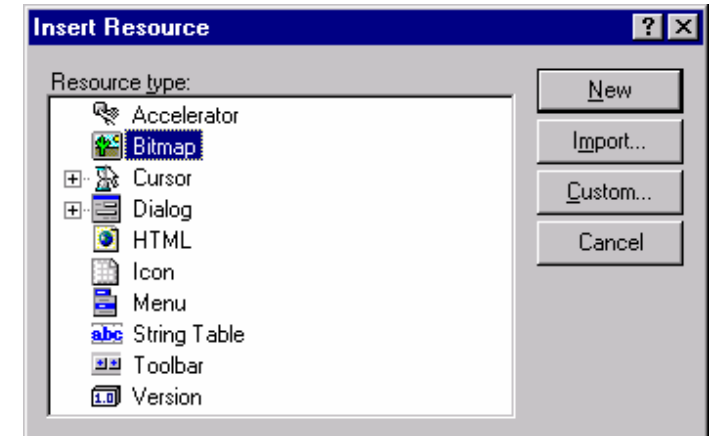
6.2 ỨNG DỤNG VỚI CỬA SỔ CHÍNH HIỂN THỊ ẢNH:

Trong phần này ta thực hiện ứng dụng có chức năng hiển thị một ảnh xác định trong vùng client của cửa sổ chính. Các bước tiến hành dự án như sau:

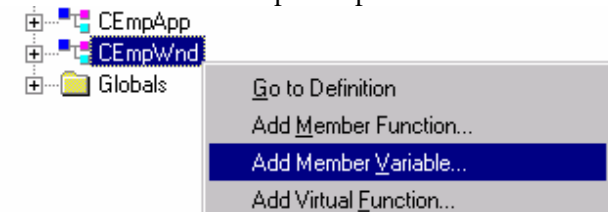
- Tạo dự án VD07 tương tự dự án VD06.
- Tạo một ảnh bitmap trong resource với số hiệu là IDB_MYPIC: Nội dung ảnh này có thể được tạo mới hoàn toàn hoặc lấy từ nội dung của một tập tin bitmap (.bmp) đã có. Chọn một trong hai cách sau:
 - **Cách 1 – ảnh bitmap được tạo mới:** Tương tự tạo mới icon (2.8).
 - **Cách 2 – ảnh bitmap được lấy từ nội dung tập tin bitmap (.bmp):**
 - Chọn trang **ResourceView** trong màn hình **Workspace**.
 - Right-click trên **Project Resource**:



- Chọn **Insert**:



- Chọn **Bitmap, Import**. Sau đó chọn tập tin chứa ảnh bitmap thông qua hộp hội thoại File-Folder.
- Ấn định các thông số của bitmap (số hiệu là IDB_MYPIC).
- Bổ sung đối tượng thuộc tính **m_myPict** kiểu CBitmap cho CEmpWnd:
 - Chọn trang **ClassView** trong màn hình **Workspace** của dự án.
 - Right-click trên tiêu đề của lớp CEmpWnd:



- Chọn **Add Member Variable...** :



- Nhập các thông tin về kiểu, tên và loại của thuộc tính. Chọn **OK**.
- Dùng hàm vi OnCreate của CEmpWnd để lấy ảnh bitmap từ resource làm nội dung của **m_myPict**. Xử lý được cài đặt như sau:

```
int CEmpWnd::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CWnd::OnCreate(lpCreateStruct) == -1)
        return -1;

    // Khởi động bitmap từ bitmap resource IDB_MYPIC
    mypict.LoadBitmap( IDB_MYPIC );
    return 0;
}
```

- Dùng hành vi OnPaint của CEmpWnd, vẽ ảnh bitmap trong **m_myPict** thông qua hành vi DrawState của đối tượng CPaintDC **dc**.

```
void CEmpWnd::OnPaint()
{
    CPaintDC dc(this);
    RECT rect;
    GetClientRect( &rect );          // Kích thước vùng client
    // draw on client
    dc.DrawState ( CPoint(0,0),      /* Góc trái trên của ảnh */
                  CSize (rect.right-rect.left, rect.bottom-rect.top),
                  &m_myPict, DST_BITMAP);
}
```

- Hành vi OnDestroy của CEmpWnd giải phóng đối tượng **m_myPict**.

```
void CEmpWnd::OnDestroy()
{
    CWnd::OnDestroy();
    m_myPict.DeleteObject();
}
```

Lưu ý: Để xử lý trang trí đồ họa trong vùng client của cửa sổ giao diện không gây ra hiện tượng "chớp", ta có thể sử dụng một số giải pháp sau:

- Không sử dụng đối tượng brush cho cửa sổ liên quan: Dùng giá trị NULL cho tham số này trong hành vi khởi tạo thông số của đối tượng cửa sổ.
- Dùng tham số FALSE cho yêu cầu cập nhật vùng client của cửa sổ (5.4.2)
- Dùng một DC trong bộ nhớ làm công cụ trang trí trung gian. Thực hiện các nội dung trang trí cần thiết lên DC này. Sau khi hoàn tất các tác vụ trang trí cần thiết thì chuyển nội dung DC công cụ lên DC của màn hình.

6.3 SAO CHÉP ẢNH TỪ DC VÀO DC, PHÓNG TO & THU NHỎ ẢNH:

Đối tượng DC cho phép sao chép lại nội dung trang trí đồ họa trên thiết bị hiển thị đồ họa được quản lý bởi một đối tượng DC khác lên thiết bị hiển thị đồ họa được quản lý bởi chính nó thông qua một số hành vi sau:

- **BitBlt** : Sao chép và giữ nguyên tỷ lệ trong nội dung ảnh.
- **StretchBlt** : Sao chép và thay đổi tỷ lệ trong nội dung ảnh.

Trong phần này ta xây dựng ứng dụng có các đặc điểm sau:

- Cửa sổ chính của ứng dụng có thể thay đổi kích thước.
- Nội dung ảnh hiển thị trong vùng client của cửa sổ tự động thay đổi kích thước một cách phù hợp khi kích thước cửa sổ thay đổi.

Việc thực hiện cần lưu ý các bước sau:

- Cửa sổ chính của ứng dụng có thuộc tính WS_THICKFRAME.
- Dùng một DC ảo để lồng ảnh bitmap thông qua đối tượng CBitmap. Vẽ ảnh bitmap từ DC này lên DC hiển thị.

Các bước thực hiện như sau:

- Tạo dự án VD08 tương tự dự án VD07.
- Hành vi OnPaint của CEmpWnd sử dụng DC trong bộ nhớ để lồng ảnh bitmap, từ đó vẽ lên vùng client của cửa sổ. Xử lý cài đặt như sau:

```
void CEmpWnd::OnPaint()
{
    CPaintDC dc(this);
    RECT rt;
    CDC memDC;
    CBitmap *oldBmp;
    BITMAP bmpInfo;

    GetClientRect(&rect);          // Kích thước vùng client
    mypict.GetBitmap(&bmpInfo);      // Lấy thông tin của ảnh bitmap
    memDC.CreateCompatibleDC(&dc);
    // Lồng bitmap m_myPict vào memDC và lưu lại bitmap cũ của nó.
    oldBmp = memDC.SelectObject(&m_myPict);
    // Chép ảnh từ memDC lên DC quản lý vùng client của cửa sổ: dc
    dc.StretchBlt( 0, 0, rt.right-rt.left, rt.bottom-rt.top, &memDC,
                  0, 0, bmpInfo.bmWidth, bmpInfo.bmHeight,
                  SRCCOPY);

    // Phục hồi ảnh bitmap cũ của memDC
    memDC.SelectObject(oldBmp); memDC.DeleteDC();
} // Xem VD08
```

6.4 DC TRONG BỘ NHỚ (DC ẢO) - VÙNG VẼ ĐỆM LÝ TƯỢNG:

Ứng dụng công cụ GDI

Nếu việc trang trí gồm nhiều thao tác phức tạp thì nên thực hiện chúng trên DC ảo, sau đó chuyển kết quả ra DC hiển thị. Chỉ một lần duy nhất cho mỗi nội dung trang trí, như thế sẽ cải thiện đáng kể chất lượng đồ họa.

Ứng với mỗi DC ảo tạo ra trong bộ nhớ, ngoài đối tượng CDC quản lý, ta cần sự phối hợp của đối tượng bitmap làm nền thay thế đối tượng bitmap tượng trưng không sử dụng được mà hệ thống gán cho DC khi tạo lập. Bộ cục xử lý của hành vi OnPaint có sử dụng đối tượng DC ảo như sau:

```

CClientDC  dc(this);           // Đối tượng DC hiển thị
RECT       rect;
CDC        memDrawDC;         // Đối tượng DC ảo để vẽ trung gian
CBitmap    memDrawBmp;
CBitmap    *memDrawOldBmp;
GetClientRect ( &rt );
int         CX = rect.right-rect.left ; int CY = rect.bottom-rect.top;
memDrawBmp.CreateCompatibleBitmap( &dc, CX, CY );
memDrawDC.CreateCompatibleDC( &dc );
memDrawOldBmp = memDrawDC.SelectObject( &memDrawBmp );

... // Trang trí memDrawDC

// Chuyển nội dung DC ảo sang dc:
dc.StretchBlt ( 0, 0, CX,CY, &memDrawDC, CX,CY, SRCCOPY );
// Hủy bỏ các đối tượng GDI:
memDrawDC.SelectObject(&memDrawOldBmp);
memDrawBmp.DeleteObject();
memDrawDC.DeleteDC();

```

Phần tiếp theo ta thực hiện ứng dụng tương tự VD08, đồng thời tạo dòng chữ chạy theo kiểu bảng chữ điện tử trong vùng client của cửa sổ chính.

- Tạo dự án VD09 tương tự dự án VD08.
- Xử lý **Trang trí memDrawDC** trong **OnPaint** của **CEmpWnd** như sau:

```

BITMAP      bmpInfo;
m_myPict.GetBitmap( &bmpInfo );
memDrawDC.StretchBlt( 0, 0, CX, CY, &memDC,
                     0, 0, bmpInfo.bmWidth, bmpInfo.bmHeight,
                     SRCCOPY );
memDrawDC.SetTextColor( RGB(255,0,0) ); // text color
memDrawDC.SetBkMode( TRANSPARENT ); // transparent
memDrawDC.TextOut( 30, 100, Chuỗi, 1 );

```

Xem VD09.

Với các đối tượng GDI được sử dụng thường xuyên thì việc lặp đi lặp lại các thao tác tạo và hủy bỏ chúng trong các hành vi trang trí của **CEmpWnd** sẽ làm lãng phí tài nguyên của hệ thống. Nên chuyển tất cả các thao tác đó về hai hành vi **OnCreate** và **OnDestroy** của **CEmpWnd** một cách phù hợp.

Bạn hãy thử áp dụng điều lưu ý này cho VD09.

6.5 ẢNH CHUYỂN ĐỘNG TRONG VÙNG CLIENT:

Được thực hiện một cách đơn giản bằng kỹ thuật hoạt hình. Ta chuẩn bị một số ảnh cơ bản của chuỗi hoạt động đó, sau đó thực hiện hiển thị và trao đổi ảnh theo trình tự với khoảng thời gian chờ hợp lý.

Các ảnh trong nội dung hoạt hình được quản lý bởi công cụ GDI thích hợp:

- **CBitmap**: Mỗi bitmap quản lý được một ảnh. Ta dùng nhiều bitmap. Ảnh vẽ bằng hành vi **DrawState** của đối tượng DC quản lý thiết bị hiển thị. Kích thước ảnh hiển thị không thay đổi.
 - **CDC**: Lồng tất cả các ảnh vào một DC. Từ DC này ta có thể chép bất kỳ phần ảnh cần vẽ nào sang DC hiển thị. Có thể thay đổi kích thước ảnh tùy ý: **StretchBlt**.
 - **CImageList**: Lớp đối tượng quản lý tập hợp nhiều ảnh có cùng kích thước. Khả năng thao tác trên danh sách ảnh của **CImageList** là rất tốt.
- Trong phần này ta xây dựng ứng dụng với hình ảnh chú bướm bay trong vùng client của cửa sổ. Tập tin **butterfly.bmp** trong thư mục **BMP** chứa các ảnh chuyển động của bướm. Ta dùng cách thứ 2, lồng các ảnh vào DC và vẽ lên DC hiển thị. Các bước thực hiện dự án như sau:
- Tạo dự án VD10 tương tự dự án VD09.
 - Bổ sung bitmap resource với số hiệu **IDB_ANIMATION** mà nội dung được lấy từ tập tin chứa các ảnh hoạt hình. Ghi nhớ số ảnh trong bitmap đó. Chẳng hạn, chọn tập tin **butterfly.bmp** trong thư mục **BMP**. Tập tin này có 4 ảnh, kích thước 32x28.
 - Bổ sung các đối tượng thuộc tính **protected** cho lớp **CEmpWnd**:
 - **m_butterDC**: Đối tượng CDC, quản lý DC lồng ảnh.
 - **m_butterBmp**: Đối tượng CBitmap, quản lý các ảnh hoạt hình.
 - **m_butterOldBmp**: Đối tượng CBitmap*, quản lý con trỏ chỉ đến đối tượng bitmap cũ của **m_butterDC**.
 - **m_pictNo**: Kiểu **int**, quản lý số thứ tự của hình đang được hiển thị trong các ảnh hoạt hình nói trên.
 - Hành vi **OnCreate** của **CEmpWnd** thực hiện các chuẩn bị:

```
int CEmpWnd::OnCreate( LPCREATESTRUCT lpCreateStruct )
```



```

{
    if (CWnd::OnCreate(lpCreateStruct) == -1)
        return -1;
    // TODO: Add your specialized creation code here
    SetTimer( IDD_TIMER, 250, NULL );
    m_myPict.LoadBitmap( IDB_MYPIC );
    // Animation objects by EX10
    m_butterBmp.LoadBitmap( IDB_ANIMATION );
    m_butterDC.CreateCompatibleDC( NULL );
    m_butterOldBmp = butterDC.SelectObject( &m_butterBmp );
    m_pictNo = 0;
    return 0;
}

```

- Hành vi OnPaint vẽ hình và tự tăng vị trí chọn hình cho lần vẽ sau đó:

```

memDrawDC.StretchBlt( 20, 50, 32, 28, &m_butterDC,
                      m_pictNo*32, 0, 32, 28, SRCCOPY );
m_pictNo++;           // Chọn ảnh kế tiếp
if (m_pictNo >= 4) m_pictNo = 0;

```

- Hành vi OnDestroy hủy bỏ các thuộc tính GDI:

```

m_butterDC.SelectObject(m_butterOldBmp);
m_butterBmp.DeleteObject();
m_butterDC.DeleteDC();

```

Nhân xét: Phần nền của ảnh hoạt hình che khuất ảnh nền. Để khắc phục ta sử dụng một ảnh bitmap làm mặt nạ cho ảnh hoạt hình để ẩn định phần nội dung được vẽ trên ảnh hoạt hình.

- Hành vi **MaskBlt** của CDC cho phép dùng monochrome bitmap làm lưới lọc ảnh điểm phần nổi của ảnh khi chép ảnh từ DC nguồn lên DC đích (95/98/Me unsupported). Bạn hãy thử thực hiện với VD10 như bài tập.

6.6 **CImageList - CÔNG CỤ QUẢN LÝ BỘ ẢNH CÙNG CỖ:**

Xây dựng dự án VD11 trên cơ sở cải tiến VD10; bộ ảnh hoạt hình sẽ được quản lý bởi đối tượng CImageList:

- Tạo dự án VD11 tương tự dự án VD10.
- Bổ sung đối tượng thuộc tính **m_butterImg** kiểu CImageList làm nhiệm vụ quản lý các ảnh. Xóa các thuộc tính m_butterBmp, m_butterDC và m_butterOldBmp vì không còn cần thiết.

- Hành vi OnCreate của CEmpWnd thực hiện các chuẩn bị như sau:

```

int CEmpWnd::OnCreate( LPCREATESTRUCT lpCreateStruct )
{
    if (CWnd::OnCreate(lpCreateStruct) == -1)
        return -1;
    // TODO: Add your specialized creation code here
    SetTimer( IDD_TIMER, 250, NULL );
    m_mypict.LoadBitmap( IDB_MYPIC );
    // animation object by EX10
    m_butterImg.Create( IDB_ANIMATION, 32, 4, RGB(255, 255, 255) );
    m_pictNo = 0;
    return 0;
}

```

- Hành vi OnPaint sử dụng hành vi Draw của **m_butterImg** vẽ ảnh:

```

m_butterImg.Draw( &memDrawDC, m_pictNo, CPoint(30, 170),
                  ILD_NORMAL );
m_pictNo++;
if (m_pictNo >= 4) m_pictNo = 0;

```

- Hành vi OnDestroy hủy bỏ đối tượng CImageList:

```

m_butterImg.DeleteImageList();

```

6.7 **CRgn - CỬA SỔ CÓ HÌNH DẠNG TÙY Ý:**

Khuôn dạng của region có thể dùng làm khuôn dạng của cửa sổ thông qua hành vi SetWindowRgn của đối tượng cửa sổ. Phần sau đây minh họa cho vấn đề trên và được cài đặt trong hành vi **OnCreate** của cửa sổ (VD12).

```

CRgn newShape;
newShape.CreateEllipticRgn( 0, 0, 200, 100 );
SetWindowRgn( newShape, TRUE );

```

THỰC HÀNH:

- Viết ứng dụng với cửa sổ giao diện chính có hình tam giác.
- Viết ứng dụng với cửa sổ giao diện chính có hình ngôi sao năm cánh.
- Viết ứng dụng hiển thị nội dung của nhiều ảnh theo thứ tự luân phiên. Sự chuyển tiếp giữa hai ảnh bất kỳ được thực hiện bằng kỹ thuật pha trộn ảnh.
- Viết ứng dụng hiển thị ảnh cuộn (scroll) từ trái sang phải.

5. Viết ứng dụng hiển thị ảnh zoom từ bé đến lớn và ngược lại.
6. Viết ứng dụng hiển thị một dòng chữ bất kỳ theo hình ảnh cuộn (scroll).
7. Viết ứng dụng với hình ảnh chú bướm bay thơ thẩn trong vùng client.
8. Viết ứng dụng ScreenSaver và sử dụng nó cho máy tính của bạn.
9. Viết ứng dụng với màn hình chính chứa các dòng chữ trôi từ đáy lên đỉnh màn hình, liên tục đến khi kết thúc ứng dụng.

HD: Tạo DC ảo và viết các dòng chữ lên DC này. Sau đó chép phần nội dung thích hợp của DC ảo sang DC thực.

- Chiều rộng DC ảo bằng chiều rộng DC thực. Chiều cao DC ảo bằng chiều cao DC thực + 2 lần chiều cao một dòng chữ trên DC.
 - Viết các dòng chữ hợp lệ (tọa độ hiển thị nằm trong giới hạn DC ảo) lên DC ảo. Sau mỗi lần hiển thị, tịnh tiến vị trí vẽ lên phía trên một đoạn tùy ý, nếu vị trí tịnh tiến làm tất cả các dòng chữ rơi ra ngoài DC ảo thì ấn định lại vị trí ấy ở cuối DC ảo.
10. Chỉnh sửa bài tập 9 như sau:
- Hiển thị các dòng chữ với độ sáng giảm dần từ dòng dưới lên dòng trên để tạo hiệu ứng 3 chiều.
 - Tác vụ chép sử dụng hàm StretchBlt trên từng dòng pixel để ảnh xạ ảnh chữ nhật từ DC ảo thành ảnh tam giác cân trên DC thực.

Kết quả nhận được là hình ảnh trôi các dòng chữ theo chiều thứ 3.

11. Viết ứng dụng với màn hình chính hiển thị "thiên thạch vũ trụ".

HD: Mỗi 'thiên thạch' được quản lý bằng tọa độ 3 giá trị (x, y, z).

- Tọa độ z tiến dần về phía người dùng sau mỗi lần hiển thị.
- Tọa độ x, y tương ứng thay đổi theo z:
 $x = \text{Hoành độ tâm vùng Client} + x * 100 / z;$
 $y = \text{Tung độ tâm vùng Client} + y * 100 / z;$
- Độ sáng phụ thuộc vào z.
- Dùng hàm vẽ Circle của DC để vẽ các 'thiên thạch'.

12. Kết hợp hai bài tập 10 và 11 để có một ứng dụng với màn hình chính như màn hình *StarWar*.

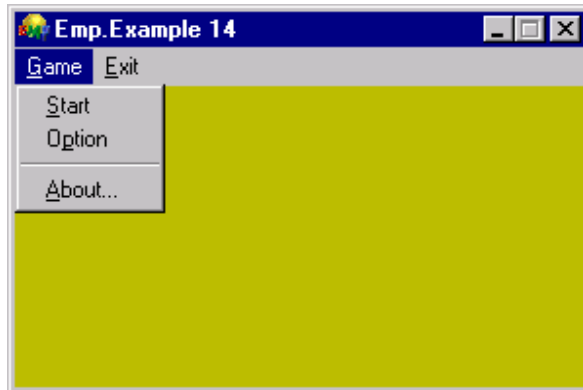
13. Dùng lớp đối tượng CTime của MFC để lấy giờ hiện hành của hệ thống. Thực hiện ứng dụng hiển thị một chiếc đồng hồ analog (đồng hồ kim) chạy theo thời gian.

MENU & PHÍM TẮT

7.1 ĐỊNH NGHĨA:

Menu là hệ thống các mục chọn tương ứng với các xử lý xác định. Thông qua menu, người dùng có thể dễ dàng ấn định thực hiện xử lý mong muốn.

Xem một ứng dụng với hệ thống menu như sau:



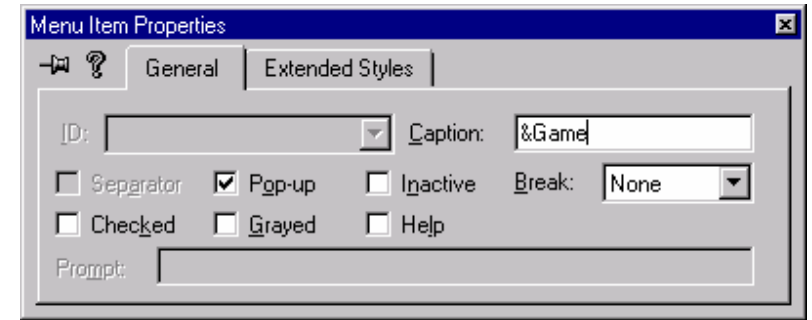
- *Game* và *Exit* là các mục chọn của menu bar.
- Mục chọn *Game* gắn với một menu popup có ba mục chọn: *Start*, *Option*, *About* và dấu ngăn cách mục (separator).

7.2 MENU RESOURCE:

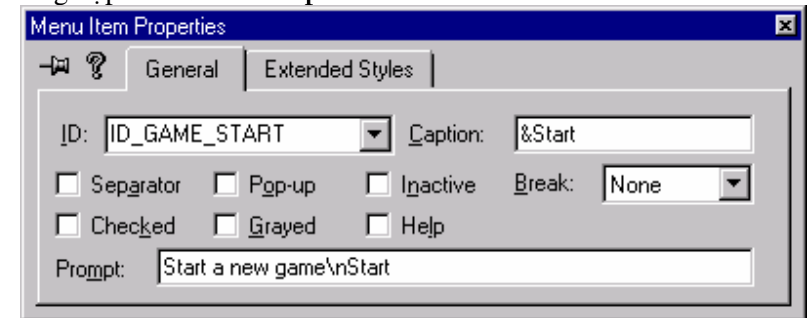
Để tiện việc sử dụng và chỉnh sửa menu trong chương trình, VC cho phép soạn thảo và lưu cấu trúc menu vào resource của ứng dụng một cách độc lập, phần chương trình sẽ dùng các lệnh cần thiết để nạp và sử dụng menu.

Cách tạo menu trong resource:

- Tạo mới menu resource: Thực hiện tương tự việc tạo mới icon (2.8).
Lưu ý: Chọn **Resource Type** là Menu.
- Đặt số hiệu cho menu (ví dụ IDR_MAINFRAME với menu chính).
- Thiết kế menu thông qua màn hình thiết kế mà ta vừa nhận được từ bước trên. Các thao tác cơ bản như sau:
 - **Cài đặt mục popup.** Double-click (hoặc gõ phím Enter) trên vị trí dự định cài đặt mục popup:
Ta nhận được hộp hội thoại **Menu Item Properties**:



- *Caption:* Nội dung thông báo. & dùng đặt trước ký tự phím tắt.
- Đánh dấu chọn mục *Pop-up*. Cuối cùng, gõ Enter kết thúc.
- **Cài đặt mục lệnh:** Thực hiện tương tự như trên nhưng phần ấn định trong hộp **Menu Item Properties** như sau:



- *Prompt:* Nội dung giải thích (được hiển thị trên StatusBar) và nội dung giải thích văn bản (Tiptext trên thanh công cụ). Giữa hai nội dung này được ngăn cách bằng ký tự \n.
- *ID:* Số hiệu của mục chọn (menu-ID). Nên đặt tên gợi nhớ.
- **Cài đặt dấu ngăn cách:** Thực hiện tương tự như trên. Đánh dấu chọn *Separator* trong hộp **Menu Item Properties**.
- **Chèn mục vào giữa các mục chọn đã có:** Đưa vật sáng đến vị trí chèn, sau đó nhấn phím Insert.
- **Xóa mục cài đặt:** Đưa vật sáng đến vị trí xóa, gõ phím Delete.

➤ Tạo mới dự án VD13 như VD12, sau đó thiết kế menu resource với số hiệu IDR_MAINFRAME. Số hiệu các mục chọn lần lượt là:

- Start = ID_GAME_START
- Option = ID_GAME_OPTION
- About = ID_GAME_ABOUT
- Exit = ID_EXIT

7.3 SỬ DỤNG MENU RESOURCE:

Menu resource là cơ sở khởi tạo hệ thống menu dùng trong ứng dụng. Hệ thống menu có thể được gắn vào cửa sổ giao diện để tiện sử dụng. Quá trình này được thực hiện thông qua các bước sau:

- **Nạp menu resource vào bộ nhớ:**

```
HMENU LoadMenu (  
    HINSTANCE hInstance,    // Handle của ứng dụng  
    LPCTSTR lpMenuName     // Chuỗi tên resource của menu  
);
```

Hàm trả về handle của menu trong bộ nhớ.

- Giá trị handle của ứng dụng nhận được từ hàm sau:
HINSTANCE AfxGetInstanceHandle();
- Mỗi đối tượng trong resource được nhận diện bằng một số hiệu hoặc chuỗi tên. Hàm sau đây giúp chuyển số hiệu của đối tượng resource sang chuỗi tên tương ứng:
LPTSTR MAKEINTRESOURCE(UINT resourceID);

- **Gắn menu với cửa sổ giao diện:** Dùng handle của menu làm tham số cho hành vi khởi tạo thông số **CreateEx** của đối tượng cửa sổ. Hành vi **InitInstance** của đối tượng quản lý ứng dụng đảm nhận việc này:

```
BOOL CEmpApp::InitInstance()  
{  
    CEmpWnd *main = new CEmpWnd;  
    HICON myIcon = LoadIcon(IDR_MAINFRAME);  
    HCURSOR myCursor = LoadCursor(IDC_MAINFRAME);  
    CBrush myBrush;  
    CString myClassName = "Emp.WndClassName";  
    myBrush.CreateSolidBrush( RGB(190, 190, 0) );  
    m_pMainWnd = main;  
    main->CreateEx( WS_EX_TOPMOST,  
        AfxRegisterWndClass( CS_VREDRAW | CS_HREDRAW,  
                             myCursor, myBrush, myIcon ),  
        _T("Emp.Example 13"), WS_SYSMENU | WS_VISIBLE |  
        WS_MINIMIZEBOX | WS_THICKFRAME,  
        100, 100, 300, 200, NULL,  
        LoadMenu( AfxGetInstanceHandle(),  
                  MAKEINTRESOURCE( IDR_MAINFRAME ) ) );  
    main->ShowWindow( SW_SHOW );  
    return TRUE;  
} // Xem VD13 (hệ thống menu chưa có xử lý)
```

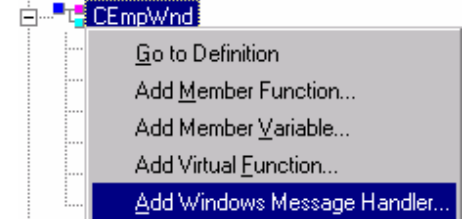
7.4 MỤC XỬ LÝ COMMAND MESSAGE TỪ MỤC CHỌN CỦA MENU:

Để mục chọn của menu có ý nghĩa sử dụng ta phải cài xử lý cho chúng.

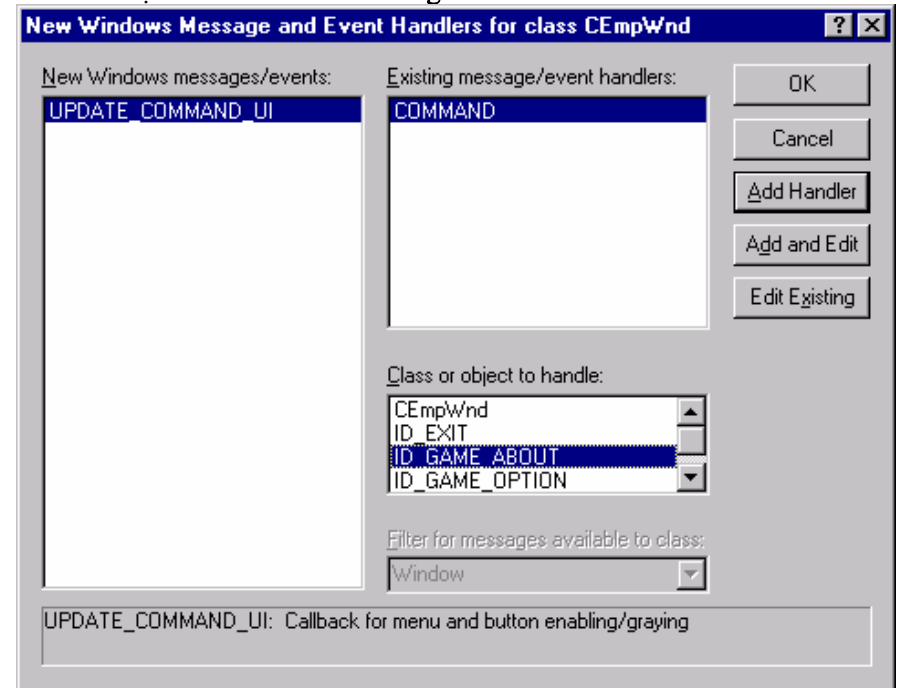
Khi người dùng chọn một mục trên menu, hệ thống lập tức gửi WM_COMMAND đến ứng dụng với tham số **wParam** chứa số hiệu (ID) của mục menu được chọn. Bất cứ đối tượng nào trong ứng dụng có chức năng xử lý message đều có thể đảm nhận việc xử lý các message này.

➤ Tiếp theo, ta xây dựng ứng dụng với hệ thống menu như VD13. Mục chọn **About** hiển thị hộp thông báo giới thiệu tác giả và sản phẩm.

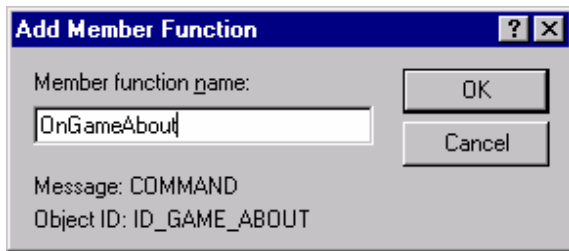
- Tạo dự án VD14 như VD13.
- Dùng lớp CEmpWnd cài đặt mục xử lý message:
 - Trong màn hình **Workspace**, chọn **ClassView**. Right-click trên tiêu đề lớp CEmpWnd:



- Chọn **Add Windows Message Handler...**



- Chọn số hiệu ID_GAME_ABOUT, click chọn COMMAND. Sau đó chọn mục **Add and Edit**.



- Đặt tên hành vi xử lý message WM_COMMAND. Chọn **OK**.
- Nội dung cài đặt của hành vi này như sau:

```
void CEmpWnd::OnGameAbout()
{
    MessageBox( "The program was written by Mr.EMP\n"
               "This product is a not-licensed one.",
               "About",
               MB_OK | MB_ICONINFORMATION );
}
```

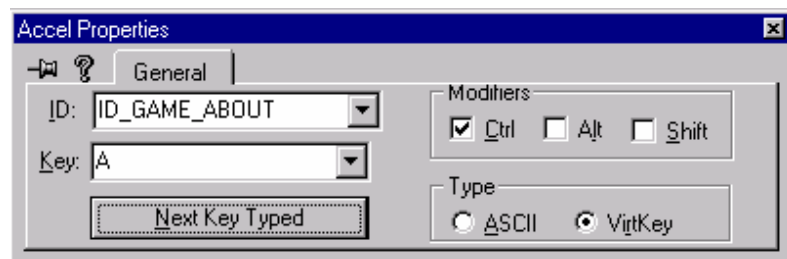
☞ Xem bảng **MessageMap** của lớp CEmpWnd, mục ID_GAME_ABOUT ?

7.5 PHÍM TẮT (HOT KEY) CHO MỤC CHỌN TRÊN MENU:

Phím tắt là tổ hợp phím cho phép thực hiện nhanh một mục chọn xác định trên hệ thống menu. Các phím tắt được định nghĩa trong phần resource của ứng dụng. Chương trình sẽ dùng lệnh để nạp bảng phím tắt khi cần.

☞ Trong phần này, ta viết ứng dụng tương tự VD14 với các phím tắt Ctrl+S, Ctrl+P, Ctrl+A và Ctrl+E cho các mục menu: Start, Option, About và Exit.

- Tạo dự án VD15 tương tự VD14.
- Tạo mới bảng phím tắt trong resource (Accelerator resource): Thực hiện tương tự việc tạo mới icon (2.8). **Resource Type** = *Accelerator*.
- Đặt số hiệu cho Accelerator (giả sử là IDR_MAINFRAME).
- Thiết kế bảng phím tắt. Các thao tác cơ bản như sau:
 - **Bổ sung định nghĩa phím tắt.** Double-click trên dòng rỗng:



- **ID** : Số hiệu mục menu sử dụng phím tắt.
- **Key** : Phím tắt.
- **Modifier**: Các phím hệ thống phối hợp.
- **Type** : *ASCII* → Phím ký tự ; *VirtKey* → Phím bất kỳ.

Sau khi ấn định xong gõ phím **Enter**.

- **Chỉnh sửa định nghĩa phím tắt.** Double-click trên dòng phím tắt, điều chỉnh các thông tin cần thiết. Gõ phím **Enter** để kết thúc.
- **Xóa định nghĩa phím tắt.** Chọn dòng định nghĩa phím, gõ phím **Del**. Lưu nội dung bảng phím tắt và đóng màn hình soạn thảo phím tắt.
- Sử dụng phím tắt trong chương trình: Thực hiện tuần tự hai bước sau:

- Nạp bảng phím tắt vào bộ nhớ:

HACCEL **LoadAccelerators** (

HINSTANCE hInstance, // Handle của ứng dụng
LPCTSTR lpTableName // Chuỗi tên resource

); Hàm trả về handle của bảng phím tắt trong bộ nhớ.

- Dịch phím tắt trên message nhận được từ hàng chờ của ứng dụng:

int **TranslateAccelerator** (

HWND hWnd, // Handle cửa sổ giao diện dùng phím tắt
HACCEL hAccTable, // Handle của bảng phím tắt
LPMMSG lpMsg // Con trỏ biến chứa message điều phối

); Hàm này phải được thực hiện trên tất cả các message mà ứng dụng nhận được. Do đó, nó được lồng vào vòng lặp **MessageLoop** của ứng dụng. Lớp **CWinThread** (xem 2.4) cho phép cài đặt đặc tính này thông qua hành vi sau của lớp:

BOOL CWinThread::PreTranslateMessage(MSG *pMsg);

Trong các lớp kế thừa CWinThread, cài đặt này có bố cục như sau:

BOOL CEmpApp::PreTranslateMessage(MSG *pMsg)

```
{ // CEmpApp là lớp kế thừa CWinApp (từ CWinThread)
  // Thực hiện hàm dịch trên message nhận được.
  TranslateAccelerator( m_pMainWnd->m_hWnd,
                       m_hAccel, pMsg );

  // m_hAccel : Handle của bảng phím tắt.
  return CWinApp::PreTranslateMessage( pMsg );
}
```

- Áp dụng cho dự án VD15: Bổ sung một số thuộc tính và hành vi cho lớp CEmpApp:

- Thuộc tính **m_hAccel** kiểu HACCEL lưu handle bảng phím tắt.
- Hành vi **InitInstance**: Bổ sung lệnh nạp bảng phím tắt và giữ giá trị handle của nó vào biến **m_hAccel** để sử dụng sau này:

```
m_hAccel = LoadAccelerators( AfxGetInstanceHandle(),
                             MAKEINTRESOURCE( IDR_MAINFRAME ) );
```

- Hành vi kế thừa **PreTranslateMessage** có cài đặt như trên.

7.6 LỚP QUẢN LÝ MENU - CMenu:

Để tiện thao tác trên menu, MFC cung cấp lớp đối tượng CMenu cho phép quản lý menu thông qua các thuộc tính và hành vi đặc trưng sau:

- **CMenu()**; Hành vi tạo lập đối tượng menu.
- **BOOL LoadMenu(UINT nIDResource);** Khởi tạo thông số cho đối tượng menu từ menu resource.
- **BOOL DestroyMenu()**; Hủy bỏ đối tượng menu.
- **BOOL DeleteMenu(UINT nPosition, UINT nFlags);** Xóa một mục chọn trong menu. Bộ giá trị (nPosition, nFlags) xác định mục chọn.
nFlags = MF_BYCOMMAND : **nPosition** là số hiệu của mục chọn (menu-ID).
= MF_BYPOSITION : **nPosition** là vị trí thứ tự của mục chọn (đếm từ 0).
- **BOOL AppendMenu (**
UINT nFlags, // Đặc điểm mục chọn
UINT nIDNewItem = 0, // Số hiệu mục chọn
LPCTSTR lpszNewItem=NULL // Chuỗi thông báo của mục
); Thêm mục chọn vào cuối hệ thống menu.
nFlags = MF_SEPARATOR : Các tham số khác không có ý nghĩa.
= MF_STRING : Các thông số được hiểu như trên.
= MF_POPUP : **nIDNewItem** là handle của menu popup.
- **BOOL InsertMenu (**
UINT nPosition, // Vị trí được chèn.
UINT nFlags, // Các thông tin khác
UINT nIDNewItem = 0, // tương tự **AppendMenu()**.
LPCTSTR lpszNewItem = NULL
); Chèn thêm mục chọn vào trước mục được chỉ bởi **nPosition**.
- **UINT CheckMenuItem (**
UINT nIDCheckItem, // Số hiệu | vị trí mục chọn

UINT nCheck // Cách thức đánh dấu mục chọn
); Đánh dấu hoặc hủy bỏ đánh dấu mục chọn trên menu.
nCheck là giá trị kết hợp của hai nội dung:

- **Cách đánh dấu mục:** **= MF_CHECKED** : Đánh dấu
= MF_UNCHECKED : Bỏ đánh dấu
- **Cách chỉ định mục:** **= MF_BYPOSITION** : Theo vị trí
= MF_BYCOMMAND : Theo số hiệu mục

nIDCheckItem tương ứng chứa số hiệu hoặc vị trí của mục chọn.

- **UINT EnableMenuItem (**
UINT nIDEnableItem, // Số hiệu | vị trí mục chọn (như trên)
UINT nEnable // Cách thức ấn định mục chọn.
); Cấm hoặc cho phép mục chọn hoạt động.
nIDEnableItem là giá trị kết hợp của hai nội dung:
- **Cách định vị mục chọn:** Như trên.
- **Trạng thái mục:** **= MF_ENABLED** : Cho phép mục hoạt động.
= MF_DISABLED : Cấm mục hoạt động.
= MF_GRAYED : Che mờ mục chọn.
- **int GetMenuString (**
UINT nIDItem, // Số hiệu mục chọn
CString& rString, // Tham biến nhận kết quả
UINT nFlags // Cách định vị mục chọn
); Lấy nội dung thông báo của một mục chọn.
- **BOOL ModifyMenu (**
UINT nPosition, // Số hiệu | vị trí của mục chọn
UINT nFlags, // Cách định vị mục chọn
UINT nIDNewItem = 0, // Số hiệu | vị trí mới của mục chọn
LPCTSTR lpszNewItem = NULL // Thông báo mới của mục chọn
); Thay đổi các thông số liên quan đến mục chọn.

Lưu ý: Hành vi **GetMenu** của CWnd trả về con trỏ đến đối tượng menu gắn với cửa sổ. Giá trị trả về = NULL nếu cửa sổ không gắn với menu nào.

✎ Giả sử có yêu cầu viết ứng dụng VD16 tương tự VD15; trong đó mục chọn **Start** tự động chuyển thành **Stop** và ngược lại mỗi khi người dùng chọn mục này. Công việc trên được thực hiện thông qua mục xử lý command message ID_GAME_START. Bạn hãy thử thực hiện ứng dụng này (xem VD16).

7.7 XỬ LÝ ĐIỀU KHIỂN MỤC CHỌN CỦA MENU:

Trong phần này, ta xây dựng ứng dụng như VD16. Khi chọn mục **Start** (Star → Stop), ứng dụng không cho phép người dùng chọn mục **Option**.

- ✓ **Cách thứ nhất:** Cài đặt xử lý cho mục chọn **Start** (**Stop**) để thực hiện cấm hoặc cho phép mục chọn **Option** một cách phù hợp.
- ✓ **Cách thứ hai:** Dùng trạng thái hiện hành của mục chọn **Start** để quyết định cho phép hay cấm hoạt động của mục **Option**. Cách làm này dựa trên cơ chế xử lý điều khiển đối tượng phát sinh command message là mục **Option**. Thông tin trạng thái của mục **Start** được lưu trong thuộc tính **m_isStop**. Thông qua giá trị này, hành vi xử lý điều khiển chọn giá trị tham số thích hợp dùng cho hành vi **Enable** của đối tượng CCmdUI chỉ bởi con trỏ làm tham số; TRUE (cho phép) , FALSE (cấm). Các bước thực hiện dự án theo cách thứ hai như sau:
 - Tạo dự án VD17 tương tự VD16. Chỉnh sửa lớp CEmpWnd như sau:
 - Bổ sung thuộc tính protected **m_isStop** kiểu BOOL cho lớp CEmpWnd. Không dùng biến cục bộ isStop như VD16, thay thế biến này bằng **m_isStop**, chỉnh sửa các lệnh liên quan. Thông qua hành vi OnCreate, gán giá trị khởi đầu cho **m_isStop** là FALSE.
 - Khai báo xử lý điều khiển cho mục **Option**: Thực hiện tương tự mục (7.4). Lưu ý chọn số hiệu mục chọn ID_GAME_OPTION, sau đó chọn UPDATE_COMMAND_UI. Cuối cùng chọn **Add and Edit**.
 - Đặt tên cho hành vi xử lý điều khiển. Cài đặt của hành vi này như sau:

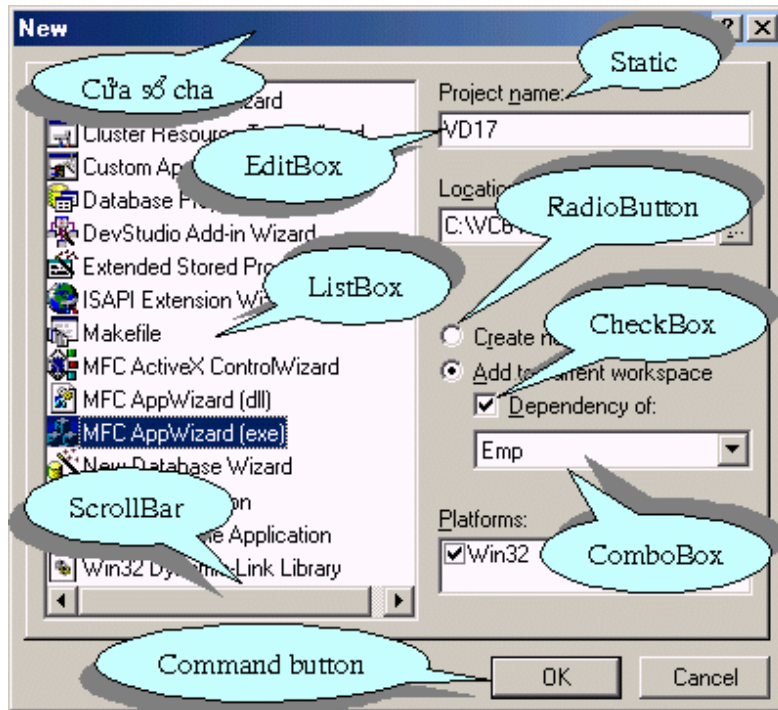
```
void CEmpWnd::OnUpdateGameOption (CCmdUI* pCmdUI)
{
    pCmdUI->Enable( !m_isStop );    // pCmdUI con trỏ tham số.
}
```

THỰC HÀNH:

1. Từ VD15, bổ sung hành vi PreTranslateMessage và cài đặt xử lý sử dụng bảng phím tắt cho lớp CEmpWnd.
2. Cài đặt hành vi xử lý mục chọn thoát (Exit) cho lớp CEmpWnd.
HD: Để chấm dứt ứng dụng, ta dùng hành vi PostMessage mà CEmpWnd kế thừa từ CWnd để gửi WM_QUIT đến cửa sổ của nó như sau:
PostMessage(WM_QUIT, 0, 0);
3. Thực hiện yêu cầu mục (7.7) bằng cách thứ nhất.

Các lớp đối tượng nhập liệu

Lớp đối tượng nhập liệu, kế thừa từ lớp `CWnd`, cho phép quản lý các mục nhập liệu (controls) trên cửa sổ giao diện. Cửa sổ chứa các control gọi là cửa sổ cha. Hình ảnh sau là một cửa sổ cha với một số loại control phổ biến:



8.1 `CStatic`:

`CStatic` là lớp đối tượng quản lý mục thông báo hoặc ảnh trên cửa sổ giao diện. Các thuộc tính và hành vi đặc trưng của lớp này như sau:

- `CStatic()`; Tạo lập đối tượng rỗng.
- `BOOL Create (`
`LPCTSTR lpszText, // Nội dung thông báo của mục`
`DWORD dwStyle, // Thông số dạng mục thông báo`
`const RECT& rect, // Tọa độ, kích thước của mục`
`CWnd* pParentWnd, // Con trỏ đối tượng cửa sổ cha`

`UINT nID = 0xFFFF // Số hiệu mục thông báo, duy nhất.`

`);` Khởi tạo thông số cho đối tượng mục thông báo.

`dwStyle`: Phải chứa các thông số qui định đối với control:

`WS_CHILD | WS_VISIBLE [| WS_DISABLED]`

và các thông số bổ sung cho đặc trưng của mục như sau:

`SS_BITMAP : Có chức năng hiển thị ảnh bitmap.`

`SS_ICON : Có chức năng hiển thị ảnh icon hoặc cursor.`

`SS_XXX : Một số thông số khác, xem MSDN.`

☞ Màu nền đối tượng `CStatic` do MFC đăng ký, muốn thay đổi phải xây dựng lớp kế thừa `CStatic`, cài đặt hành vi `PreCreateWindow`.

▪ `HBITMAP SetBitmap (`

`HBITMAP hBitmap // Handle của ảnh bitmap`

`);` Dùng ảnh bitmap làm nội dung của mục thông báo.

▪ `HBITMAP GetBitmap ();` Trả về handle của ảnh bitmap đang dùng.

▪ `HICON SetIcon (`

`HICON hIcon // Handle của icon`

`);` Dùng icon làm nội dung của mục thông báo.

▪ `HICON GetIcon ();` Trả về handle của icon đang dùng.

▪ `HCURSOR SetCursor (`

`HCURSOR hCursor // Handle của cursor`

`);` Chọn cursor cho mục thông báo. Windows lấy cursor này làm hình dạng con chuột khi nó di chuyển vào vùng giới hạn của mục.

▪ `HCURSOR GetCursor ();` Trả về handle của cursor đang dùng.

☞ Có thể dùng hành vi `CreateEx` (lớp `CStatic` kế thừa từ `CWnd`) khởi tạo thông số mục thông báo với dạng mở rộng (`WS_EX_XXX`). Trong trường hợp này, giá trị cho tham số `lpszClassName` là `_T("STATIC")`.

☞ Phần này, ta thực hiện ứng dụng VD18 tương tự VD15; cài đặt hai mục thông báo trên cửa sổ giao diện chính: mục thứ nhất có nội dung "My Icon is", mục thứ hai hiển thị icon của ứng dụng.

☞ Dùng hai đối tượng thuộc lớp `CStatic` để tạo và quản lý hai mục thông báo. Đối tượng `CEmpWnd` lấy chúng làm thuộc tính để tiện quản lý.

Dự án được thực hiện qua các bước sau:

- Tạo dự án VD18 tương tự VD15. Chỉnh sửa lớp `CEmpWnd` như sau:
- Bổ sung hai đối tượng thuộc tính protected: `m_staticIcon` và `m_staticText` có kiểu `CStatic`.
- Hành vi `OnCreate` thực hiện khởi tạo thông số cho hai đối tượng này:


```

int CEmpWnd::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CWnd::OnCreate(lpCreateStruct) == -1)
        return -1;
    m_staticText.Create( _T("This is my Icon:"),
                        WS_CHILD|WS_VISIBLE,
                        CRect(10, 15, 120, 35), this );
    m_staticIcon.Create( _T(""),
                        WS_CHILD | WS_VISIBLE | SS_ICON,
                        CRect( 125,7,157,39 ), this );
    m_staticIcon.SetIcon (
                        AfxGetApp()->LoadIcon(IDR_MAINFRAME) );
    return 0;
}

```

- Biên dịch và chạy thử ứng dụng.

8.2 CEdit:

CEdit là lớp đối tượng quản lý hộp nhập trên cửa sổ giao diện. Ngoài việc hỗ trợ xử lý các hoạt động nhập liệu, CEdit có khả năng thông tin cho cửa sổ cha của hộp nhập về tình hình nhập liệu đang diễn ra trong hộp.

- **CEdit();** Tạo lập đối tượng rỗng.
- **BOOL Create (**
 DWORD dwStyle, // Thông số dạng hộp nhập
 const RECT& rect, // Tọa độ và kích thước hộp nhập
 CWnd* pParentWnd, // Con trỏ đối tượng cửa sổ cha
 UINT nID // Số hiệu hộp nhập, phải duy nhất
); Khởi tạo thông số cho đối tượng hộp nhập liệu.
 dwStyle: Gồm thông số qui định đối với control và các dạng bổ sung:
 ES_MULTILINE : Hộp nhập cho phép nhiều dòng.
 ES_PASSWORD : Hộp nhập dùng nhập password.
 ES_READONLY : Hộp nhập chỉ xem nội dung.
 ES_XXX : Xem thông tin trong MSDN.
- **void GetRect (**
 LPRECT lpRect // Con trỏ đến biến RECT chứa kết quả
); Lấy thông tin về tọa độ, kích thước hộp nhập.
- **BOOL GetModify();** Trả về TRUE nếu nội dung hộp nhập thay đổi.
- **void SetModify(BOOL bModified = TRUE);** Xác lập hoặc xóa cờ hiệu ghi nhận sự thay đổi nội dung trong hộp nhập.

- **void SetMargins (**
 UINT nLeft, // Lề trái và
 UINT nRight // lề phải tính bằng pixel
); Ấn định biên trái và biên phải của hộp nhập.
- **DWORD GetMargins();** Trả về giá trị chứa thông tin biên trái (WORD thấp) và biên phải (WORD cao) của hộp nhập.
- **void GetSel (**
 int& nStartChar, // Biến chứa vị trí ký tự đầu tiên và
 int& nEndChar // ký tự cuối cùng của đoạn văn bản.
); Lấy thông tin về đoạn văn bản đang được đánh dấu trong hộp nhập.
- **void SetSel (**
 int nStartChar, // Vị trí ký tự đầu tiên
 int nEndChar, // Vị trí ký tự cuối cùng của đoạn văn bản
 BOOL bNoScroll = FALSE
); Đánh dấu một đoạn văn bản trong hộp nhập.
- **BOOL SetReadOnly(BOOL bReadOnly = TRUE);** Xác lập trạng thái chỉ xem đối với dữ liệu của hộp nhập.
- **void Copy();** Chép nội dung đoạn văn bản đang được đánh dấu trong hộp nhập vào vùng nhớ hệ thống (clipboard).
- **void Paste();** Chèn nội dung văn bản trong clipboard vào hộp nhập bắt đầu từ vị trí đang chọn (hoặc vị trí dấu caret).
- **void Clear();** Xóa nội dung đoạn văn bản đang được đánh dấu.
- **void Cut();** Thực hiện các thao tác **Copy** và **Clear**.
- **void SetPasswordChar (**
 TCHAR ch // Ký tự được chọn
); Đặt ký tự ‘che’ cho nội dung của password trong hộp nhập.
- **int GetLineCount();** Trả về số dòng văn bản trong hộp nhập.
- **int GetLine (**
 int nIndex, // Chỉ số dòng văn bản trong hộp nhập
 LPTSTR lpszBuffer, // Vùng đệm chứa kết quả
 int nMaxLength // Chiều dài vùng đệm
); Lấy nội dung một dòng văn bản trong hộp nhập.
- **int GetFirstVisibleLine();** Trả về chỉ số dòng văn bản được nhìn thấy đầu tiên trong hộp. Các dòng ở trước dòng này trong nội dung văn bản là bị che khuất.

- int **LineFromChar** (
 - int *nIndex* = -1 // Chỉ số ký tự. -1 là ký tự cuối cùng.
); Trả về chỉ số dòng văn bản chứa ký tự (Hộp nhập có nhiều dòng).
- int **LineIndex** (
 - int *nLine* = -1 // Chỉ số dòng văn bản, -1 là dòng cuối cùng
); Trả về chỉ số của ký tự đầu tiên trong dòng văn bản làm tham số.
- int **LineLength** (
 - int *nLine* = -1 // Chỉ số dòng. -1 là dòng đang chứa caret.
); Trả về số ký tự trong nội dung của dòng văn bản.
- CPoint **PosFromChar** (
 - UINT *nChar* // Chỉ số ký tự trong hộp nhập
); Trả về tọa độ điểm ở góc trái trên của ký tự.
- int **CharFromPos** (
 - CPoint *pt* // Tọa độ của điểm
); Trả về chỉ số của ký tự gần điểm đang xét nhất.
- void **LineScroll** (
 - int *nLines*, // Số dòng cuộn dọc, < 0 là cuộn xuống.
 - int *nChars* = 0 // Số ký tự cuộn ngang, < 0 là cuộn trái.
); Cuộn nội dung văn bản đang hiển thị trong hộp nhập.

☞ **Hộp nhập và cửa sổ cha:** Đối tượng hộp nhập có thể gửi message đến cửa sổ cha để thông báo tình hình nhập liệu trong hộp. Cửa sổ cha định hướng xử lý các message thông qua mục xử lý message có dạng như sau:

ON_Notification (*ID*, *memberFxn*)

Trong đó: - *ID* : Số hiệu mục đối tượng edit
 - *memberFxn* : Tên hành vi xử lý message, khai báo như sau:
 afx_msg void **memberFxn**(void);

ON_Notification có thể là một trong các mục xử lý cụ thể sau:

Mục xử lý message	Ý nghĩa của message được xử lý
ON_EN_CHANGE	Nội dung edit đang thay đổi.
ON_EN_HSCROLL	Người dùng click chọn thanh trượt ngang.
ON_EN_VSCROLL	Người dùng click chọn thanh trượt dọc.
ON_EN_KILLFOCUS	Edit chấm dứt hoạt động.
ON_EN_MAXTEXT	Nội dung vượt quá chiều dài cho phép.
ON_EN_SETFOCUS	Edit bắt đầu hoạt động.
ON_EN_UPDATE	Dữ liệu nhập đang được cập nhật cho edit.

☞ **Thực hành 1:** Viết ứng dụng với mục thông báo "Enter your name:" và hộp nhập dữ liệu bên cạnh trên màn hình giao diện chính.

- Tạo dự án VD19 tương tự VD18. Chỉnh sửa CEmpWnd như sau:
- Bổ sung hai thuộc tính protected: **m_staticName** kiểu CStatic và **m_editName** kiểu CEdit.
- Trong hành vi **OnCreate**, thực hiện khởi tạo hai đối tượng này:

```
int CEmpWnd::OnCreate (LPCREATESTRUCT lpCreateStruct)
{
    if (CWnd::OnCreate(lpCreateStruct) == -1)
        return -1;
    m_staticName.Create( _T("Enter your name:"),
                        WS_CHILD | WS_VISIBLE,
                        CRect(10, 75, 130, 95), this );
    m_editName.Create( WS_CHILD | WS_VISIBLE | WS_BORDER,
                      CRect(135, 75, 280, 95), this, _1 );
    return 0;
}
```

☞ **Thực hành 2:** Chỉnh sửa ứng dụng để khi người dùng sửa tên trong hộp nhập thì nội dung này lập tức chuyển lên làm tiêu đề của cửa sổ chính.

HD: Cửa sổ cha có hành vi xử lý message thông báo sự thay đổi dữ liệu trong hộp nhập thông qua mục xử lý message ON_EN_CHANGE.

- Tạo dự án VD20 tương tự VD19, Chỉnh sửa lớp CEmpWnd như sau:
- Bổ sung hành vi xử lý: **afx_msg void OnYourNameChange()**

```
void CEmpWnd::OnYourNameChange ()
{
    CString yourname;
    m_editName.GetWindowText(yourname);
    SetWindowText(yourname);
}
```

- Tạo mới số hiệu resource: IDC_YOURNAME, dùng giá trị này làm số hiệu cho đối tượng **m_editName**. Trong hành vi OnCreate:

```
int CEmpWnd::OnCreate ( LPCREATESTRUCT lpCreateStruct )
{
    if (CWnd::OnCreate(lpCreateStruct) == -1) return -1;
    m_staticName.Create ( _T("Enter your name:"),
                        WS_CHILD | WS_VISIBLE,
                        CRect( 10, 75, 130, 95 ), this );
```

```
m_editName.Create( WS_CHILD | WS_VISIBLE | WS_BORDER,
                  CRect( 135, 75, 280, 95 ), this, IDC_YOURNAME );
return 0;
}
```

- Trong bảng **MessageMap**, bổ sung mục xử lý message:

```
BEGIN_MESSAGE_MAP(CEmpWnd, CWnd)
//{{AFX_MSG_MAP(CEmpWnd)
ON_WM_CREATE()
ON_WM_DESTROY()
ON_WM_PAINT()
//}}AFX_MSG_MAP
ON_EN_CHANGE( IDC_YOURNAME, OnYourNameChange )
END_MESSAGE_MAP()
```

Mục bổ sung

8.3 CButton:

CButton là lớp đối tượng quản lý nút chọn trên cửa sổ giao diện. Trong windows, các nút chọn có thể hoạt động độc lập hay theo nhóm.

- CButton();** Tạo lập đối tượng nút chọn rỗng.
- BOOL Create (**
 LPCTSTR *lpszCaption*, // Nội dung thông báo
 DWORD *dwStyle*, // Thông số dạng nút (BS_PUSHBUTTON)
 const RECT& *rect*, // Tọa độ, kích thước nút chọn
 CWnd* *pParentWnd*, // Con trỏ cửa sổ cha
 UINT *nID* // Số hiệu nút chọn
); Khởi tạo thông số cho đối tượng nút chọn.
- HBITMAP GetBitmap() const;** Handle của ảnh bitmap đang dùng.
- HBITMAP SetBitmap(HBITMAP hBitmap);** Gắn ảnh bitmap vào nút.
- HICON GetIcon() const;** Handle của icon đang dùng.
- HICON SetIcon(HICON hIcon);** Gắn icon vào nút.
- virtual void DrawItem(LPDRAWITEMSTRUCT lpDrawItemStruct);**
 Hành vi kế thừa để thực hiện trang trí riêng trên nút chọn.

- Nút chọn và cửa sổ cha:** Nút chọn có thể gửi message đến cửa sổ cha. Mục **ON_Notification** mà cửa sổ cha dùng xử lý message từ nút chọn là:

Mục xử lý message	Ý nghĩa của message được xử lý
ON_BN_CLICKED	Người dùng click trên nút.
ON_BN_DOUBLECLICKED	Người dùng double-click trên nút

- Thực hành 1:** Viết ứng dụng như VD19. Bổ sung nút chọn "Nhập xong", mà khi chọn, sẽ dùng hộp thông báo để hiển thị nội dung vừa nhập.

- Tạo dự án VD21 như VD19. Chỉnh sửa cho CEmpWnd như sau:
- Bổ sung hành vi xử lý: **afx_msg void OnClickNhapxong()**

```
void CEmpWnd::OnClickNhapxong ()
{
    CString yourname;
    m_editName.GetWindowText( yourname );
    MessageBox ( yourname, "Hello !",
                MB_OK | MB_ICONEXCLAMATION );
}
```

- Khai báo ID resource: IDC_NHAPXONG, bổ sung đối tượng thuộc tính protected **m_buttonNhapxong** kiểu CButton. Trong OnCreate:

```
int CEmpWnd::OnCreate( LPCREATESTRUCT lpCreateStruct )
{
    if (CWnd::OnCreate(lpCreateStruct) == -1)
        return -1;
    m_staticName.Create( _T("Enter your name:"),
                        WS_CHILD | WS_VISIBLE,
                        CRect(10, 75, 130, 95), this );
    m_editName.Create( WS_CHILD | WS_VISIBLE | WS_BORDER,
                      CRect( 135, 75, 280, 95 ), this, -1 );
    m_buttonNhapxong.Create (
        _T("Nhập xong"), WS_CHILD | WS_VISIBLE,
        CRect(10, 115, 90, 140), this, IDC_NHAPXONG );
    return 0;
}
```

- Trong bảng **MessageMap** của CEmpWnd, bổ sung mục xử lý message:

```
ON_BN_CLICKED ( IDC_NHAPXONG , OnClickNhapxong )
```

8.4 CListBox:

CListbox là lớp đối tượng quản lý hộp nhập chứa danh sách các mục chọn trên cửa sổ giao diện. Tùy theo loại listbox mà khi sử dụng, người dùng được phép đánh dấu chọn một hoặc nhiều mục đồng thời trong listbox.

- CListBox();** Tạo lập đối tượng rỗng.

- **BOOL Create** (
 - `DWORD dwStyle,` // Thông số ấn định dạng listbox
 - `const RECT& rect,` // Tọa độ, kích thước của listbox
 - `CWnd* pParentWnd,` // Con trỏ đối tượng cửa sổ cha
 - `UINT nID` // Số hiệu của listbox
); Khởi tạo thông số cho đối tượng ListBox.
`dwStyle`: Gồm thông số qui định đối với control và các dạng bổ sung:
 - `LBS_MULTIPLESEL` : Cho phép chọn nhiều mục đồng thời
 - `LBS_NOTIFY` : Thông tin cho cửa sổ cha
 - `LBS_SORT` : Các mục được xếp thứ tự
 - `LBS_MULTICOLUMN` : Có nhiều cột chứa các mục
 - `LBS_STANDARD` : = `LBS_NOTIFY` | `LBS_SORT`
- `int GetCount`(); Trả về số mục chọn trong Listbox.
- `int GetCurSel`(); Trả về chỉ số của mục được chọn (single).
- `int SetCurSel` (
 - `int nSelect` // Chỉ số mục được chọn
); Ấn định mục chọn trong listbox.
- `int GetSelCount`(); Trả về số mục được chọn trong một listbox. Chỉ dùng cho listbox cho phép chọn nhiều mục.
- `int GetSelItems` (
 - `int nMaxItems,` // Số mục được chọn và
 - `LPINT rgIndex` // mảng chứa các chỉ số của chúng
); Lấy chỉ số của các mục được chọn trong listbox.
- `int AddString` (
 - `LPCTSTR lpszItem` // Thông báo của mục
); Thêm một mục vào listbox. Nếu listbox không có đặc tính xếp thứ tự thì mục mới thêm được đặt vào cuối danh sách các mục.
- `int DeleteString` (
 - `UINT nIndex` // Chỉ số của mục bị xóa
); Xóa một mục trong listbox.
- `int InsertString` (
 - `int nIndex,` // Vị trí chèn, -1 là vị trí cuối listbox
 - `LPCTSTR lpszItem` // Thông báo của mục
); Chèn một mục vào listbox tại vị trí xác định.
- `void ResetContent`(); Xóa tất cả các mục hiện có trong listbox, làm rỗng nội dung listbox quản lý bởi đối tượng.

- `virtual void DrawItem` (
 - `LPDRAWITEMSTRUCT lpDrawItemStruct`
); Hành vi kế thừa nhằm thực hiện trang trí các mục theo cách riêng.
- `virtual void MeasureItem` (
 - `LPMEASUREITEMSTRUCT lpMeasureItemStruct`
); Lấy thông tin làm cơ sở cho việc tự trang trí các mục trong listbox.
- `virtual void DeleteItem` (
 - `LPDELETEITEMSTRUCT lpDeleteItemStruct`
); Hành vi kế thừa để tùy nghi xử lý khi một mục bị xóa khỏi listbox.
- `virtual int VKeyToItem` (
 - `UINT nKey,` // Mã phím (virtual key) được gõ
 - `UINT nIndex` // Chỉ số mục hiện hành trong listbox
); Hành vi kế thừa cho phép tùy nghi xử lý trên phím.
- `virtual int CharToItem` (
 - `UINT nKey,` // Mã phím (character) được gõ
 - `UINT nIndex` // Chỉ số mục hiện hành trong listbox
); Hành vi kế thừa cho phép tùy nghi xử lý phím ký tự.

☞ **Listbox và cửa sổ cha**: ListBox với thông số dạng `LBS_NOTIFY` có thể gửi message đến cửa sổ cha để thông báo tình trạng hoạt động của nó. Mục **ON_Notification** mà cửa sổ cha dùng xử lý message từ listbox là:

Mục xử lý message	Ý nghĩa của message được xử lý
<code>ON_LBN_DBLCLK</code>	Người dùng double-click trên listbox.
<code>ON_LBN_KILLFOCUS</code>	Kết thúc hoạt động nhập.
<code>ON_LBN_SELCANCEL</code>	Hủy bỏ thao tác chọn.
<code>ON_LBN_SELCHANGE</code>	Thay đổi mục chọn trong listbox.
<code>ON_LBN_SETFOCUS</code>	Bắt đầu nhập liệu trên listbox.
<code>ON_WM_CHARTOITEM</code>	Dùng cho ownerDraw listbox.
<code>ON_WM_VKEYTOITEM</code>	Xử lý <code>WM_KEYDOWN</code>

☞ **Thực hành**: Viết ứng dụng tương tự VD21, thêm listbox có ba mục chọn: "Ông A", "Ba B" và "Cô C". Khi người dùng click chọn một mục trong listbox thì nội dung của mục ấy được điền vào hộp nhập Name.
HD: Dùng `ON_LBN_SELCHANGE` định hướng xử lý message liên quan.

- Tạo ứng dụng VD22 như VD21. Chỉnh sửa lớp `CEmpWnd` như sau:
- Bổ sung hành vi protected: `afx_msg void OnSelectDanhsach()`

`void CEmpWnd::OnSelectDanhsach()`

```
{
    CString info;
    int iSel = m_listboxDanhsach.GetCurSel();
    if (iSel == LB_ERR)           // Không có lựa chọn
        iSel = 0;                // Xem như chọn mục đầu tiên
    m_listboxDanhsach.GetText( iSel, info );
    m_editName.SetWindowText( info );
}
```

- Khai báo ID resource: IDC_DANHSACH, thuộc tính protected **m_listboxDanhsach** kiểu CListBox quản lý listbox.
- Hành vi OnCreate thực hiện các khởi tạo cần thiết cho listbox.

```
int CEmpWnd::OnCreate( LPCREATESTRUCT lpCreateStruct )
{
    if (CWnd::OnCreate(lpCreateStruct) == -1) return -1;
    ...
    // Thực hiện tương tự dự án VD21,
    // Và bổ sung các nội dung sau:
    m_listboxDanhsach.Create (
        WS_CHILD | WS_VISIBLE | WS_BORDER |
        LBS_NOTIFY | WS_VSCROLL,
        CRect( 195, 7, 280, 65 ), this, IDC_DANHSACH );
    m_listboxDanhsach.AddString( "Ong A" );
    m_listboxDanhsach.AddString( "Ba B" );
    m_listboxDanhsach.AddString( "Co C" );
    return 0;
}
```

- Trong **MessageMap** của CEmpWnd, bổ sung mục xử lý message:

```
ON_LBN_SELCHANGE( IDC_DANHSACH, OnSelectDanhsach )
```

8.5 CComboBox:

CComboBox là lớp đối tượng quản lý hộp nhập và phần hỗ trợ nhập với danh sách các mục thông tin cho trước. Cách kết hợp giữa hộp nhập và danh sách hỗ trợ tạo thành các dạng khác nhau của comboBox:

- *Drop-list* : Giá trị nhập chỉ được chọn từ danh sách.
- *Simple* : Như *Drop-list* mà danh sách hiển thị thường trực.
- *Drop-down* : Như *Drop-list* và có thể nhập nội dung mới.
- **CComboBox()**; Tạo lập đối tượng rỗng.

- **BOOL Create (**
 DWORD *dwStyle*, // Chứa thông số về dạng của comboBox
 const RECT& *rect*, // Tọa độ, kích thước comboBox
 CWnd* *pParentWnd*, // Con trỏ đối tượng cửa sổ cha
 UINT *nID* // Số hiệu comboBox
); Khởi tạo thông số cho đối tượng comboBox.
 dwStyle: Gồm thông số qui định đối với control và các dạng bổ sung:
 CBS_DROPDOWNLIST : Drop-list comboBox.
 CBS_SIMPLE : Simple comboBox.
 CBS_DROPDOWN : Drop-Down comboBox.
 CBS_SORT : Danh sách các mục xếp thứ tự.
- int **GetCount()**; Trả về số mục chọn trong comboBox.
- int **GetCurSel()**; Trả về chỉ số mục được chọn trong comboBox.
- int **SetCurSel(int *nSelect*);** Ấn định mục chọn trong comboBox.
- void **GetLBText (**
 int *nIndex*, // Chỉ số mục trong comboBox
 CString& *rString* // Biến chứa kết quả
); Lấy nội dung thông báo của một mục trong comboBox.
- void **ShowDropDown (**
 BOOL *bShowIt* = TRUE // TRUE = hiển thị ; FALSE = dấu
); Hiển thị hoặc danh sách chứa các mục hỗ trợ trong comboBox.
- int **AddString(LPCTSTR *lpszString*);** Như listbox.
- int **DeleteString(UINT *nIndex*);** Như listbox.
- int **InsertString(int *nIndex* , LPCTSTR *lpszString*);** Như listbox.
- void **ResetContent()**; Như listbox.
- virtual void **DrawItem (**
 LPDRAWITEMSTRUCT *lpDrawItemStruct*
); Như listbox.
- virtual void **MeasureItem (**
 LPMEASUREITEMSTRUCT *lpMeasureItemStruct*
); Như listbox.
- virtual void **DeleteItem (**
 LPDELETEITEMSTRUCT *lpDeleteItemStruct*
); Như listbox.

- ✎ **ComboBox và cửa sổ cha:** Đối tượng comboBox có thể gửi message đến cửa sổ cha để thông báo tình hình hoạt động của nó. Mục xử lý message **ON_Notification** mà cửa sổ cha dùng xử lý message từ comboBox là:

Mục xử lý message	Ý nghĩa của message được xử lý
ON_CBN_CLOSEUP	Danh sách vừa được đóng lại
ON_CBN_DBLCLK	Double-click trên mục chọn
ON_CBN_DROPDOWN	Danh sách vừa được mở ra
ON_CBN_EDITCHANGE	Nội dung hộp nhập đang thay đổi
ON_CBN_EDITUPDATE	Sắp cập nhật nội dung hộp nhập
ON_CBN_SELENDCANCEL	Bỏ qua việc thay đổi mục chọn.
ON_CBN_SELENDOK	Một mục trong danh sách được chọn
ON_CBN_KILLFOCUS	Kết thúc nhập liệu trong comboBox
ON_CBN_SELCHANGE	Thay đổi mục chọn trong danh sách
ON_CBN_SETFOCUS	Bắt đầu nhập liệu trong comboBox

- ✎ **Thực hành:** Viết ứng dụng tương tự VD22, Bổ sung combobox kiểu dropdown có ba mục chọn: "Vo van A", "Nguyen thi B" và "Tran thi C". Khi chọn trên combobox, nội dung chọn được điền vào hộp nhập Name.
HD: Dùng ON_CBN_SELCHANGE định hướng xử lý message liên quan.

- Tạo dự án VD23 tương tự VD22. Chỉnh sửa lớp CEmpWnd như sau:
- Bổ sung hành vi protected: **afx_msg void OnSelectCombobox ()**

```
void CEmpWnd::OnSelectCombobox()
{
    CString info;
    int iSel = m_Combobox.GetCurSel( );
    if (iSel == LB_ERR)                // no selection
        iSel = 0;
    m_Combobox.GetLBText( iSel, info );
    m_editName.SetWindowText( info );
}
```

- Khai báo ID resource: IDC_COMBOBOX. Khai báo đối tượng thuộc tính protected **m_Combobox** kiểu CComboBox quản lý comboBox.
- Hành vi OnCreate thực hiện các khởi tạo cần thiết cho comboBox:

```
int CEmpWnd::OnCreate (LPCREATESTRUCT lpCreateStruct)
{
    if (CWnd::OnCreate(lpCreateStruct) == -1) return -1;
```

```
...                // Như VD22, bổ sung các nội dung sau:
m_Combobox.Create (
    WS_CHILD | WS_VISIBLE | WS_BORDER |
    CBS_DROPDOWNLIST,
    CRect( 135, 100, 280, 225 ), this,
    IDC_COMBOBOX );
m_Combobox.AddString( "Vo van A" );
m_Combobox.AddString( "Nguyen thi B" );
m_Combobox.AddString( "Tran thi C" );
return 0;
}
```

- Trong MessageMap của CEmpWnd, khai báo mục xử lý message:

```
ON_CBN_SELCHANGE( IDC_COMBOBOX, OnSelectCombobox )
```

8.6 CSpinButtonCtrl:

CSpinButtonCtrl là lớp đối tượng quản lý nút tăng-giảm trên cửa sổ giao diện. Nút tăng-giảm được dùng hỗ trợ cho các hộp nhập số nguyên.

- **CSpinButtonCtrl();** Tạo lập đối tượng rỗng.
- **BOOL Create (**
 DWORD dwStyle, // Chứa thông số dạng nút tăng giảm
 const RECT& rect, // Tọa độ, kích thước. Không bắt buộc.
 CWnd* pParentWnd, // Con trỏ đối tượng cửa sổ cha
 UINT nID // Số hiệu nút tăng giảm
); Khởi tạo thông số cho đối tượng nút tăng giảm.
 dwStyle: Gồm thông số qui định đối với control và các dạng bổ sung:
 UDS_HORZ : Hai nút tăng-giảm nằm ngang
 UDS_SETBUDDYINT : Dùng cho hộp nhập số nguyên.
 UDS_ALIGNRIGHT : Nằm bên phải hộp nhập.
 UDS_ALIGNLEFT : Nằm bên trái hộp nhập.
 UDS_ARROWKEYS : Dùng các phím ↑, ↓ để tăng-giảm.
- **CWnd* SetBuddy (**
 CWnd* pWndBuddy // Con trỏ đối tượng được hỗ trợ.
); Ấn định đối tượng hộp nhập mà nút tăng-giảm này sẽ hỗ trợ. Hành vi trả về con trỏ đối tượng hộp nhập được hỗ trợ trước đó.
- **CWnd* GetBuddy();** Trả về con trỏ đối tượng hộp nhập được hỗ trợ.

- void **SetRange** (
 - int nLower, // Giá trị cận dưới
 - int nUpper // Giá trị cận trên.
); Ấn định giới hạn biến thiên của nút tăng-giảm.
- void **GetRange** (
 - int &lower, // Tham biến chứa giá trị cận dưới
 - int& upper // Tham biến chứa giá trị cận trên
); Lấy thông tin về giới hạn của nút tăng-giảm.
- int **SetPos**(int nPos); Đặt giá trị cho nút tăng-giảm.
- int **GetPos**(); Trả về giá trị hiện tại của nút tăng-giảm.

📖 **Thực hành:** Viết ứng dụng tương tự VD21; mục thông báo "Enter your age:" và hộp nhập tuổi. Hộp nhập được hỗ trợ của nút spin có giới hạn 15-250. Nút chọn "Nhập xong" cho phép hiển thị tên và tuổi vừa nhập.

- Tạo dự án VD24 tương tự VD21. Chỉnh sửa lớp CEmpWnd như sau:
- Khai báo thuộc tính **m_editAge** kiểu CEdit quản lý hộp nhập.
- Hành vi OnCreate thực hiện khởi tạo thông số cho các mục. Riêng mục thông báo và nút spin, do không có yêu cầu truy xuất nên các đối tượng quản lý chúng sẽ được khai báo static trong OnCreate:

```
int CEmpWnd::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CWnd::OnCreate(lpCreateStruct) == -1) return -1;
    ... // Tương tự VD21, và các nội dung bổ sung sau:
    m_editAge.Create (
        WS_CHILD | WS_VISIBLE | WS_BORDER | ES_NUMBER,
        CRect( 135, 45, 190, 65 ), this, -1 );
    static CStatic m_staticAge;
    static CSpinButtonCtrl m_spinAge;
    m_staticAge.Create ( _T( "Enter your age:" ),
        WS_CHILD | WS_VISIBLE,
        CRect( 10, 45, 130, 65 ), this );
    m_spinAge.Create ( WS_CHILD | WS_VISIBLE |
        UDS_ARROWKEYS | UDS_SETBUDDYINT | UDS_ALIGNRIGHT,
        CRect(0,0,1,1), this, -1 );
    m_spinAge.SetBase( 1 ); m_spinAge.SetRange( 15, 250 );
    m_spinAge.SetBuddy ( &m_editAge );
    return 0;
}
```

- Hành vi **OnClickNhapxong** thực hiện hiển thị thông tin nhập:

```
void CEmpWnd::OnClickNhapxong( )
{
    CString yourname, yourage, mess;
    m_editName.GetWindowText( yourname );
    m_editAge.GetWindowText( yourage );
    mess.Format ( "%s\n is %s years old", yourname, yourage );
    MessageBox( mess, "Hello!", MB_OK | MB_ICONEXCLAMATION );
}
```

8.7 CProgressCtrl:

CProgressCtrl là lớp đối tượng quản lý thanh tiến độ xử lý trên giao diện.

- **CProgressCtrl**(); Tạo lập đối tượng rỗng.
- **BOOL Create** (
 - DWORD dwStyle, // Thông số dạng thanh tiến độ xử lý
 - const RECT& rect, // Toạ độ, kích thước của thanh
 - CWnd* pParentWnd, // Con trỏ đối tượng cửa sổ cha
 - UINT nID // Số hiệu của thanh tiến độ
); Hành vi khởi tạo thông số cho thanh tiến độ trên giao diện.
 dwStyle: Gồm thông số qui định đối với control và các dạng bổ sung:
 - PBS_VERTICAL : Thanh tiến độ nằm thẳng đứng.
 - PBS_SMOOTH : Chỉ mục tiến độ là dải màu (xanh) liên tục.
- void **SetRange** (
 - short nLower, // Giá trị nhỏ nhất
 - short nUpper // Giá trị lớn nhất
); Ấn định giới hạn tiến độ xử lý chỉ bởi thanh tiến độ.
- void **GetRange**(
 - int& nLower, // Biến chứa giá trị nhỏ nhất
 - int& nUpper // Biến chứa giá trị lớn nhất
); Lấy thông tin về giới hạn tiến độ xử lý chỉ bởi thanh tiến độ.
- int **SetPos**(int nPos); Ấn định vị trí hiện tại của thanh tiến độ.
- int **GetPos**(); Trả về vị trí hiện tại của thanh tiến độ xử lý.
- int **OffsetPos**(int nPos); Ấn định mức tăng (giảm) trên một đơn vị của chỉ mục tiến độ trong thanh tiến độ bằng nPos, đồng thời cập nhật trạng thái hiển thị của thanh tiến độ theo giá trị ấn định mới.
- int **StepIt**(); Thay đổi chỉ mục tiến độ một đơn vị.

8.8 CScrollBar:

CScrollBar là lớp đối tượng quản lý thanh cuộn trên cửa sổ giao diện.

- **CScrollBar();** Tạo lập đối tượng rỗng.

- **BOOL Create (**

```
    DWORD dwStyle,           // Thông số dạng của thanh cuộn
    const RECT& rect,         // Tọa độ, kích thước của thanh cuộn
    CWnd* pParentWnd,        // Con trỏ đối tượng cửa sổ cha
    UINT nID                  // Số hiệu thanh cuộn
```

); Khởi tạo thông số cho đối tượng thanh cuộn trên giao diện.

dwStyle: Gồm thông số qui định đối với control và các dạng bổ sung:

SBS_HORZ : Thanh cuộn đặt ngang (horizontal)

SBS_VERT : Thanh cuộn đặt đứng (vertical)

- **int GetScrollPos();** Trả về vị trí hiện thời của một thanh cuộn.

- **int SetScrollPos (**

```
    int nPos,                // Vị trí đặt nút cuộn
    BOOL bRedraw = TRUE      // Cập nhật lại hình ảnh thanh cuộn
```

); Đặt vị trí nút cuộn trên thanh cuộn.

- **void SetScrollRange (**

```
    int nMinPos,             // Giá trị nhỏ nhất
    int nMaxPos,             // Giá trị lớn nhất
    BOOL bRedraw = TRUE      // Cập nhật lại thanh cuộn
```

); Ấn định giới hạn vị trí đầu và cuối của nút cuộn trên thanh cuộn.

- **void GetScrollRange (**

```
    LPINT lpMinPos,          // Chứa vị trí đầu
    LPINT lpMaxPos           // Chứa vị trí cuối
```

); Lấy giá trị giới hạn vị trí đầu và cuối của nút cuộn trên thanh cuộn.

- **void ShowScrollBar (BOOL bShow = TRUE);** Bật / Tắt thanh cuộn.

☞ **ScrollBar và cửa sổ cha:** Đối tượng scrollbar gửi WM_HSCROLL (đối với loại scrollbar ngang) hoặc WM_VSCROLL (đối với loại scrollbar đứng) đến cửa sổ cha để thông báo trạng thái hiện thời của nó. Các hành vi OnVScroll hoặc OnHScroll của đối tượng CWnd quản lý cửa sổ cha sẽ xử lý các message tương ứng nói trên.

☞ Thực hiện ứng dụng như VD15. Bổ sung thanh cuộn, và mục thông báo để hiển thị vị trí nút cuộn khi người dùng tác động lên thanh cuộn.

- Tạo dự án VD24A như dự án VD15. Chỉnh sửa lớp CEmpWnd như sau:

- Khai báo các thuộc tính protected: **m_staticScroll** kiểu CStatic quản lý mục thông báo, **m_Scroll** kiểu CScrollBar quản lý thanh cuộn.

- Hành vi OnCreate thực hiện khởi tạo thông số cho các mục:

```
int CEmpWnd::OnCreate( LPCREATESTRUCT lpCreateStruct )
{
    if (CWnd::OnCreate(lpCreateStruct) == -1) return -1;

    m_staticScroll.Create( _T(""),
                          WS_CHILD | WS_VISIBLE,
                          CRect( 10, 15, 120, 35 ), this );
    m_Scroll.Create( WS_CHILD | WS_VISIBLE | SBS_HORZ,
                   CRect( 10, 45, 127, 65 ), this, IDC_SCROLLBAR );
    m_Scroll.SetScrollRange( 0, 100 );
    return 0;
}
```

- Hành vi OnHScroll xử lý WM_HSCROLL:

```
void CEmpWnd::OnHScroll( UINT nSBCode, UINT nPos,
                       CScrollBar* pScrollBar )
{
    char s[10];
    int pos = nPos;
    switch (nSBCode) {
        case SB_LINELEFT:
            pos = pScrollBar->GetScrollPos() - 1;
            if (pos < 0) pos = 0;
            pScrollBar->SetScrollPos( pos );
            sprintf( s, "%d", pos ); m_staticScroll.SetWindowText(s);
            break;
        case SB_LINERIGHT:
            pos = pScrollBar->GetScrollPos() + 1;
            if (pos > 90) pos = 100;
            pScrollBar->SetScrollPos(pos);
            sprintf(s, "%d", pos); m_staticScroll.SetWindowText(s);
            break;
        case SB_THUMBPOSITION:
            pScrollBar->SetScrollPos(pos);
            sprintf(s, "%d", pos); m_staticScroll.SetWindowText(s);
            break;
        default: CWnd::OnHScroll(nSBCode, nPos, pScrollBar);
    }
}
```

- Biên dịch và chạy thử ứng dụng.

8.9 CSliderCtrl:

CSliderCtrl là lớp đối tượng quản lý thanh trượt trên cửa sổ giao diện.

- **CSliderCtrl** (); Tạo lập đối tượng rỗng.

- **BOOL Create** (

```
    DWORD dwStyle,           // Thông số dạng của thanh trượt.  
    const RECT& rect,         // Tọa độ, kích thước của thanh  
    CWnd* pParentWnd,        // Con trỏ đối tượng cửa sổ cha  
    UINT nID                  // Số hiệu thanh trượt
```

); Khởi tạo thông số cho đối tượng thanh trượt trên giao diện.

dwStyle: Gồm thông số qui định đối với control và các dạng bổ sung:

```
TBS_HORZ           : Thanh trượt ngang  
TBS_VERT           : Thanh trượt đứng  
TBS_NOTICKS        : Thanh trượt không có thước làm mốc.  
TBS_LEFT, TBS_RIGHT : Thước đặt trái/phải thanh trượt đứng  
TBS_TOP, TBS_BOTTOM : Thước đặt trên/dưới thanh trượt ngang  
TBS_BOTH           : Thước đặt ở cả hai phía thanh trượt.
```

- **int GetPos**(); Trả về vị trí hiện thời của nút trượt trên thanh trượt.

int SetPos(**int** *nPos*); Đặt vị trí nút trượt trên thanh trượt.

- **void SetRange** (

```
    int nMinPos,              // Giá trị nhỏ nhất  
    int nMaxPos,              // Giá trị lớn nhất  
    BOOL bRedraw = TRUE      // Cập nhật lại thanh cuộn
```

); Ấn định giới hạn nhỏ nhất và lớn nhất của thanh trượt.

- **void GetRange** (

```
    LPINT lpMinPos,          // Lấy giá trị nhỏ nhất  
    LPINT lpMaxPos          // Lấy giá trị lớn nhất
```

); Lấy giá trị giới hạn nhỏ nhất và lớn nhất của thanh trượt.

☞ **SliderCtrl và cửa sổ cha**: Tương tự ScrollBar.

THỰC HÀNH:

1. Viết ứng dụng như VD18. Tạo 5 icon resource có nội dung phối hợp hoạt hình trong resource của ứng dụng. Cài đặt timer cho CEmpWnd để thực hiện đổi icon cho đối tượng *m_staticIcon* tuần tự theo thời gian với 5 icon nói trên.
2. Viết ứng dụng với thanh ProgressBar chạy theo thời gian. Khi progressbar đầy thì kết thúc ứng dụng.
3. Thực hiện dự án tương tự VD24A cho thanh trượt.

Hộp hội thoại

9.1 HỘP HỘI THOẠI (DIALOG):

Dialog là cửa sổ giao diện với các đặc điểm trang trí được mô tả thông qua một cấu trúc độc lập. Cấu trúc này được lưu trữ trong resource của ứng dụng, gọi là dialog resource. Có hai kiểu hoạt động của dialog:

- *Dialog khóa (modal dialog):* Tác vụ thực hiện dialog phải chờ đến khi dialog chấm dứt hoạt động.
- *Dialog không khóa (modeless dialog):* Tác vụ thực hiện dialog có thể tiếp tục các xử lý tiếp theo ngay sau lệnh thực hiện dialog.

9.2 LỚP CDialog:

CDialog là lớp đối tượng kế thừa từ lớp CWnd, được sử dụng để quản lý dialog. Ngoài các thuộc tính và hành vi kế thừa public từ lớp CWnd, lớp CDialog có các hành vi bổ sung sau đây:

- **CDialog** (
 - UINT *nIDTemplate*, // Số hiệu của dialog resource
 - CWnd* *pParentWnd* = NULL // Con trỏ đối tượng cửa sổ cha
); Tạo lập đối tượng dialog hoạt động kiểu khóa.
- **CDialog**(); Tạo lập đối tượng dialog hoạt động kiểu không khóa.
- virtual BOOL **OnInitDialog**(); Hành vi mà đối tượng dialog sử dụng để xử lý WM_INITDIALOG, message do windows gửi đến dialog trước khi kích hoạt dialog. Kế thừa hành vi này nhằm thực hiện khởi tạo thông số cho dialog và các controls của nó.
- afx_msg HBRUSH **OnCtlColor** (
 - CDC* *pDC*, // Con trỏ đối tượng DC của control
 - CWnd* *pWnd*, // Con trỏ đối tượng cửa sổ quản lý control
 - UINT *nCtlColor* // Thông tin về loại control được trang trí
); Hành vi xử lý WM_CTLCOLOR, message do windows gửi đến dialog khi có nhu cầu trang trí một control trên dialog.
nCtlColor có thể là một trong các giá trị sau:
 - CTLCOLOR_BTN : Đối tượng trang trí là button
 - CTLCOLOR_DLG : Dialog
 - CTLCOLOR_EDIT : Hộp nhập liệu

CTLCOLOR_LISTBOX : Listbox
 CTLCOLOR_SCROLLBAR : Thanh trượt
 CTLCOLOR_STATIC : Mục thông báo

- virtual int **DoModal**(); Kích hoạt dialog kiểu khóa. Hành vi chỉ kết thúc khi dialog kết thúc hoạt động và giá trị trả về của nó là tham số của hành vi EndDialog được dùng để kết thúc dialog.
- void **EndDialog** (
 - int *nResult* // Giá trị tham số của hành vi.
); Hành vi được dùng để kết thúc hoạt động của dialog kiểu khóa.
- virtual void **OnOK**(); Hành vi kế thừa để cài đặt xử lý chọn button có số hiệu IDOK. Trong CDialog, phần cài đặt của OnOK chỉ là lệnh gọi hành vi EndDialog với tham số IDOK.
- virtual void **OnCancel**(); Hành vi kế thừa để cài đặt xử lý chọn button có số hiệu IDCANCEL (phím tắt là ESC). Trong CDialog, cài đặt của OnCancel chỉ là lệnh gọi hành vi EndDialog với tham số IDCANCEL.
- BOOL **Create** (
 - UINT *nIDTemplate*, // Số hiệu dialog resource
 - CWnd* *pParentWnd* = NULL // Con trỏ đối tượng cửa sổ cha.
); Hành vi kích hoạt dialog kiểu không khóa.
- virtual BOOL **DestroyWindow**(); Hành vi được dùng để kết thúc hoạt động của dialog kiểu không khóa.
- CWnd* **GetDlgItem** (
 - int *nID* // Số hiệu của control trên dialog
); Trả về con trỏ đối tượng thuộc lớp CWnd quản lý control.
- int **GetDlgItemText** (
 - int *nID*, // Số hiệu của control trên dialog
 - CString& *rsString* // Tham biến kiểu chuỗi chứa kết quả
); Lấy nội dung chuỗi thông báo của control.
- UINT **GetDlgItemInt** (
 - int *nID* // Số hiệu của control trên dialog
); Trả về giá trị số của nội dung nhập trong control.
- void **SetDlgItemText** (
 - int *nID*, // Số hiệu của control trên dialog
 - LPCTSTR *lpszString* // Giá trị kiểu chuỗi
); Đặt nội dung kiểu chuỗi cho control.

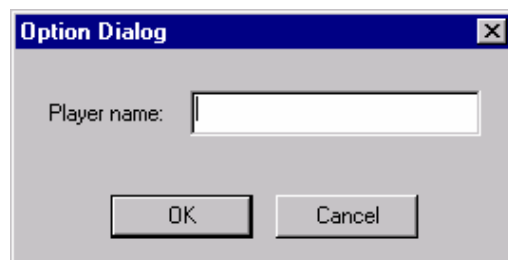
- void **SetDlgItemInt** (
 int *nID*, UINT *nValue* // Số hiệu control và giá trị gán
); Gán giá trị số *nValue* cho control.
- UINT **IsDlgButtonChecked** (
 int *nIDButton* // Số hiệu của button control
); Trả về giá trị TRUE nếu control được đánh dấu chọn.
- int **GetCheckedRadioButton** (
 int *nIDFirstButton*, // Số hiệu nút RadioButton đầu tiên và
 int *nIDLastButton* // cuối cùng trong nhóm các nút RadioButtons
); Trả về số hiệu nút radioButton được đánh dấu chọn.
- void **CheckRadioButton** (
 int *nIDFirstButton*, // Số hiệu nút RadioButton đầu tiên và
 int *nIDLastButton*, // cuối cùng trong nhóm các nút RadioButtons
 int *nIDCheckButton* // Số hiệu nút RadioButton được đánh dấu.
); Đánh dấu chọn một nút trong nhóm các nút RadioButtons.

9.3 TẠO VÀ SỬ DỤNG DIALOG TRONG CHƯƠNG TRÌNH:

Mỗi đối tượng dialog hình thành trong chương trình là kết quả kết hợp giữa lớp đối tượng kế thừa từ CDialog và dialog resource. Như vậy, để sử dụng dialog trong chương trình, ta phải thực hiện hai bước sau:

- Thiết kế dialog resource.
- Khai báo lớp kế thừa từ CDialog sử dụng dialog resource nói trên.

Trong chương trình, mỗi khi có nhu cầu sử dụng dialog, ta chỉ việc khai báo biến đối tượng thuộc lớp nói trên và sử dụng nó một cách thích hợp.

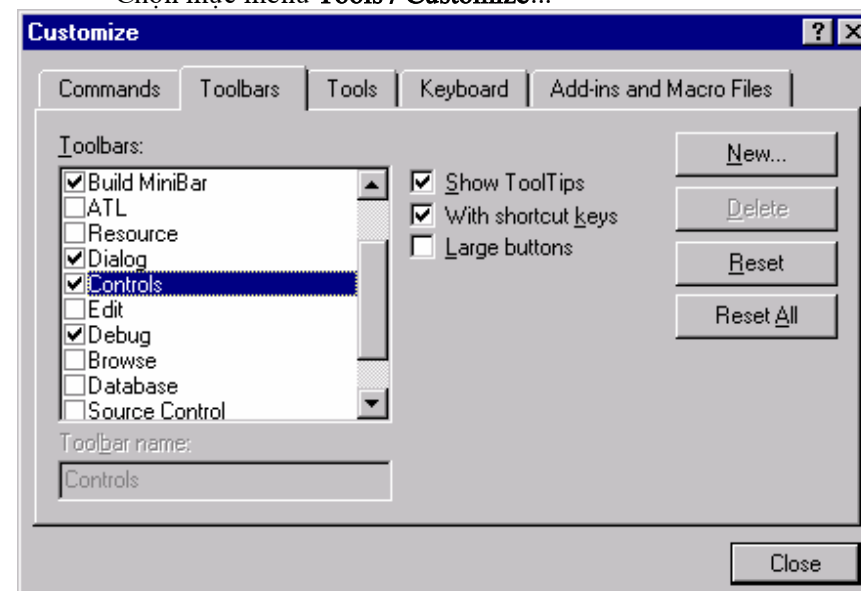


- ☞ Trong phần này, ta thực hiện ứng dụng tương tự VD21. Khi người dùng chọn mục **Option** trên menu thì hiển thị dialog như hình vẽ trên. Trước tiên, tạo dự án mới VD25 tương tự dự án VD21. Sau đó thực hiện các công việc sau:

9.3.1 Tạo Dialog resource:

- Tạo mới dialog resource: Thực hiện tương tự việc tạo mới icon (2.8).
Lưu ý: Chọn **Resource Type** là **Dialog**.
- Đặt số hiệu cho dialog resource (ví dụ IDD_OPTION).

- Thiết kế dialog thông qua màn hình thiết kế mà ta vừa nhận được từ bước trên. Các thao tác cơ bản như sau:
 - Bật / Tắt thanh công cụ hỗ trợ thiết kế dialog:
 - Chọn mục menu **Tools / Customize...**

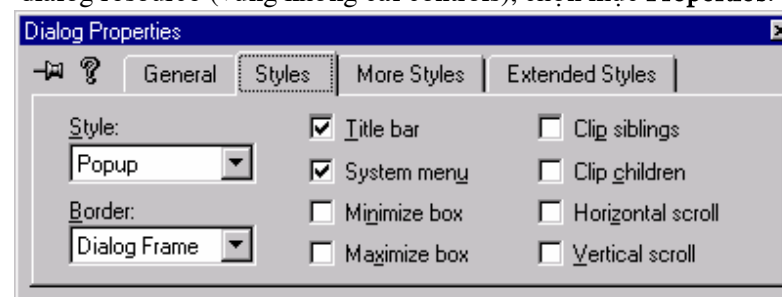


- Đánh dấu hoặc bỏ đánh dấu mục **Controls**. Chọn **Close**.



(Thanh công cụ với các loại control sử dụng được trên dialog)


- Ấn định các đặc tính của dialog resource: Right-click trên khung dialog resource (vùng không cài controls), chọn mục **Properties**:












- Chọn **General** để ấn định các thông số chung.

- *ID* : Số hiệu của dialog resource.
- *Caption* : Nội dung tiêu đề của dialog.
- *Menu* : Số hiệu của menu resource gắn vào dialog.
- *Font* : Ấn định font dùng cho nội dung chữ trên dialog.
- Chọn **Styles** để ấn định thông số dạng cửa sổ của dialog:
 - *Style*: Đặc tính của dialog. Chẳng hạn chọn *Popup* cho phép dialog tạo ra có thông số dạng WS_POPUP.
 - *Border*: Kiểu đường viền của dialog.
 - *Title Bar*: Dialog có tiêu đề.
 - *System Menu*: Dialog có hộp System Menu.
- Chọn **More Styles** để ấn định các thông số dạng mở rộng:
 - *Visible*: Dialog hiển thị. Đối với dialog resource dùng cho modeless dialog thì mục này phải luôn được chọn.
- Chọn **Extended Style** để ấn định các thông số dạng mở rộng:
 - *ToolWindow*: Tương ứng WS_EX_TOOLWINDOW.
 - *Static edge*: Vùng client được chìm xuống (3D).

Gõ phím **Enter** để kết thúc.

- Cài đặt đối tượng mục thông báo lên dialog:
 - Click chọn biểu tượng  trên thanh công cụ.
 - Drag chuột trên vùng dành cho mục thông báo trên dialog.
 - Right-click trên đối tượng mục thông báo vừa cài đặt, chọn mục **Properties**. Thực hiện các ấn định cần thiết:
 - *ID*: Số hiệu mục thông báo, mặc nhiên IDC_STATIC. Để dialog nhận diện được mục khi xử lý message thì giá trị này phải được khai báo tường minh và duy nhất.
 - *Caption*: chuỗi thông báo
- Style**:
 - *Visible*: Nếu đánh dấu thì mục được hiển thị.
 - *Align Text*: Canh chỉnh nội dung thông báo trong mục.
 - *Center Vertically*: Canh chỉnh giữa nội dung thông báo theo chiều dọc trong mục.
 - *Border*: Có khung bao quanh mục thông báo.
 - *Sunken*: Khung chìm.
 - *Notify*: Mục thông báo có khả năng thông tin cho cửa sổ cha.
- Extended style**:
 - *Transparent*: Nền mục trong suốt.

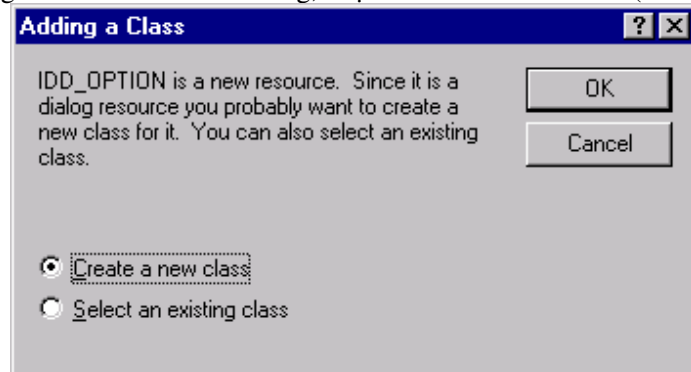
- *Static edge*: Khung chìm.
- Cài đặt đối tượng hộp nhập lên dialog:
 - Click chọn biểu tượng  trên thanh công cụ.
 - Thực hiện cài đặt và ấn định thông số như với mục thông báo:
 - *Multi Line*: Hộp nhập cho phép nhập nhiều dòng.
 - *AutoHScroll/AutoVScroll*: Tự động trượt nội dung khi thông tin nhập vượt quá kích thước hộp nhập.
 - *HorizontalScroll / VerticalScroll*: Hiển thị thanh trượt ngang, dọc của hộp nhập.
 - *Want return*: Sử dụng phím enter để xuống dòng trong hộp nhập nhiều dòng.
 - *Password*: Dùng nhập password.
- Cài đặt đối tượng comboBox lên dialog:
 - Click chọn biểu tượng  trên thanh công cụ.
 - Thực hiện cài đặt và ấn định thông số như với mục thông báo:
 - *Data*: Chứa các mục chọn. Các mục này được nhập trên các dòng khác nhau. Lưu ý dùng phím Ctrl+Enter để xuống dòng.
 - *Type*: Kiểu comboBox.
 - *Sort*: Các mục trong comboBox được sắp xếp theo nội dung.
- Cài đặt đối tượng comboBox lên dialog:
 - Click chọn biểu tượng  trên thanh công cụ.
 - Thực hiện cài đặt và ấn định thông số như với mục thông báo:
 - *Selection*: Kiểu listbox.
 - *Multi-column*: Listbox có nhiều cột.
 - *Want Key Input*: Listbox cho phép xử lý phím.
- Cài đặt đối tượng button lên dialog:
 - Click chọn biểu tượng  trên thanh công cụ.
 - Thực hiện cài đặt và ấn định thông số như với mục thông báo:
 - *Default Button*: Button ứng với phím tắt là Enter.
 - *Multi-lines*: Nội dung thông báo của button có nhiều dòng.
 - *Notify*: Button có khả năng thông tin cho cửa sổ cha.
- Đánh dấu chọn các đối tượng trên dialog: Thực hiện thao tác click. Phối hợp phím Shift hoặc Ctrl để đánh dấu nhiều đối tượng.
- Chỉnh vị trí của một đối tượng: Thực hiện thao tác drag đối tượng.
- Chỉnh kích thước của một đối tượng:
 - Click chọn đối tượng.

- Thực hiện thao tác *Drag* trên biên của đối tượng để đạt kích thước mong muốn. 
- Canh chỉnh vị trí, kích thước một nhóm đối tượng:
 - Đánh dấu nhóm đối tượng
 - Chọn công cụ phù hợp trên thanh công cụ:
 - Canh thẳng theo biên: 
 - Canh thẳng giữa dialog: 
 - Các đều nhau: 
 - Bằng cỡ đối tượng chọn cuối cùng trong nhóm: 

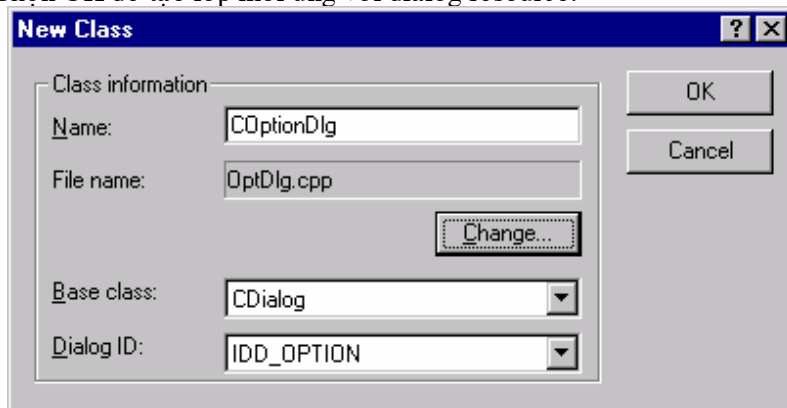
9.3.2 Khai báo lớp kế thừa CDialog sử dụng dialog resource:

Thông qua lớp này, ta thực hiện cài đặt các xử lý phù hợp trên dialog và các đối tượng nhập liệu được mô tả trong dialog resource ở trên.

- Trong màn hình thiết kế dialog, chọn **View / ClassWizard** (Ctrl+W).



- Chọn OK để tạo lớp mới ứng với dialog resource.



- **Name** = *COptionDlg* : Nhập tên lớp mới

- **File name** = *OptDlg* : Tên tập tin,
 - **Base class** = *CDialog* : Lớp cơ sở
 - **Dialog ID** = *IDD_OPTION* : Số hiệu dialog resource.
- Sau cùng chọn **OK**.

- Đóng màn hình thiết kế dialog. Trong màn hình **Workspace**, chọn **ClassView**, ta có lớp COptionDlg trong danh sách các lớp của dự án. Trên lớp COptionDlg, ta có thể thực hiện bổ sung thuộc tính, cài đặt các hành vi thông thường cũng như các hành vi xử lý message. Các thao tác hoàn toàn tương tự như đã thực hiện với các lớp CEmpApp và CEmpWnd.

9.3.3 Sử dụng dialog trong chương trình:

- Khai báo đối tượng thuộc lớp dialog mới tạo. Dùng chỉ thị `#include` tập tin (.H) chứa khai báo lớp ở đầu chương trình:

```
#include "Optdlg.h"           // EmpWnd.cpp : implement file
...
COptionDlg dlg (this);       // Đối tượng COptionDlg
```

- Gọi hành vi **DoModal** hoặc **Create** của đối tượng dialog một cách phù hợp tùy theo yêu cầu dùng dialog khóa hay không khóa.

```
dlg.DoModal();               // Dialog hoạt động ở chế độ khóa
```

- **Thực hành:** Bổ sung dự án VD25: Cài đặt hành vi xử lý mục chọn Option trên menu cho CEmpWnd với nội dung thực hiện dialog COptionDlg.

- Bổ sung hành vi xử lý mục chọn Option cho lớp CEmpWnd.
- Trong phần cài đặt hành vi này, ta khai báo đối tượng COptionDlg và gọi hành vi DoModal của nó:

```
void CEmpWnd::OnGameOption ()
{
    COptionDlg dlg(this);
    dlg.DoModal();
}
```

9.4 LIÊN KẾT GIỮA DIALOG VÀ CÁC THÀNH PHẦN KHÁC:

Dialog là công cụ giao diện rất tiện lợi với người dùng. Tạo mối liên kết giữa dialog và các thành phần khác của ứng dụng để trao đổi thông tin từ người dùng là vấn đề thường gặp. Vấn đề này có thể giải quyết như sau:

- Đối tượng nhận thông tin sẽ chuyển con trỏ (handle) quản lý mình cho đối tượng cung cấp thông tin.

- Đối tượng cung cấp thông tin sẽ dùng handle của đối tượng nhận tin để xác định các ô chứa tin của đối tượng này, sau đó điền các thông tin của mình vào các ô chứa tin của đối tượng nhận tin.

🔗 **Thực hành:** Tạo ứng dụng VD26 như VD25. Thực hiện các bổ sung: Khi option dialog hoạt động, người dùng gõ thông tin vào hộp nhập. Nếu người dùng chọn OK thì thông tin nhập chuyển vào hộp nhập YourName trên cửa sổ chính, ngược lại mục chọn Cancel sẽ không xử lý gì cả.

HD: Mục nhập YourName được quản lý bởi thuộc tính **m_editName** của đối tượng CEmpWnd, đây là ô chứa tin của đối tượng nhận tin CEmpWnd. Đối tượng COptionDlg là đối tượng cung cấp tin. Xử lý điền thông tin chỉ xảy ra khi người dùng chọn **OK**. Xử lý này được cài đặt trong hành vi mà đối tượng COptionDlg dùng để trả lời thao tác click trên nút **OK**.

Các bước thực hiện dự án VD26:

- Tạo dự án VD26 tương tự dự án VD25.
- Hành vi thực hiện OptionDlg trong CEmpWnd sử dụng dùng con trỏ chỉ đến nó làm tham số cho hành vi tạo lập của đối tượng COptionDlg:

```
void CEmpWnd::OnGameOption ()
{
    COptionDlg dlg( this );
    dlg.DoModal();
}
```

- Bổ sung thuộc tính protected **m_parent** kiểu con trỏ CEmpWnd* cho lớp COptionDlg. Thuộc tính này được dùng để chứa con trỏ đến cửa sổ cha của COptionDlg (CEmpWnd). Bổ sung #include "EmpWnd.h" vào đầu tập tin khai báo (.h) của lớp COptionDlg.
- Hành vi tạo lập của COptionDlg lưu giữ giá trị con trỏ cửa sổ cha được truyền cho nó vào thuộc tính **m_parent**.

```
COptionDlg::COptionDlg ( CWnd* pParent )
                        : CDialog( COptionDlg::IDD, pParent )
{
    m_parent = (CEmpWnd*) pParent;
   //{{AFX_DATA_INIT(COptionDlg)
    // NOTE: the ClassWizard will add member initialization here
   //}}AFX_DATA_INIT
}
```

- Hành vi xử lý chọn nút **OK** của lớp COptionDlg: Thực hiện việc lấy thông tin từ hộp nhập của nó để gán cho đối tượng hộp nhập **m_editName** trên cửa sổ cha CEmpWnd.

```
void COptionDlg::OnOK()
{
    CString name;
    // Lấy thông tin nhập trong hộp nhập của dialog
    GetDlgItemText( IDC_NAME, name );
    // và chuyển thông tin này sang hộp nhập trên cửa sổ chính
    m_parent->m_editName.SetWindowText( name );
    // Dùng hành vi lớp cơ sở để kết thúc dialog.
    CDialog::OnOK();
}
```

- 🔗 Hành vi **OnOK** của lớp COptionDlg bị lỗi do truy xuất thuộc tính kiểu protected **m_editName** của lớp CEmpWnd. Để khắc phục lỗi này, ta khai báo lớp COptionDlg là một lớp bạn (friend) của lớp CEmpWnd.

```
class CEmpWnd : public CWnd
{
    friend class COptionDlg;      // COptionDlg is a friend
public:
    CEmpWnd();
    ...                          // other declarations
};
```

9.5 SỬ DỤNG DIALOG LÀM GIAO DIỆN CHÍNH CỦA ỨNG DỤNG:

Kế thừa từ CWnd, lớp CDialog và các lớp kế thừa từ nó có thể dùng khai báo các đối tượng cửa sổ giao diện chính của ứng dụng.

9.5.1 Thực hiện ứng dụng với giao diện chính là dialog:

- Tạo dự án VD27 tương tự VD03 (dự án chỉ có lớp kế thừa CWinApp).
- Tạo dialog resource có nội dung tùy ý làm giao diện chính.
- Tạo lớp quản lý dialog resource. Giả sử lớp có tên là CMainDlg có mã nguồn trong các tập tin MainDlg.H và MainDlg.CPP.
- Đăng ký sử dụng lớp CMainDlg cho phần cài đặt của lớp quản lý ứng dụng: Bổ sung vào đầu tập tin cài đặt của lớp (VD27.cpp):

```
#include "maindlg.h"    // at the begin of program
```

- Hành vi InitInstance của lớp CEmpApp tạo dialog giao diện.

```

BOOL CEmpApp::InitInstance ()
{
    CMainDlg main;           // Khai báo đối tượng dialog
    m_pMainWnd = &main;      // Dùng dialog làm cửa sổ chính
    main.DoModal();           // Thực hiện dialog
    return TRUE;
}

```

* **Cài biểu tượng ứng dụng trên tiêu đề của dialog:**

- Khai báo *style* là Popup hoặc Overlap cho dialog resource.
- Hành vi OnInitDialog của dialog sẽ thực hiện cài đặt icon

```

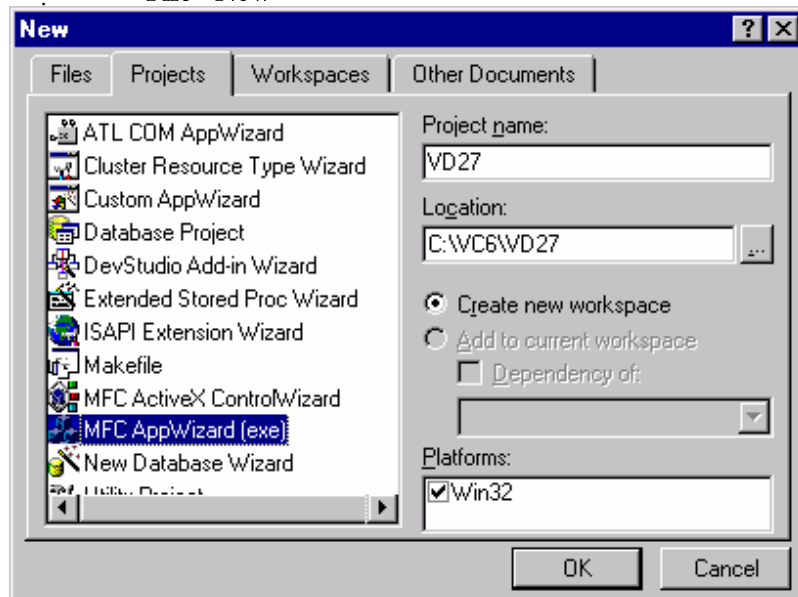
BOOL CMainDlg::OnInitDialog()
{
    CDialog::OnInitDialog();
    SetIcon(AfxGetApp()->LoadIcon(IDR_MAINFRAME), TRUE);
    return TRUE;
}

```

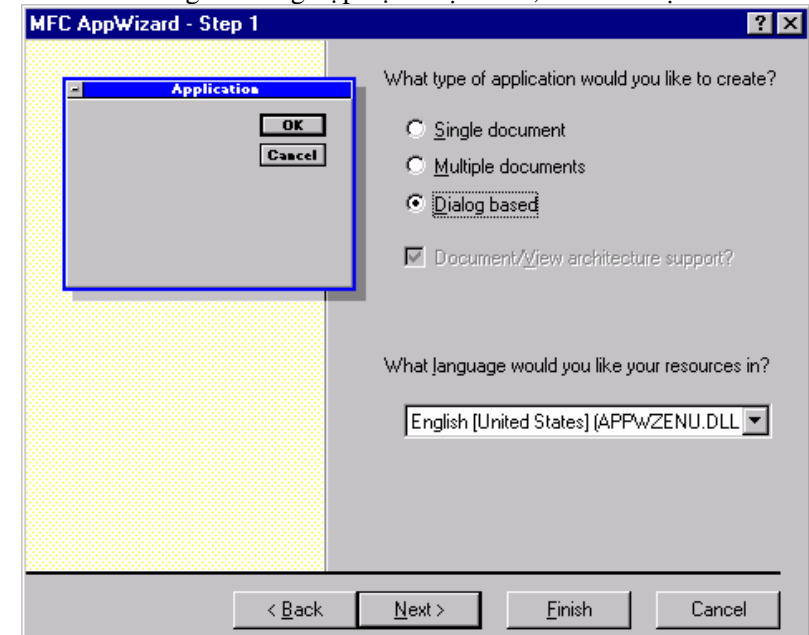
9.5.2 Dùng MFC wizard tạo ứng dụng với giao diện chính là dialog:

Để người dùng có ngay một dự án cơ VD27 mà không phải mất công thực hiện các việc như trên, MFC wizard cung cấp chức năng hỗ trợ tạo nhanh dự án với dialog làm cửa sổ chính. Cách sử dụng chức năng hỗ trợ này như sau:

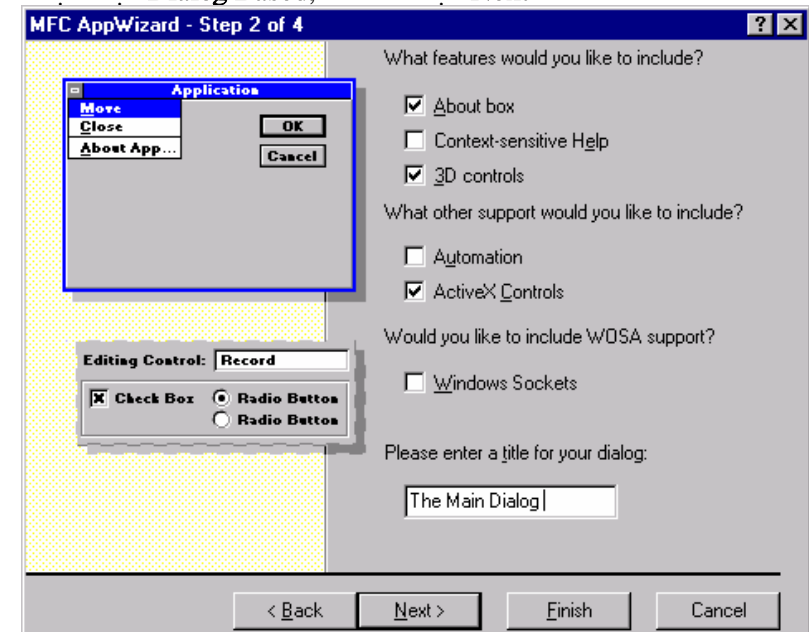
- Chọn menu **File / New**.



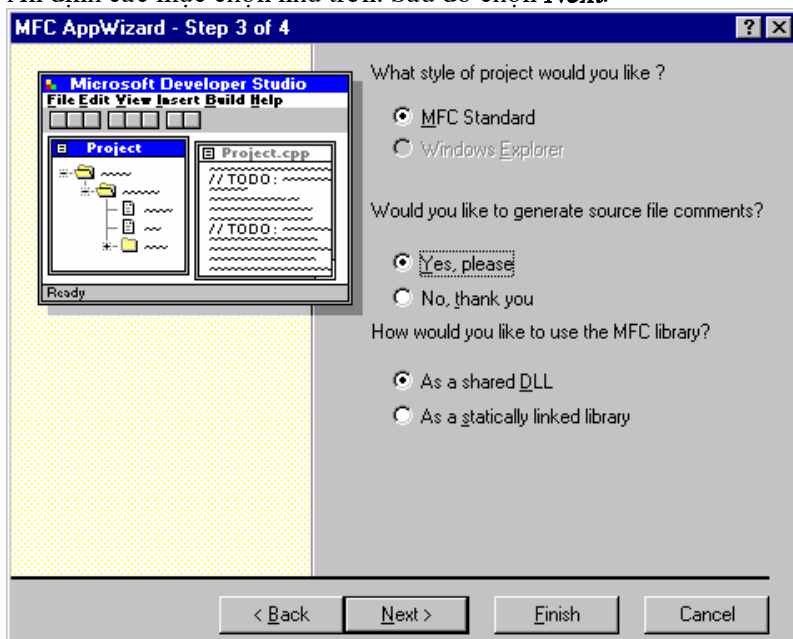
- Điền các thông tin trong hộp hội thoại **New**, Sau đó chọn **OK**.



- Chọn mục **Dialog Based**, sau đó chọn **Next**.



- Ấn định các mục chọn như trên. Sau đó chọn **Next**.



- Chọn cơ chế liên kết với thư viện MFC. Sau đó chọn **next**.
- Cuối cùng, ấn định tên tập tin chứa khai báo các lớp. Chọn **Finish**.
- Biên dịch và chạy thử ứng dụng.

9.6 KHAI BÁO BIẾN CHO CONTROL TRÊN DIALOG:

Khai báo biến cho control trên dialog là thực hiện định nghĩa biến đối tượng có kiểu phù hợp và thiết lập mối quan hệ giữa biến đối tượng và control liên quan. Thông qua biến đối tượng, ta dễ dàng tiến hành các xử lý cần thiết để tác động hoặc lấy giá trị của control. Có hai loại biến:

- **Biến giá trị (value variable):** Biến được sử dụng để lưu trữ giá trị của control.
- **Biến điều khiển (control variable):** Biến đối tượng, có kiểu phù hợp và các hành vi xử lý cần thiết, được sử dụng để tác động lên control.

Đồng bộ nội dung nhập trong control và nội dung biến giá trị:

Khi người dùng thao tác nhập liệu trên control, nội dung nhập chưa thực sự trở thành giá trị của biến. Ngược lại, việc gán trị cho biến cũng không làm thay đổi ngay nội dung hiển thị trong control.

Để thực hiện đồng bộ hai nội dung khi một trong hai hiện tượng nói trên xảy ra, ta sử dụng hành vi sau mà lớp CDialog kế thừa từ CWnd:

```
BOOL UpdateData( BOOL bSaveAndValidate = TRUE );
```

Tham số *bSaveAndValidate* có ý nghĩa như sau:

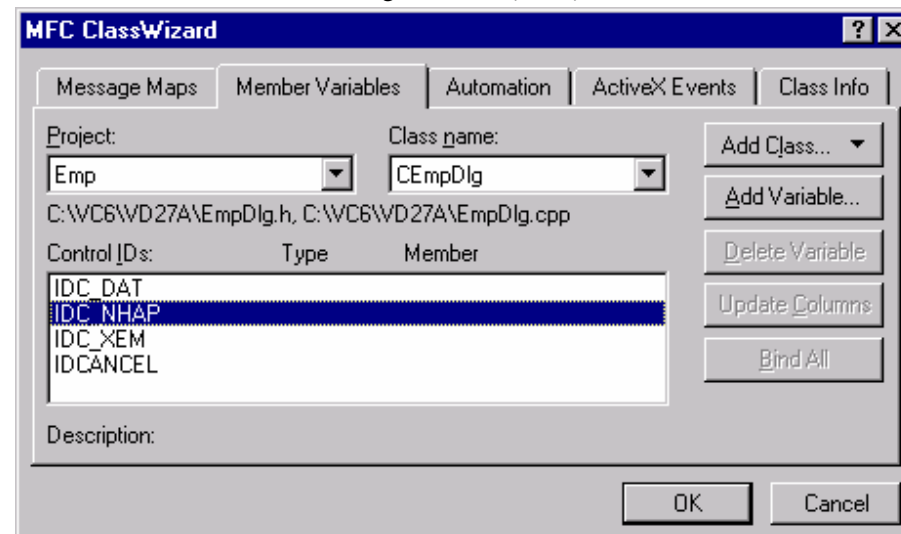
- TRUE : Chuyển giá trị trong control vào biến giá trị.
- FALSE : Chuyển giá trị trong biến giá trị vào control.

Sử dụng biến điều khiển:

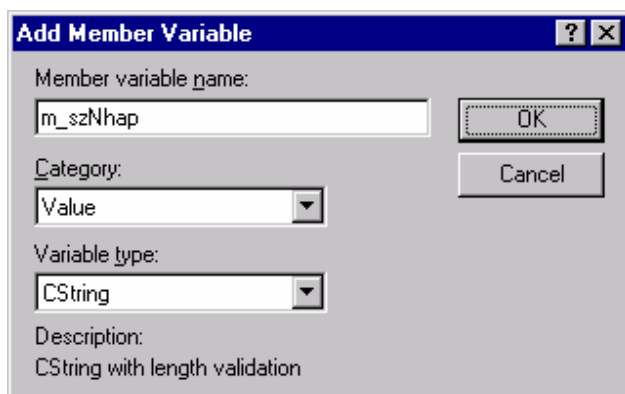
Biến điều khiển là đối tượng thuộc một lớp đối tượng nhập liệu (xem chương 8) có kiểu phù hợp với control. Thông qua các hành vi của biến đối tượng này, ta có thể thực hiện các tác động cho phép lên control.

Thực hiện ứng dụng với giao diện chính là dialog, một mục nhập và hai nút **View** và **Set** để hiển thị và đặt nội dung trong hộp nhập.

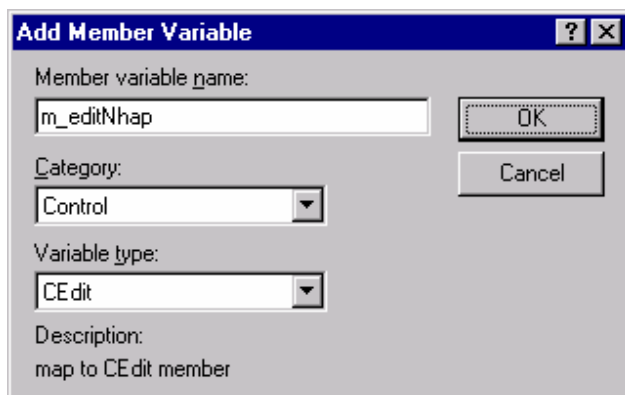
- Áp dụng mục (9.5.2), tạo dự án VD27A với giao diện chính là dialog.
- Mở resource của dialog giao diện chính, cài đặt các control sau:
 - **Hộp nhập liệu** : Edit Số hiệu IDC_NHAP
 - **Nút lệnh hiển thị nội dung hộp nhập** : Button IDC_XEM
 - **Nút lệnh đặt nội dung hộp nhập** : Button IDC_DAT
- Khai báo biến **m_szNhap** kiểu CString, dùng lưu giá trị **hộp nhập liệu**. Cách thực hiện như sau:
 - Mở resource của dialog chính, chọn mục **View/ClassWizard**.



- Chọn trang **Member Variables**, chọn mục có số hiệu IDC_NHAP. Sau đó chọn mục **Add Variable....**



- **Member variable name** = *m_szNhap* : Tên biến
- **Category** = *Value* : Loại biến
- **Variable type** = *CString* : Kiểu của biến. Chọn **OK**.
- Khai báo biến *m_editNhap* kiểu CEdit. Biến này được sử dụng để tác động giá trị *hộp nhập liệu*. Thực hiện như khai báo biến *m_szNhap*.



- Hành vi OnXem ứng với nút chọn IDC_XEM hiển thị nội dung nhập:

```
void CEmpDlg::OnXem()
{
    UpdateData(TRUE);    // Cập nhật giá trị control vào biến
    MessageBox( m_szNhap, "Ban da nhap", MB_OK );
}
```

- Hành vi OnDat ứng với nút chọn IDC_XEM đặt giá trị cho hộp nhập:

```
void CEmpDlg::OnDat()
{
    m_editNhap.SetWindowText( "Mr.Emp, Hello !" );
}
```

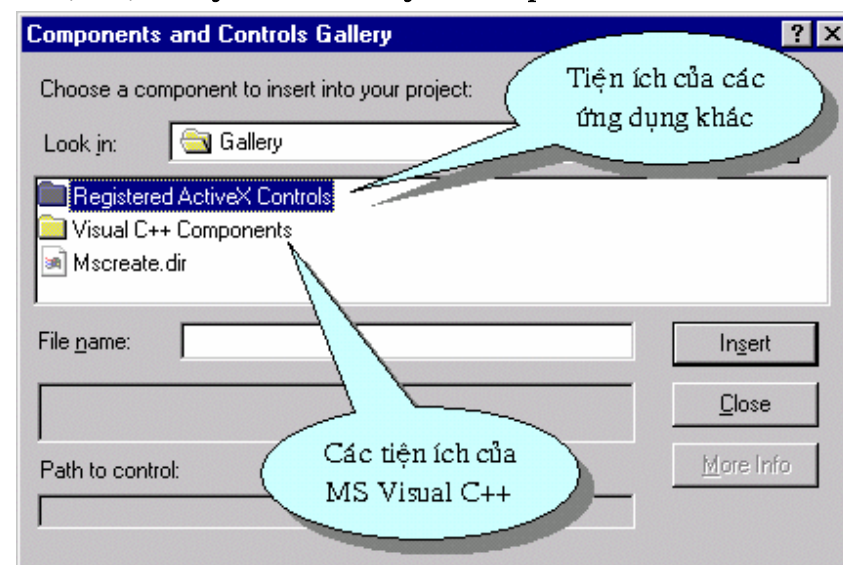
- Biên dịch và chạy thử ứng dụng.
- ☞ Hành vi OnDat có thể cài đặt như sau:

```
void CEmpDlg::OnDat()
{
    m_szNhap = "Mr.Emp, Hello !";
    UpdateData(FALSE);    // Cập nhật giá trị biến cho control
}
```

9.7 KHAI THÁC CÁC TIỆN ÍCH HỖ TRỢ:

Ứng dụng phát triển trong VC có thể dễ dàng khai thác các tiện ích hỗ trợ bởi VC (VC Components) hoặc bởi các ứng dụng khác được cài đặt trong hệ thống (ActiveX Controls). Việc bổ sung và khai thác các tiện ích này trong ứng dụng được thực hiện thông qua các bước sau:

- Mở dự án của ứng dụng trong VC.
- Đăng ký sử dụng tiện ích:
Chọn mục: **Project / Add to Project / Components and Controls...**



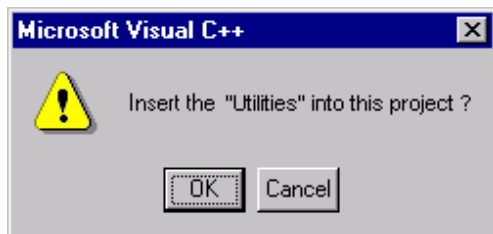
- **Chọn nhóm tiện ích:** Double-click mục nhóm tiện ích cần dùng.



(Danh sách các tiện ích của MS Visual C++)

- **Chọn tiện ích:**

Chọn mục thích hợp trong danh sách tiện ích chi tiết. Chọn **Insert**.

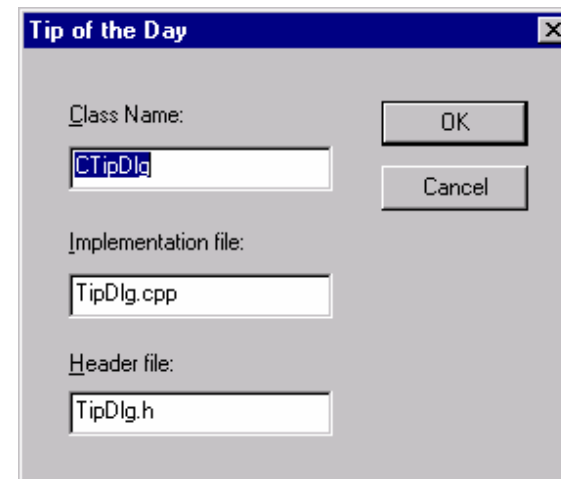


Chọn **OK** để khẳng định thao tác.

- Khai báo thông tin lớp đối tượng quản lý tiện ích bổ sung (nếu có).
- Khai báo thông tin lớp đối tượng trong ứng dụng sử dụng tiện ích.
- Kết thúc quá trình bổ sung một tiện ích vào ứng dụng.

Thực hiện ứng dụng **TipApp** với giao diện chính là dialog. Mỗi khi thực hiện, ứng dụng hiển thị hộp thông báo "Tip of the Day".

- Dùng MFC Wizard tạo dự án **TipApp** với giao diện chính là dialog.
- Đăng ký sử dụng tiện ích:
 - **Nhóm tiện ích** = *Visual C++ Components*.
 - **Tiện ích** = *Tip of the day*
 - Khai báo thông tin lớp đối tượng quản lý tiện ích:



Chọn **OK**, Chọn **Close** để đóng hộp hội thoại tiện ích.

- Với tiện ích "Tip of the Day", lớp đối tượng quản lý tiểu trình chính của ứng dụng là lớp sử dụng. Hai hành vi bổ sung tự động cho lớp:
 - void CTipAppApp::ShowTipAtStartup(void);
 - void CTipAppApp::ShowTipOfTheDay(void);

Hành vi ShowTipAtStartup được thực hiện bởi hành vi InitInstance.

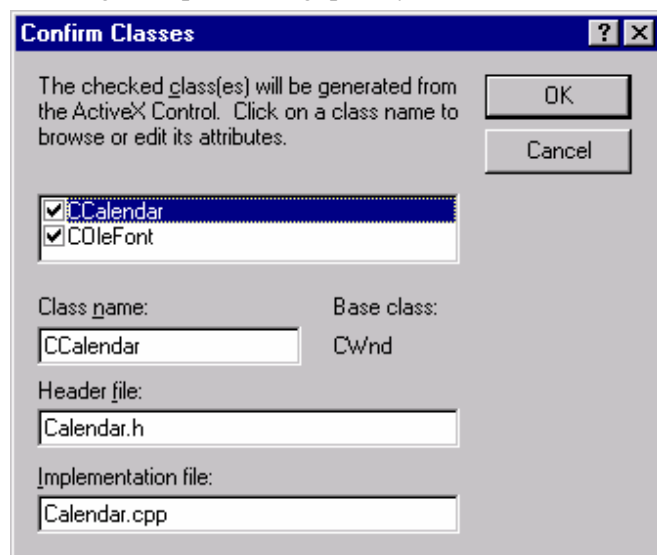
Hãy thử quan sát các lớp của dự án **TipApp** để tự nhận xét.

- Biên dịch và chạy thử ứng dụng.

Thực hiện ứng dụng **XemLich** với giao diện chính là dialog. Bổ sung vào ứng dụng công cụ xem và chọn lịch Calendar 8 (ActiveX Control). Cài đặt control này lên dialog giao diện.

- Dùng MFC Wizard tạo dự án **XemLich** với dialog giao diện chính.
- Đăng ký sử dụng tiện ích:
 - **Nhóm tiện ích** = *Registered ActiveX Controls*.
 - **Tiện ích** = *Calendar Control 8.0*

- Khai báo thông tin lớp đối tượng quản lý tiện ích:



Chọn **OK**, Chọn **Close** để đóng hộp hội thoại tiện ích.

- Lớp đối tượng CXemLichDlg quản lý dialog giao diện chính của dự án làm nhiệm vụ sử dụng control bổ sung này:
 - Mở resource dialog giao diện, cài đặt control sau:
 - Hộp xem và chọn ngày CCalendar IDC_DATE
 - Tạo biến điều khiển **m_Lich** kiểu CCalendar cho control.
 - Hành vi OnInitDialog của lớp đặt giá trị cho control:

```

BOOL CXemLichDlg::OnInitDialog()
{
    CDialog::OnInitDialog();
    SetIcon(m_hIcon, TRUE);
    SetIcon(m_hIcon, FALSE);

    m_Lich.SetDay( 19 );
    m_Lich.SetMonth( 11 );
    m_Lich.SetYear( 2002 );


    return TRUE;
}

```

Có thể sử dụng các hành vi: *GetDay*, *GetMonth*, *GetYear* của đối tượng **m_List** để lấy thông tin về ngày được chọn.

- Biên dịch và chạy thử ứng dụng.

THỰC HÀNH:

1. Như VD26, khi option dialog được thực hiện, hộp nhập của nó chứa sẵn thông tin là nội dung đã được nhập trên màn hình chính CEmpWnd.
- HD: Hành vi OnInitDialog của COptionDlg thực hiện lấy thông tin từ thuộc tính **m_editName** của đối tượng CEmpWnd và gán cho hộp nhập của nó.
2. Viết ứng dụng với option dialog cho phép nhập 4 thông số tọa độ (left-top, right-bottom). Khi chọn **OK**, các thông số này được áp đặt cho cửa sổ chính.
3. Viết ứng dụng với **n** chú bướm bay trong vùng client của cửa sổ chính và option dialog cho phép ấn định số lượng (n), vận tốc của bướm.
4. Viết ứng dụng thực hiện chức năng của một cái máy tính bỏ túi. Giao diện là một dialog, các nút bấm số và phép tính được cài đặt bằng các control.
5. Viết ứng dụng kính lúp với giao diện chính là dialog. Khi di chuyển dialog trên màn hình desktop, thông tin của desktop bên dưới dialog được phóng to (với tỉ lệ ấn định được) và hiển thị trong vùng client của dialog.
6. Thực hiện ứng dụng với giao diện chính là dialog có hình ngôi sao.
7. Thực hiện ứng dụng với giao diện chính là dialog mà khuôn dạng của nó được hình thành từ một ảnh bitmap bất kỳ.
8. Thực hiện ứng dụng với giao diện chính là dialog. Dialog này cho phép thay đổi vị trí bằng cách drag chuột ở bất kỳ vị trí nào trên bề mặt của nó.
9. Thực hiện dialog với màu nền tùy chọn, một mục thông báo có nội dung "Mailto: emp@hcmueco.edu.vn". Hình ảnh chuột đổi thành  khi di chuyển chuột lên vị trí mục. Click chọn mục cho phép gửi mail.

HD:

- Xử lý WM_CTLCOLOR để đặt màu nền của dialog và màu chữ mục.
- Mục cài đặt với số hiệu xác định, thông số dạng chứa WM_NOTIFY.
- Xử lý BN_CLICKED trên mục, và gửi mail thông qua hàm sau:

```

HINSTANCE ShellExecute (
    HWND hWnd,           // Handle cửa sổ cha.
    LPCTSTR lpOperation,  // Chuỗi nội dung tác vụ
    LPCTSTR lpFile,       // Đường dẫn tập tin chương trình
    LPCTSTR lpParameters, // Chuỗi tham số dòng lệnh
    LPCTSTR lpDirectory,  // Đường dẫn thư mục làm việc
    int nShowCmd          // Thông số hiển thị cửa sổ
);

```

Lệnh thực hiện một chương trình ứng dụng trong windows.

VÍ DỤ: Để gửi mail, ta sử dụng hàm trên như sau.

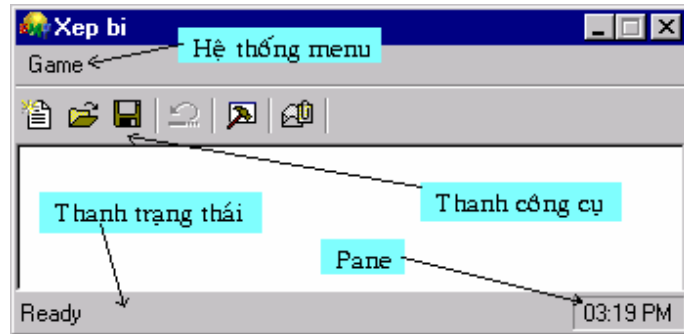
```
ShellExecute( NULL, "open", "Mailto:emp@hcmueco.edu.vn","", "", SW_SHOW );
```

CHƯƠNG 10:

KHUNG CỬA SỔ GIAO DIỆN CHÍNH

10.1 KHUNG CỬA SỔ GIAO DIỆN (FRAME WINDOW):

Frame window là loại cửa sổ chuyên dùng làm màn hình chính của ứng dụng. Frame window có các đặc điểm như sau:



- Cho phép lồng vào frame window thanh trạng thái (statusbar), thanh công cụ (toolbar), menu và các phím tắt trên menu.
- Cho phép lồng một đối tượng view (thuộc lớp kế thừa từ CWnd) vào vùng client của frame window để hoạt động phối hợp trong việc hiển thị nội dung trao đổi của ứng dụng. Hành vi OnSetFocus của frame window thực hiện hành vi OnSetFocus của view để tạo sự đồng bộ giữa frame và view khi kích hoạt. Thông qua hành vi OnCmdMsg, frame window có thể điều phối command message cho view.
- Phối hợp, đồng bộ việc thực hiện xử lý và thay đổi trạng thái của các đối tượng trực thuộc: menu, toolbar, statusbar, view.
- Cho phép cơ chế thực hiện WinHelp và tự động kết thúc WinHelp khi frame window chấm dứt hoạt động.

10.2 THANH TRẠNG THÁI (STATUSBAR) & LỚP CStatusBar:

StatusBar là khung bao gồm các hộp (pane) chứa thông báo. Thông qua statusbar, ứng dụng có thể thông tin cho người dùng về các hoạt động của ứng dụng và các thông số liên quan. Statusbar thường được đặt ở cạnh dưới của frame window. Mỗi pane trên statusbar có thể được ấn định một kiểu dạng thể hiện và nội dung thông tin riêng.

Việc quản lý statusbar trong ứng dụng được MFC hỗ trợ thông qua lớp đối tượng CStatusBar. Các hành vi đặc trưng của lớp này như sau:

- **CStatusBar()**; Tạo lập đối tượng rỗng.
- **BOOL Create** (
 CWnd* pParentWnd, /* Con trỏ đối tượng cửa sổ cha */
 /* Thông số dạng của statusbar và giá trị mặc nhiên */
 DWORD dwStyle WS_CHILD | WS_VISIBLE | CBS_BOTTOM,
 /* Số hiệu statusbar, giá trị mặc nhiên cho statusbar chính */
 UINT nID = AFX_IDW_STATUS_BAR
); Khởi tạo thông số cho đối tượng statusbar.
- **BOOL SetIndicators** (
 const UINT* lpIDArray, /* Mảng chứa số hiệu panes
 int nIDCount /* Số pane trên statusbar
); Qui định số pane và số hiệu của chúng.
Các số hiệu pane đặc biệt như sau:
 ID_SEPARATOR : Dùng cho pane thông tin trạng thái,
 ID_INDICATOR_CAPS : Pane trạng thái phím capslock.
 ID_INDICATOR_NUM : Pane trạng thái phím numlock
 ID_INDICATOR_SCRL : Pane trạng thái phím scrolllock
- **void SetWindowText** (
 LPCTSTR lpszString /* Nội dung thông báo cập nhật
); Cập nhật thông báo cho pane có số hiệu là 0.
- **BOOL SetPaneText** (
 int nIndex, /* Số hiệu pane
 LPCTSTR lpszNewText, /* Nội dung thông báo cập nhật
 BOOL bUpdate=TRUE /* Vẽ lại thông tin cập nhật
); Cập nhật nội dung thông báo trong pane một pane.
- **void SetPaneInfo** (
 int nIndex, /* Số thứ tự pane trong statusbar
 UINT nID, /* Số hiệu gán cho pane
 UINT nStyle, /* Thông số dạng của pane
 int cxWidth /* Độ rộng pane (tính bằng pixel)
); Ấn định các thông số liên quan hoạt động hiển thị của pane.
Giá trị nStyle có thể là:
 SBPS_NOBORDERS : Không viền.
 SBPS_POPOUT : Viền nổi.
 SBPS_DISABLED : Cấm.
 SBPS_STRETCH : Tự chỉnh kích thước (cho 1 pane duy nhất).

- SBPS_NORMAL : Không có kiểu dáng đặc biệt,
- void **SetPaneStyle** (
 - int *nIndex*, // Số thứ tự pane trong statusbar
 - UINT *nStyle* // Thông số dạng của pane
); Ấn định thông số dạng của pane.

10.3 THANH CÔNG CỤ (TOOLBAR) & LỚP CToolBar:

Toolbar là khung chứa các mục chọn có hình ảnh gợi nhớ và được kết vào cạnh của frame window. Mỗi mục chọn của toolbar có một số hiệu riêng hoặc gắn với một mục chọn xác định trong hệ thống menu. CToolBar là lớp do MFC cung cấp nhằm hỗ trợ việc quản lý toolbar trong ứng dụng.

- CToolBar**(); Tạo lập đối tượng rỗng.
- BOOL **Create** (
 - CWnd* *pParentWnd*, /* Con trỏ đối tượng cửa sổ cha */
 - /* Thông số dạng của toolbar với giá trị mặc nhiên */
 - DWORD *dwStyle* = WS_CHILD | WS_VISIBLE | CBRS_TOP,
 - /* Số hiệu của toolbar, giá trị mặc nhiên cho toolbar chính */
 - UINT *nID* = AFX_IDW_TOOLBAR
); Khởi tạo thông số cho đối tượng toolbar.
dwStyle có thể nhận các thông số sau:
 - CBRS_GRIPPER : Toolbar di chuyển được
 - CBRS_FLOAT_MULTI : Cho phép nhiều toolbar trong 1 frame
 - CBRS_TOOLTIPS : Có tiptext cho mỗi mục chọn
 - CBRS_FLYBY : Đồng bộ tooltip và message info
 - CBRS_SIZE_DYNAMIC : Kích thước thay đổi được
 - CBRS_SIZE_FIXED : Kích thước cố định
- BOOL **CreateEx** (
 - CWnd* *pParentWnd*,
 - DWORD *dwCtrlStyle* = TBSTYLE_FLAT, // Dạng phẳng
 - DWORD *dwStyle* = WS_CHILD | WS_VISIBLE | CBRS_TOP
 - CRect *rcBorders* = CRect (0, 0, 0, 0),
 - UINT *nID* = AFX_IDW_TOOLBAR
); Khởi tạo đối tượng toolbar với thông số dạng mở rộng.
dwCtrlStyle có thể là TBSTYLE_TRANSPARENT (trong suốt).
- BOOL **LoadToolBar** (
 - UINT *nIDResource* // Số hiệu của toolbar resource
); Tạo dạng cho toolbar từ toolbar resource.

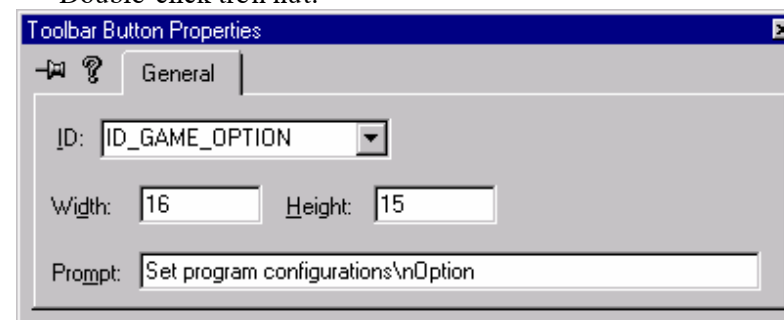
- void **SetHeight**(int *cyHeight*); Ấn định chiều cao của toolbar.
- void **EnableDocking** (
 - DWORD *dwStyle* // Cách kết toolbar vào frame window
 - // xem EnableDocking() của CFrameWnd
); Ấn định kiểu kết cho phép của toolbar với cửa sổ cha của nó.
- CToolBarCtrl& **GetToolBarCtrl**(); Trả về đối tượng quản lý các nút chọn trên toolbar. Với đối tượng này ta có thể thực hiện các chỉnh sửa phù hợp trên từng nút chọn.


☞ **Thực hiện toolbar trong ứng dụng:** Ta tiến hành các bước sau đây.

10.3.1 Thiết kế toolbar resource:

Toolbar resource chứa thông tin về hình ảnh, số hiệu, nội dung thông báo và các hướng dẫn cho từng mục chọn trên thanh toolbar.

- Mở dự án cần bổ sung toolbar resource trong VC.
- Tạo mới toolbar resource: Thực hiện tương tự tạo mới icon (2.8).
Lưu ý: Chọn **Resource Type = Toolbar**.
- Đặt số hiệu cho toolbar (ví dụ IDR_MAINFRAME cho toolbar chính).
- Thiết kế toolbar thông qua màn hình thiết kế mà ta vừa nhận được từ bước trên. Các thao tác cơ bản như sau:
 - Khai báo thông số cho nút chọn:**
 - Double-click trên nút:



- Ấn định các thông số cho nút:
 - ID*: Số hiệu của nút chọn, có thể là số hiệu của mục menu.
 - Width, Height*: Chiều rộng và chiều cao của nút.
 - Prompt*: Tương tự như *Prompt* của mục chọn trên menu.
- Thêm một nút mới:** Double-click trên . Sau đó thực hiện khai báo thông số cho nút như trên.
- Chuyển vị trí của nút:** Drag nút đến vị trí thích hợp.

- **Tách nhóm nút:** Drag nút ra xa vị trí nút kế cận.
 - **Xóa nút:** Drag nút ra khỏi thanh toolbar.
 - **Trang trí nút:** Thực hiện như trang trí icon.
- Khi thiết kế xong, chọn mục **Save** và đóng màn hình thiết kế toolbar.

10.3.2 Dùng toolbar resource cho đối tượng CToolBar của FrameWnd:

- Khai báo đối tượng thuộc tính kiểu CToolBar trong lớp CFrameWnd của ứng dụng:

```
CToolBar m_toolbar; // Đặt trong khai báo lớp CFrameWnd
```

- Hành vi **OnCreate** của FrameWnd khởi tạo thông số cho đối tượng toolbar. Sau đó kết đối tượng toolbar vào frame window.

```
int CEmpFrame::OnCreate( LPCREATESTRUCT lpCreateStruct )
{
    if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
        return -1;

    // Create the main toolbar for frame window
    m_toolbar.CreateEx(this, TBSTYLE_FLAT,
        WS_CHILD | WS_VISIBLE | CBRS_TOP
        | CBRS_GRIPPER | CBRS_TOOLTIPS | CBRS_FLYBY);
    // Initiate toolbar's appearance by toolbar resource
    m_toolbar.LoadToolBar(IDR_MAINFRAME);
    m_toolbar.EnableDocking(CBRS_ALIGN_ANY);
    // attach the toolbar to frame window
    this->EnableDocking(CBRS_ALIGN_ANY);
    this->DockControlBar(&m_toolbar, AFX_IDW_DOCKBAR_TOP);
    return 0;
}
```

10.4 LỚP CFrameWnd:

CFrameWnd là lớp đối tượng kế thừa từ CWnd cho phép quản lý frame window trong ứng dụng. Ngoài các thuộc tính và hành vi kế thừa public từ CWnd, CFrameWnd có các thuộc tính và hành vi đặc trưng giúp việc ấn định và điều khiển frame window được dễ dàng và hiệu quả.

- **CFrameWnd()**; Hành vi tạo lập.
- **BOOL Create** (
 - LPCTSTR *lpzClassName*, // Tên lớp đã đăng ký của frame
 - LPCTSTR *lpzWindowName*, // Tên, tiêu đề của frame window
 - /* thông số dạng của frame */

```
DWORD dwStyle = WS_OVERLAPPEDWINDOW,
/* Tọa độ, kích thước của frame
rectDefault: Giá trị mặc nhiên */
```

```
const RECT& rect = rectDefault,
```

```
// Con trỏ đối tượng cửa sổ cha
```

```
CWnd* pParentWnd = NULL,
```

```
// Chuỗi tên menu resource
```

```
LPCTSTR lpzMenuName = NULL,
```

```
DWORD dwExStyle // Thông số dạng frame mở rộng
```

```
); Khởi tạo thông số cho frame window.
```

- **afx_msg int OnCreate**(LPCREATESTRUCT *lpCreateStruct*); Hành vi kế thừa cho phép khởi tạo thông số các đối tượng trực thuộc.

- **virtual BOOL LoadFrame** (

```
UINT nIDResource, // Số hiệu các resource liên quan: icon,
```

```
// cursor, bảng phím tắt, menu (bắt buộc).
```

```
DWORD dwDefaultStyle // Thông số dạng frame window
```

```
= WS_OVERLAPPEDWINDOW | FWS_ADDTOTITLE,
```

```
CWnd* pParentWnd // Con trỏ đối tượng cửa sổ cha
```

```
); Khởi tạo thông số cho frame window.
```

- **BOOL LoadAccelTable** (

```
LPCTSTR Bảng_phím_tắt // Tên bảng phím tắt trong resource
```

```
); Nạp bảng phím tắt, chỉ dùng khi frame được khởi tạo bởi Create.
```

- **virtual void ActivateFrame** (

```
int nCmdShow = -1 // Thông số về trạng thái kích hoạt
```

```
); Hành vi kế thừa để ấn định thông số trạng thái frame khi kích hoạt.
```

- **void EnableDocking** (

```
DWORD dwDockStyle // Thông số ấn định
```

```
); Qui định cách kết thanh công cụ vào frame window.
```

dwDockStyle có thể nhận một trong các giá trị sau:

```
CBRS_ALIGN_TOP : Kết ở cạnh trên vùng client.
```

```
CBRS_ALIGN_BOTTOM : Kết ở cạnh dưới
```

```
CBRS_ALIGN_LEFT : Kết ở cạnh trái
```

```
CBRS_ALIGN_RIGHT : Kết ở cạnh phải
```

```
CBRS_ALIGN_ANY : Kết ở mọi vị trí nói trên.
```


- void **DockControlBar** (
CControlBar * *pBar*, // Con trỏ đối tượng toolbar
UINT *nDockBarID* // Thông số ấn định
); Kết đối tượng toolbar vào frame window, sử dụng khi trước đó định cách kết toolbar cho frame window là CBRs_ALIGN_ANY. Thông số ấn định phải phù hợp với đặc tính của *pBar*, có thể là các giá trị sau:
AFX_IDW_DOCKBAR_TOP : Kết ở cạnh trên
AFX_IDW_DOCKBAR_BOTTOM : Kết ở cạnh dưới
AFX_IDW_DOCKBAR_LEFT : Kết ở cạnh trái
AFX_IDW_DOCKBAR_RIGHT : Kết ở cạnh phải
- virtual BOOL **OnCreateClient** (
LPCREATESTRUCT *lpcs*, // Con trỏ đến cấu trúc chứa
CCreateContext* *pContext* // Các thông số liên kết.
); Hành vi được thực hiện khi các view đã được tạo xong. Kế thừa hành vi này để ấn định các thông số riêng của ứng dụng.
- virtual CWnd* **GetMessageBar**(); Trả về con trỏ trỏ đến đối tượng statusbar trực thuộc.

10.5 SỬ DỤNG FRAME WINDOW LÀM GIAO DIỆN CHÍNH:

Kế thừa từ CWnd, CFrameWnd và các lớp kế thừa từ nó có thể dùng khai báo các đối tượng cửa sổ giao diện chính của ứng dụng.

10.5.1 Thực hiện ứng dụng với giao diện chính là CFrameWnd:

📖 **Thực hành 1:** Thực hiện ứng dụng với cửa sổ chính là frame window.

- Tạo dự án VD28 chỉ chứa lớp CEmpApp kế thừa từ CWinApp.
- Trong hành vi InitInstance của CEmpApp, khai báo biến con trỏ đối tượng CFrameWnd* và dùng nó làm cửa sổ giao diện chính:

```

BOOL CEmpApp::InitInstance( )
{
    CFrameWnd *main = new CFrameWnd;
    m_pMainWnd = main; // frame window as the main window
    main->Create (
        AfxRegisterWndClass(CS_HREDRAW|CS_VREDRAW,
            LoadCursor(IDC_MAINFRAME), NULL,
            LoadIcon(IDR_MAINFRAME) ),
        "Emp.Example 28" );
    main->ShowWindow(SW_SHOW); // Show the frame window
    return TRUE;
}

```

📖 **Thực hành 2:** Thực hiện ứng dụng tương tự VD28, cài hệ thống menu tương tự ví dụ chương 7 cho đối tượng frame window.

- Tạo mới dự án VD29 tương tự dự án VD28.
- Tạo mới menu resource với số hiệu IDR_MAINFRAME, thiết kế menu này như mô tả ở chương 7.
- Tạo mới bảng phím tắt với số hiệu IDR_MAINFRAME. Khai báo nội dung bảng phím tắt như mục (7.5).
- Hành vi InitInstance của CEmpApp thực hiện khai báo con trỏ đối tượng CFrameWnd*. Dùng hành vi LoadFrame của đối tượng này để khởi động thông số cho nó từ các resource liên quan:

```

BOOL CEmpApp::InitInstance()
{
    CFrameWnd *main = new CFrameWnd;
    m_pMainWnd = main;
    main->LoadFrame(IDR_MAINFRAME);
    main->ShowWindow(SW_SHOW); // Show the frame window
    return TRUE;
}

```

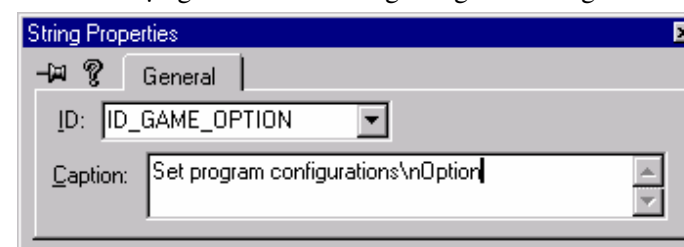
10.5.2 String Table và CFrameWnd:

StringTable, một thành phần trong resource của ứng dụng, dùng chứa các giá trị hằng chuỗi. Mỗi hằng chuỗi có một số hiệu phân biệt. Việc tạo mới hằng chuỗi trong resource của ứng dụng được thực hiện như sau:

- Tạo mới resource (như 2.8). Lưu ý chọn resource type là **String Table**.

ID	Value	Caption
IDR_MAINFRAME	128	Emp.Example 29
ID_GAME_START	32771	Start the new game\nStart
ID_GAME_OPTION	32772	Set program configurations\nOption
ID_GAME_ABOUT	32773	Show the author's name and progr

- Double-click hoặc gõ Enter trên dòng trống của StringTable:



- **ID** = Số hiệu hằng chuỗi.
- **Caption** = Nội dung hằng chuỗi. Gõ phím **Enter** để kết thúc.
- Cuối cùng, lưu và đóng màn hình khai báo StringTable.

Φ **Sử dụng hằng chuỗi trong StringTable**: Có thể sử dụng hằng chuỗi trong StringTable làm giá trị cho biến chuỗi trong chương trình. Đối với biến chuỗi được quản lý bởi đối tượng chuỗi CString, xử lý này được thực hiện thông qua hành vi sau:

```
BOOL CString::LoadString (
    UINT nID           // Số hiệu hằng chuỗi trong resource
);
```

Sau đây là một ví dụ khởi tạo giá trị chuỗi myString từ string resource:

```
CString myString;
myString.LoadString (ID_GAME_OPTION);
```

Φ **Dùng hằng chuỗi trong Stringtable làm tiêu đề frame window**.

Hằng chuỗi có số hiệu trùng với giá trị số hiệu làm tham số *nIDResource* cho hành vi khởi tạo thông số LoadFrame của đối tượng frame window sẽ được sử dụng làm tiêu đề của frame window này.

- Mở dự án VD29.
- Tạo mới hằng chuỗi "Emp.Example 29" trong StringTable với số hiệu là IDR_MAINFRAME.
- Thực hiện chạy thử ứng dụng và quan sát tiêu đề cửa frame window.

⇒ **Thực hành 3**: Thực hiện ứng dụng như VD29; thanh statusbar với ba mục (pane): mục thứ nhất chứa nội dung "Mr.Emp", mục thứ hai chứa nội dung "Hello world!", mục thứ ba hiển thị thông tin hoạt động của ứng dụng.

HD: CFrameWnd không có sẵn statusbar. Dùng CEmpFrame kế thừa CFrameWnd và bổ sung đối tượng thuộc tính này. Hành vi OnCreate của CEmpFrame thực hiện khởi tạo thông số cho đối tượng statusbar. Dùng CEmpFrame khai báo đối tượng cửa sổ chính của ứng dụng.

- Tạo dự án VD30 như VD29.
- Bổ sung lớp CEmpFrame kế thừa CFrameWnd. Xem (5.4.1).
- Bổ sung đối tượng thuộc tính protected **m_status** thuộc lớp CStatusBar cho lớp CEmpFrame:

```
CStatusBar m_status;
```

- Bổ sung hành vi OnCreate (xử lý WM_CREATE) cho CEmpFrame và thực hiện cài đặt như sau:

```
int CEmpFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CFrameWnd::OnCreate( lpCreateStruct ) == -1)
        return -1;
    // The IDs of all statusbar panes
    UINT ID_array[3] = { 1, 1, ID_SEPARATOR };

    m_status.Create( this );           // Thanh statusbar chính
    m_status.SetIndicators( ID_array, 3 );
    m_status.SetPaneText( 0, "Mr.Emp" );
    m_status.SetPaneText( 1, "Hello world!" );

    // Set the size of each pane on statusbar
    m_status.SetPanelInfo( 0, 1000, 0, 40 );
    m_status.SetPanelInfo( 1, 1001, 0, 70 );
    m_status.SetPanelInfo( 2, ID_SEPARATOR, 0, 300 );
    return 0;
}
```

⇒ **Thực hành 4**: Viết ứng dụng như VD30 với thanh công cụ có 4 nút chọn tương ứng 4 mục chọn trong hệ thống menu.

- Tạo dự án VD31 như VD30.
- Tạo mới toolbar resource có số hiệu IDR_MAINFRAME với các nút chọn tương ứng các mục menu.
- Bổ sung đối tượng thuộc tính **m_toolbar** thuộc lớp CToolBar cho lớp CEmpFrame. Hành vi OnCreate của CEmpFrame thực hiện khởi tạo thông số cho các đối tượng **m_status** và **m_toolbar**, sau đó kết đối tượng **m_toolbar** vào frame window

```
int CEmpFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    ...                               // Các cài đặt như VD30
    m_toolbar.CreateEx ( this,
        TBSTYLE_FLAT, WS_CHILD | WS_VISIBLE | CBRS_TOP
        | CBRS_GRIPPER | CBRS_TOOLTIPS | CBRS_FLYBY );
    m_toolbar.LoadToolBar ( IDR_MAINFRAME );
    m_toolbar.EnableDocking ( CBRS_ALIGN_ANY );
    this->EnableDocking ( CBRS_ALIGN_ANY );
    this->DockControlBar( &m_toolbar, AFX_IDW_DOCKBAR_TOP );
    return 0;
}
```


- Hành vi `InitInstance` của `CEmpApp` sử dụng lớp `CEmpFrame` khai báo đối tượng cửa sổ chính của ứng dụng:

```

BOOL CEmpApp::InitInstance()
{
    CEmpFrame *main = new CEmpFrame;
    m_pMainWnd = main;
    main->LoadFrame(IDR_MAINFRAME);
    main->ShowWindow(SW_SHOW);
    return TRUE;
}

```

🔗 **Thực hành 5:** Thực hiện ứng dụng tương tự VD31. Trong frame window, cài đặt cửa sổ view chứa dòng chữ chạy kiểu băng chữ điện tử.

- Tạo dự án VD32 tương tự VD31.
- Bổ sung lớp `CEmpView` kế thừa từ `CWnd`. `CEmpView` đảm nhận việc hiển thị nội dung chữ chạy. Các cài đặt cần thiết như sau:
 - Hành vi `OnCreate` thực hiện cài đặt timer:

```

int CEmpView::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CWnd::OnCreate( lpCreateStruct) == -1 )
        return -1;
    SetTimer( ID_TIMER, 300, NULL );    // Interval = 300ms
    return 0;
}

```

- Hành vi `OnDestroy` hủy bỏ timer:

```

void CEmpView::OnDestroy()
{
    KillTimer( ID_TIMER );
    CWnd::OnDestroy();
}

```

- Hành vi `OnTimer` kích hoạt `OnPaint`:

```

void CEmpView::OnTimer(UINT nIDEvent)
{
    if (nIDEvent == ID_TIMER) {
        Invalidate();    // Generates a WM_PAINT
    }
    CWnd::OnTimer(nIDEvent);
}

```

- Hành vi `OnPaint` thực hiện vẽ chữ chạy:

```

void CEmpView::OnPaint()
{
    static char mess[] = "Chao cac ban !";
    char ch;
    UINT i;
    CPaintDC dc(this); // device context for painting
    ch = mess[0];
    for (i=0; i < strlen(mess)-1; i++)
        mess[i] = mess[i+1];
    mess[i] = ch;
    dc.TextOut(10, 10, mess, 15);
}

```

- Dùng lớp `CEmpView` khai báo đối tượng view trong `CEmpFrame`:
 - Bổ sung thuộc tính protected `m_view` thuộc lớp `CEmpView` vào lớp `CEmpFrame`.
 - Hành vi `OnCreate` của `CEmpFrame` thực hiện khởi tạo thông số phù hợp cho đối tượng `m_view`, dùng số hiệu cho `m_view` là `AFX_IDW_PANE_FIRST` (view mặc nhiên của frame)

```

int CEmpFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    ... // Như cài đặt của VD31
    m_view.Create ( NULL, NULL, AFX_WS_DEFAULT_VIEW,
                   CRect(0,0,0,0), this, AFX_IDW_PANE_FIRST);
    return 0;
}

```

- Hành vi `PreCreateWindow` thực hiện hủy thông số ấn định dạng "chìm xuống" cho vùng client (chứa view – to get a nice look):

```

BOOL CEmpFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    if (!CFrameWnd::PreCreateWindow(cs))
        return FALSE;
    cs.dwExStyle &= ~WS_EX_CLIENTEDGE;
    return TRUE;
}

```

- Dùng `CEmpFrame` làm cửa sổ chính của ứng dụng (như VD31).

📖 **Thực hành 6:** Thực hiện ứng dụng như VD32. Khi chọn mục **Start** của menu thì màn hình view thực hiện chữ chạy, đồng thời nội dung mục chuyển thành **Stop**. Nếu người dùng chọn lại mục này (**Stop**), hoạt động chạy chữ dừng lại và nội dung mục chọn chuyển thành **Start**.

- Tạo dự án VD33 tương tự VD32.
- Bổ sung thuộc tính **m_isRun** kiểu BOOL cho lớp CEmpView. **m_isRun** sẽ làm cờ hiệu cho chữ chạy. Một số chỉnh sửa sau của CEmpView:
 - Khởi đầu ấn định không chạy chữ:

```
int CEmpView::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CWnd::OnCreate(lpCreateStruct) == -1)
        return -1;
    isRun = FALSE;           // no animation
    SetTimer( ID_TIMER, 300, NULL );
    return 0;
}
```

- Hành vi OnTimer() dựa trên **m_isRun** để hành động:

```
void CEmpView::OnTimer(UINT nIDEvent)
{
    if (nIDEvent == ID_TIMER) {
        if ( isRun ) Invalidate();    // animation allowed ?
    }
    CWnd::OnTimer(nIDEvent);
}
```

- Bổ sung hành vi trả lời WM_COMMAND phát sinh bởi mục chọn ID_GAME_START (Start/Stop) trên menu. Đồng thời điều chỉnh nội dung mục thông báo của menu cho phù hợp:

```
void CEmpView::OnGameStart()
{
    isRun = !isRun;
    CMenu *theMenu = GetParent()->GetMenu();
    theMenu->ModifyMenu(ID_GAME_START,
        MF_BYCOMMAND, ID_GAME_START,
        (isRun)? "&Stop" : "&Start" );
}
```

- Bổ sung hành vi OnCmdMsg cho CEmpFrame để chuyển các message WM_COMMAND cho **m_view**: **m_view** trong lớp CEmpFrame chỉ là cửa sổ con, không trực tiếp nhận WM_COMMAND từ menu cửa sổ cha.

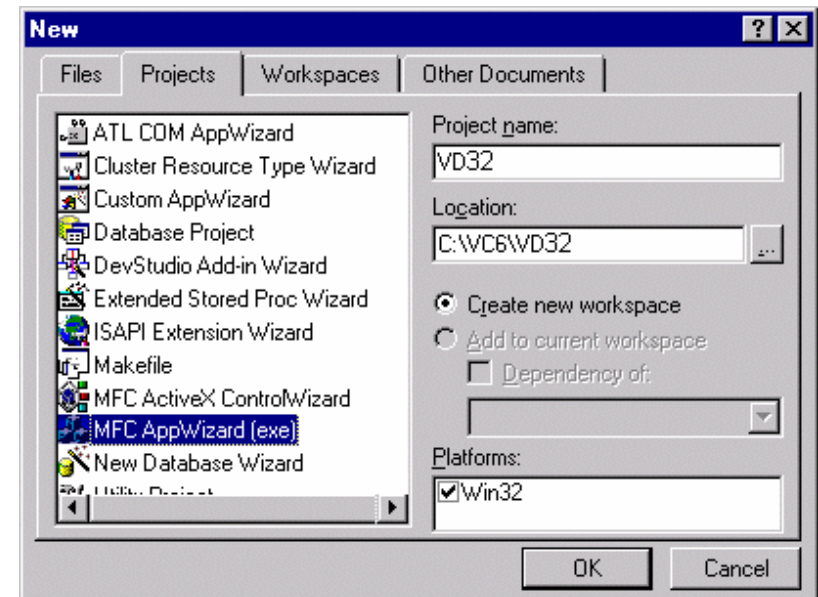
```
BOOL CEmpFrame::OnCmdMsg (
    UINT nID, int nCode, void *pExtra,
    AFX_CMDHANDLERINFO *pHandlerInfo )
{
    // Dispatch command messages to m_view
    if (m_view.OnCmdMsg(nID, nCode, pExtra, pHandlerInfo))
        return TRUE;    // message was processed by m_view

    return CFrameWnd::OnCmdMsg ( nID, nCode, pExtra,
        pHandlerInfo );
}
```

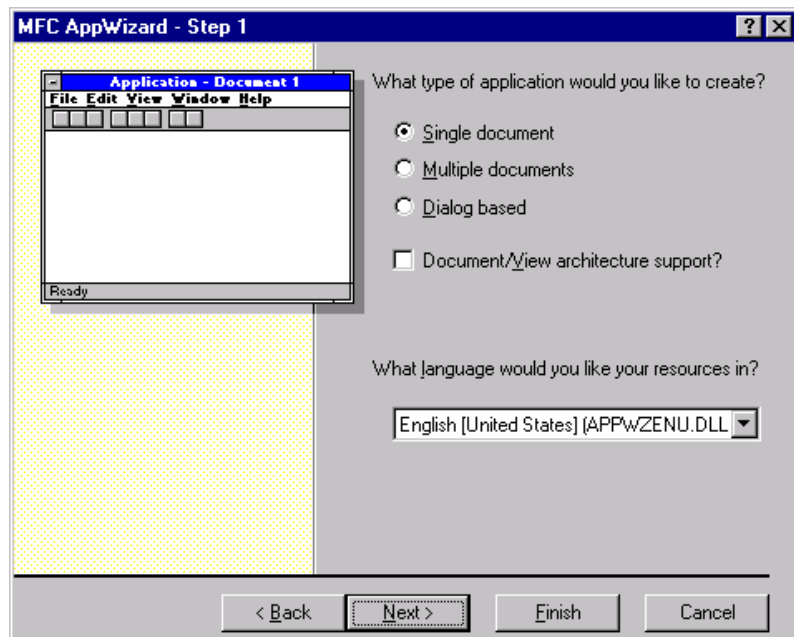
10.5.3 Dùng MFC wizard tạo ứng dụng với giao diện chính framewindow:

Để người dùng có ngay một dự án cỡ VD32 mà không phải mất công thực hiện các công việc như trên, MFC wizard cung cấp chức năng hỗ trợ tạo nhanh dự án với cửa sổ chính là một frame window. Cách sử dụng chức năng hỗ trợ này như sau:

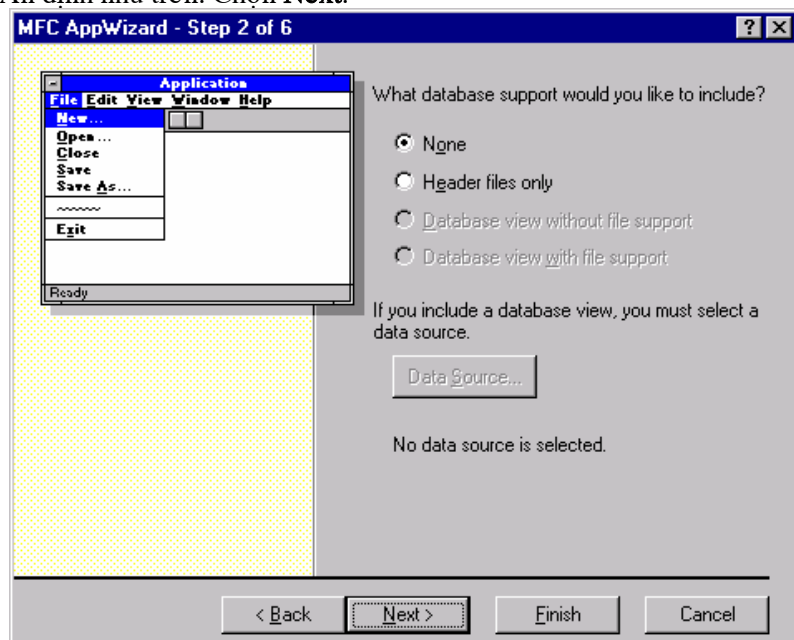
- Chọn menu **File / New**.



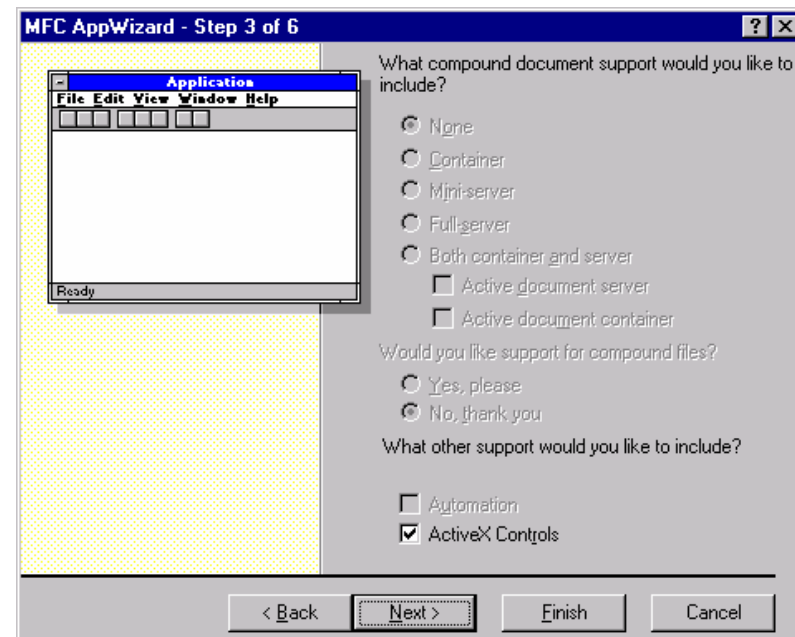
- Điền các thông tin trong hộp hội thoại **New**, Sau đó chọn **OK**.



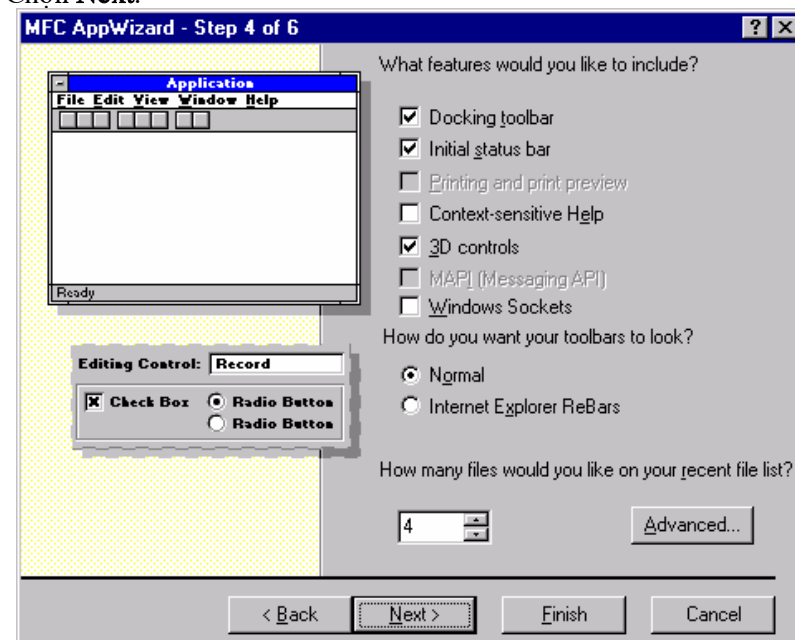
- Ấn định như trên. Chọn **Next**.



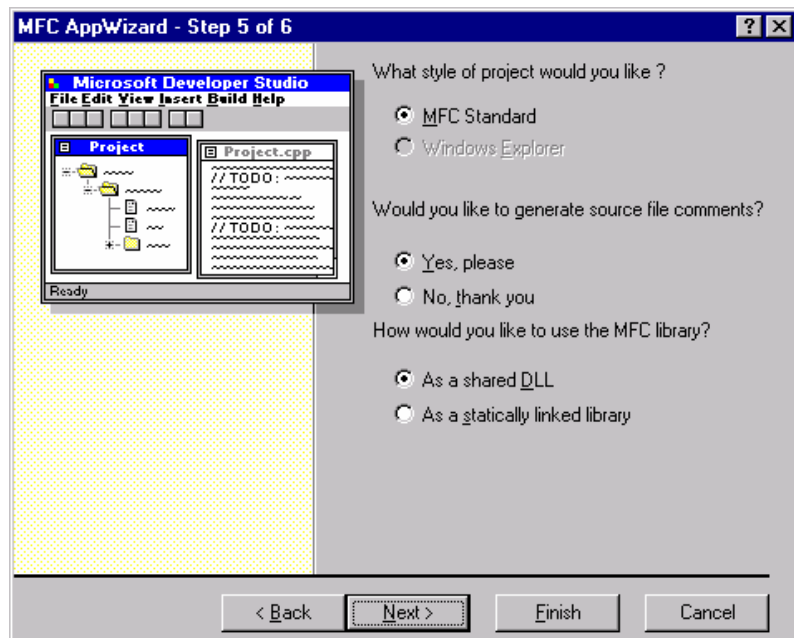
- Ấn định như trên. Chọn **Next**.



- Chọn **Next**.



- Chọn **StatusBar** và **ToolBar**. Sau đó chọn **Next**



- Chọn cơ chế liên kết với thư viện MFC. Sau đó chọn **Next**.
- Cuối cùng, ấn định tên tập tin chứa khai báo các lớp. Chọn **Finish**.

THỰC HÀNH:

1. Tương tự VD33. Khi người dùng chọn mục Option thì hiển thị dialog với hai mục nhập: nội dung chuỗi chạy và tốc độ chạy chữ. Nếu người dùng click chọn OK thì thông tin nhập trong dialog sẽ được áp dụng cho dòng chữ chạy trong view.

2. Tương tự VD33. Thay hộp hiển thị câu thông báo "Mr.Emp" bằng một chiếc đồng hồ có dạng hh:mm:ss chạy theo thời gian lưu trong máy.

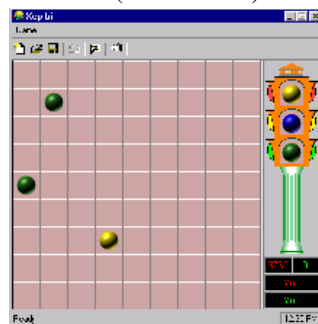
HD: Cài timer cho CEmpFrame. Sử dụng CTime của MFC (xem VD34).

3. Cài đặt mục progressbar trên thanh statusbar của ứng dụng.

4. Viết ứng dụng với dialog giao diện chính có thanh trạng thái.

5. Viết ứng dụng với dialog giao diện chính có thanh công cụ và thanh trạng thái.

6. Quan sát trò chơi line. Phân tích và thực hiện ứng dụng này trong VC.



Các KIẾN TRÚC Document & View

MFC hỗ trợ mạnh mẽ các dự án sử dụng dữ liệu, bao gồm dữ liệu dạng văn bản (text), dạng văn bản kèm hình ảnh và định dạng (lưu dưới dạng văn bản – rich text format và lưu dưới dạng mã nhị phân – compound file), dạng lưu trữ trên các hệ quản trị cơ sở dữ liệu và truy xuất thông qua ODBC, ADO.

Ứng dụng sử dụng dữ liệu có thể dễ dàng phát triển trên VC nhờ MFC cung cấp bộ khung chuẩn cho các dự án thao tác dữ liệu. Bộ khung này bao gồm ba thành phần: Document-View-Frame (DVF). Trong đó:

- **Document:** Quản lý toàn bộ nội dung dữ liệu của ứng dụng được lưu trữ trong bộ nhớ.
- **View:** Thực hiện chức năng hiển thị và quản lý một phần dữ liệu của document.
- **Frame Window:** Chứa view, điều phối command message từ người dùng đến các view một cách thích hợp.

Bộ khung này được thực hiện thông qua các lớp liên quan sau đây.

11.1 CDocument:

CDocument là lớp đối tượng quản lý một nội dung dữ liệu (Document).

- CDocument(); Tạo lập đối tượng rỗng.
- void **AddView** (
 - CView* *pView* // Con trỏ đối tượng view
); Bổ sung view vào danh sách các view của document.
- void **RemoveView** (
 - CView* *pView* // Con trỏ đối tượng view
); Loại bỏ một view ra khỏi danh sách các view của document.
- void **UpdateAllViews** (
 - CView* *pSender*, // Đối tượng view miễn cập nhật
 - LPARAM *lHint* = 0L, // Số hiệu cập nhật
 - CObject* *pHint* = NULL // Cấu trúc chứa thông tin cập nhật
); Thông báo các view trong danh sách view của document cập nhật lại nội dung hiển thị.
- const CString& **GetTitle**(); Trả về nội dung tiêu đề của dữ liệu.
- virtual void **DeleteContents**(); Xóa rỗng nội dung document.

- virtual BOOL **OnNewDocument**(); Hành vi được thực hiện mỗi khi đối tượng CDocument được tạo mới; kế thừa để cài đặt khởi tạo riêng.
- virtual BOOL **OnOpenDocument** (
 - LPCTSTR *lpszPathFileName* // Đường dẫn và tên tập tin
); Hành vi được thực hiện khi đối tượng CDocument chuẩn bị nhận nội dung từ tập tin; kế thừa để cài đặt xử lý riêng về việc đọc tập tin.
- virtual BOOL **OnSaveDocument** (
 - LPCTSTR *lpszPathName* // Đường dẫn và tên tập tin
); Hành vi được thực hiện khi nội dung của document sắp được lưu vào tập tin. Việc kế thừa hành vi này nhằm thực hiện xử lý lưu trữ theo cách riêng của ứng dụng.
- BOOL **IsModified**(); Trả về tình trạng cập nhật dữ liệu trong document ; =TRUE (có chỉnh sửa) hoặc =FALSE (không có chỉnh sửa).

11.2 CView:

CView, lớp kế thừa từ lớp CWnd, giúp quản lý thành phần view của ứng dụng. Trong bộ ba DVF, view là thành phần giao diện quan trọng cho phép người dùng thao tác dữ liệu của ứng dụng một cách dễ dàng và hiệu quả.

- **CView**(); Tạo lập đối tượng view.
 - virtual void **OnInitialUpdate**(); Hành vi được thực hiện khi đối tượng view được kết vào danh sách các đối tượng view của document. Việc kế thừa nhằm cài đặt các ấn định khởi tạo thông số cho view.
 - virtual void **OnActivateView** (
 - BOOL *bActivate*, // =TRUE: kích hoạt ; và ngược lại
 - CView* *pActivateView*, // Đối tượng view được kích hoạt
 - CView* *pDeactivateView* // Đối tượng view bị ngừng hoạt động
); Hành vi được thực hiện mỗi khi đối tượng view được kích hoạt hoặc thôi kích hoạt.
 - virtual void **OnActivateFrame** (
 - UINT *nState*, // Trạng thái kích hoạt
 - CFrameWnd* *pFrameWnd* // Con trỏ đối tượng frame chứa view
); Hành vi được thực hiện mỗi khi đối tượng frame chứa view được kích hoạt.
- Trạng thái kích hoạt của frame chứa view có thể là:
- WA_INACTIVE : Frame chứa view ngừng hoạt động.
 - WA_ACTIVE : Frame chứa view được kích hoạt.

- CDocument* **GetDocument**(); Trả về con trỏ đối tượng CDocument đang sử dụng view.
- virtual void **OnUpdate** (
 - CVIEW* *pSender*, // Đối tượng view làm thay đổi dữ liệu
 - LPARAM *IHint*, // Tương tự UpdateAllViews()
 - CObject* *pHint* // của CDocument
); Hành vi được thực hiện khi nội dung dữ liệu trong document liên quan đến view được cập nhật.
- BOOL **DoPreparePrinting**(CPrintInfo* *pInfo*); Mở hộp in ấn.

11.3 CFrameWnd:

CFrameWnd là lớp đối tượng quản lý khung cửa sổ giao diện chính của ứng dụng (chương 10). Tham gia vào bộ ba DVF của ứng dụng sử dụng dữ liệu, lớp đối tượng CFrameWnd có các hoạt động xử lý bổ sung như sau:

- Khi frame được kích hoạt, nó gọi hành vi OnActivateView của view.
- Nếu frame được khởi tạo với thông số dạng FWS_ADDTOTITLE thì tiêu đề của dữ liệu (document) hiển thị trong view thuộc frame sẽ được đưa lên tiêu đề của frame.
- Frame thực hiện điều phối command message cho view.

11.4 CDocTemplate:

CDocTemplate là lớp đối tượng quản lý bộ ba DVF của MFC. Bộ ba này bao gồm CDocument – CView – CFrameWnd.

- **CDocTemplate** (
 - UINT *nIDResource*, // Số hiệu các resource liên quan:
 - CRuntimeClass* *pDocClass*, // menu, icon, phím tắt, stringTable
 - CRuntimeClass* *pFrameClass*,
 - CRuntimeClass* *pViewClass*
); Tạo lập và khởi tạo thông số cho đối tượng CDocTemplate.
 - *pDocClass*: Con trỏ đối tượng CRuntimeClass quản lý thông tin của lớp document tại thời điểm thực thi chương trình.
 - *pFrameClass*: Con trỏ đối tượng CRuntimeClass quản lý thông tin của lớp frame window tại thời điểm thực thi chương trình.
 - *pViewClass*: Con trỏ đối tượng CRuntimeClass quản lý thông tin của lớp view tại thời điểm thực thi chương trình.
 Con trỏ chứa thông tin thi hành của một lớp nhận được từ macro sau:

CRuntimeClass* **RUNTIME_CLASS** (Tên_Lớp)

- virtual BOOL **GetDocString** (
 - CString& *rString*, // Biến chuỗi chứa kết quả
 - enum DocStringIndex *index* // Chỉ số mục thông tin cần lấy.
);

Chỉ số mục thông tin cần lấy có thể là:
 - CDocTemplate:: *windowTitle* : Tiêu đề
 - CDocTemplate:: *docName* : Tên document.
 - CDocTemplate:: *fileNewName* : Tên mặc nhiên dùng cho tập tin dữ liệu được tạo mới.

☞ Cả CDocument và CView trong bộ DVF đều có thể đọc, ghi dữ liệu trên đối tượng lưu trữ dữ liệu thông qua hành vi kế thừa được: **Serialize**.

```
virtual void Serialize( CArchive& ar );
throw( CMemoryException ); // Dừng khi ar liên quan bộ nhớ trong.
throw( CArchiveException ); // Dừng khi ar là Archive trên đĩa.
throw( CFileException ); // Dừng khi ar là File.
```

11.5 HỖ TRỢ TỪ PHÍA ĐỐI TƯỢNG QUẢN LÝ ỨNG DỤNG:

Lớp CWinApp có các hành vi liên quan việc khởi tạo ứng dụng sử dụng dữ liệu và thao tác dữ liệu dựa trên bộ khung DVF như sau:

- void **AddDocTemplate** (
 - CDocTemplate* *pTemplate* // Con trỏ đối tượng quản lý bộ DVF
); Đưa một bộ ba DVF vào danh sách dữ liệu quản lý bởi ứng dụng.
- BOOL **ProcessShellCommand** (CCommandLineInfo& *rCmdInfo*);

Thực hiện tác vụ xử lý dữ liệu theo yêu cầu của **system shell**. Ứng dụng chỉ cho phép xử lý dữ liệu (tập tin) phù hợp với chức năng của các bộ DVF trong danh sách dữ liệu quản lý bởi ứng dụng.

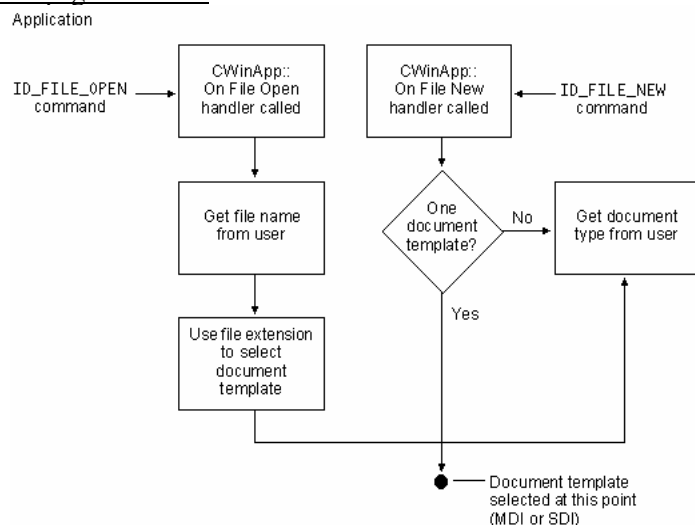
rCmdInfo : Chứa các thông số dòng lệnh (nếu có) bao gồm tên dữ liệu, lệnh thao tác (edit, open, ...) trên dữ liệu do **shell** chuyển đến.
- void **ParseCommandLine** (CCommandLineInfo& *rCmdInfo*);

Chuẩn bị thông số cần thiết trong *rCmdInfo* để chuyển cho hành vi **ProcessShellCommand**. Các thông số này tương ứng với nội dung tham số dòng lệnh của ứng dụng.
- afx_msg void **OnFileNew**(); Hành vi trả lời cho WM_COMMAND được phát ra bởi mục menu có số hiệu ID_FILE_NEW.
- afx_msg void **OnFileOpen**(); Hành vi trả lời cho WM_COMMAND được phát ra bởi mục menu có số hiệu ID_FILE_OPEN.

- `afx_msg void OnFilePrintSetup();` Hành vi xử lý `WM_COMMAND` được phát ra bởi mục menu có số hiệu `ID_FILE_PRINT_SETUP`.

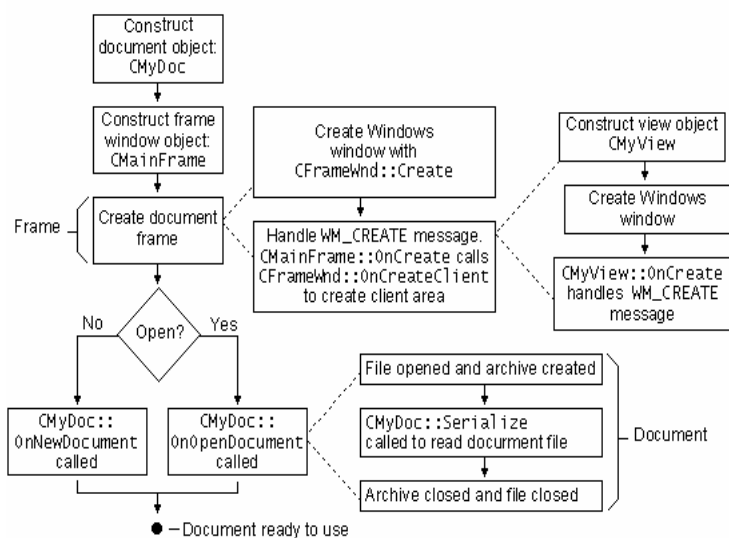
11.6 TRÌNH TỰ TẠO LẬP CÁC ĐỐI TƯỢNG THAM GIA BỘ DVF:

❖ Đối tượng document:



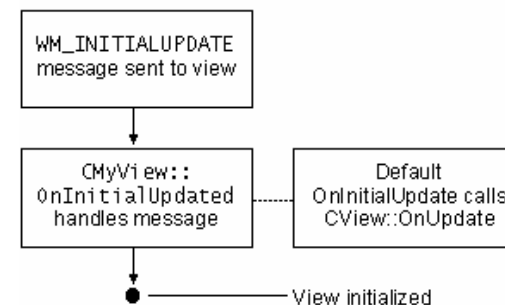
❖ Đối tượng Frame window:

Document Template: `OpenDocumentFile`



❖ Đối tượng view:

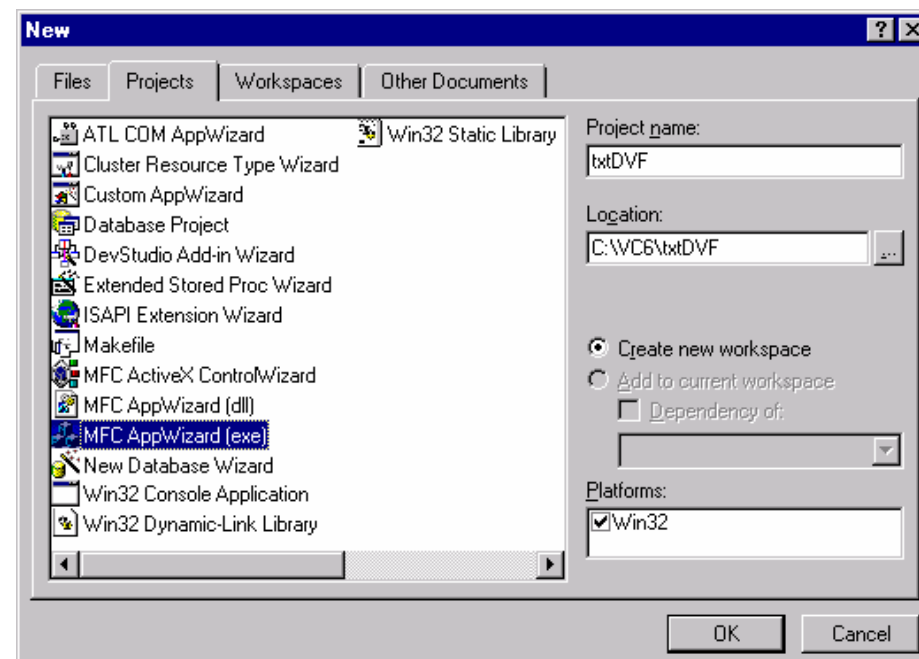
View



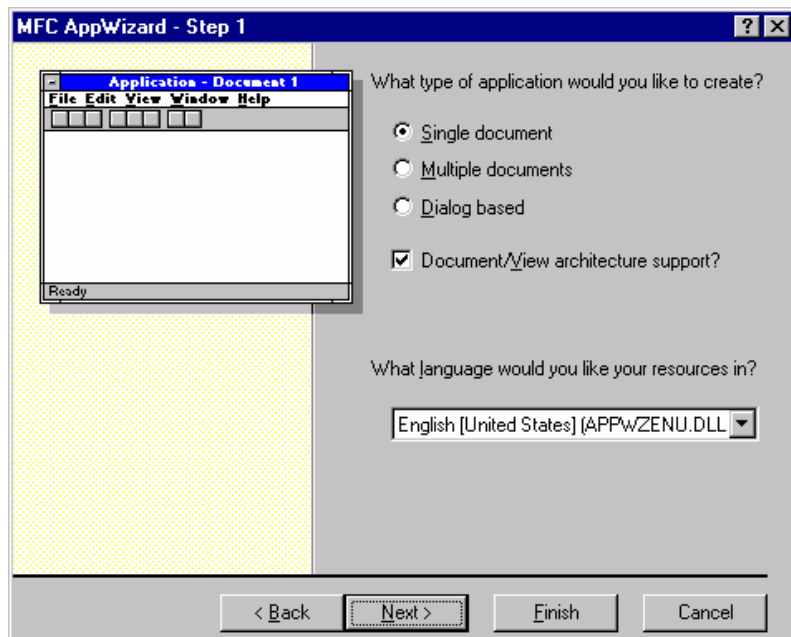
➤ Nhằm hỗ trợ người dùng thực hiện các dự án liên quan đến việc sử dụng dữ liệu, MFC cung cấp các bộ DVF phổ biến và cài đặt các bộ này trong phần hỗ trợ MFC Wizard. Sau đây là một số bộ DVF đặc trưng của MFC.

11.7 TEXT DOCUMENT APPLICATION:

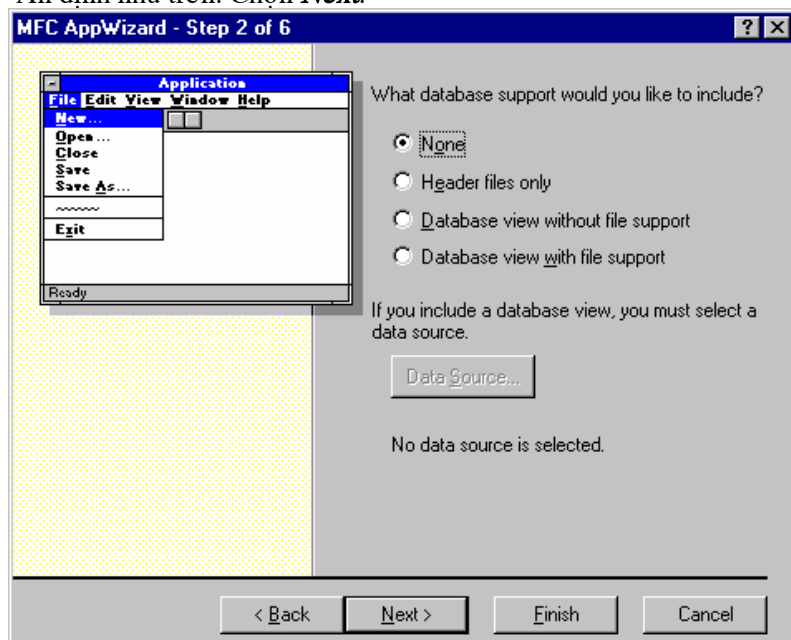
- Chọn **File / New:**



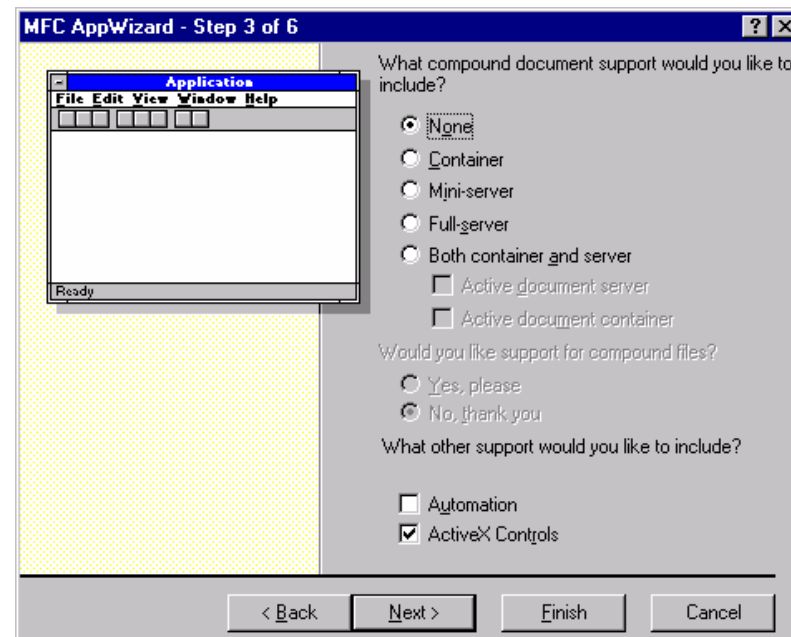
- Ấn định như trên. Chọn **OK**.



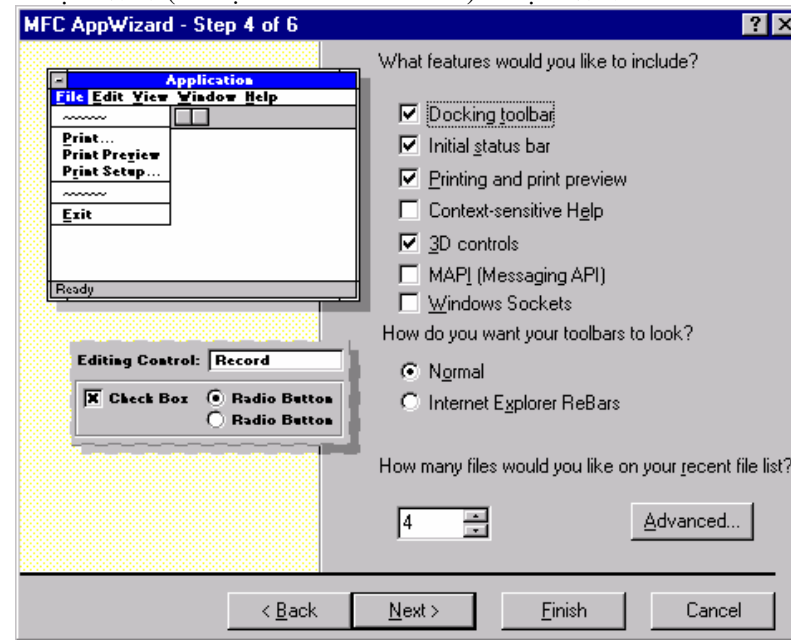
- Ấn định như trên. Chọn **Next**.



- Chọn **None** (không sử dụng cơ sở dữ liệu). Chọn **Next**.

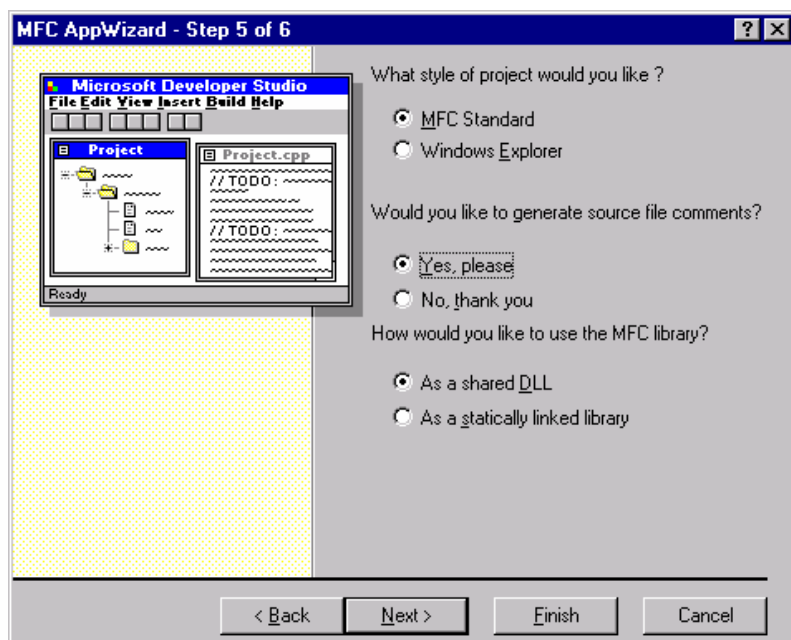


- Chọn **None** (dữ liệu chỉ chứa văn bản). Chọn **Next**.

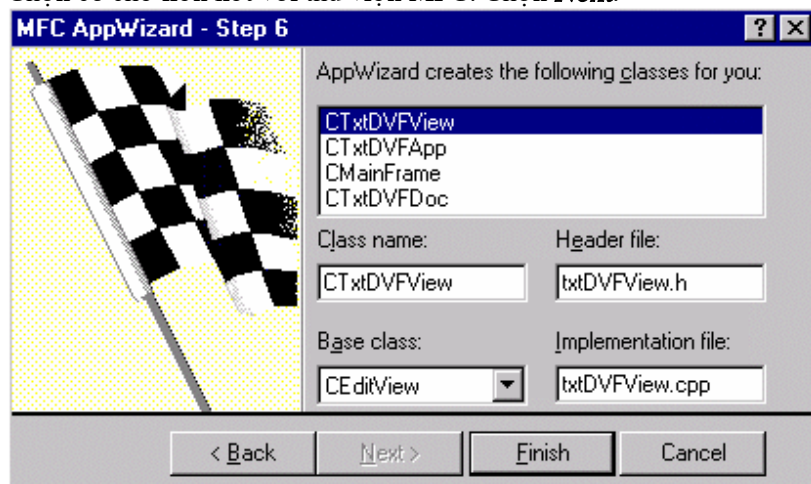


- Ấn định các mục cần cần thiết:

- **Printing and print preview:** Cho phép chức năng in ấn.
 - **MAPI:** Sử dụng dịch vụ mail cho dữ liệu của ứng dụng.
- Chọn **Next**



- Chọn **MFC Standard**: Ứng dụng có giao diện bình thường hoặc **Windows Explorer**: Ứng dụng có giao diện như *windows explorer*. Chọn cơ chế liên kết với thư viện MFC. Chọn **Next**.

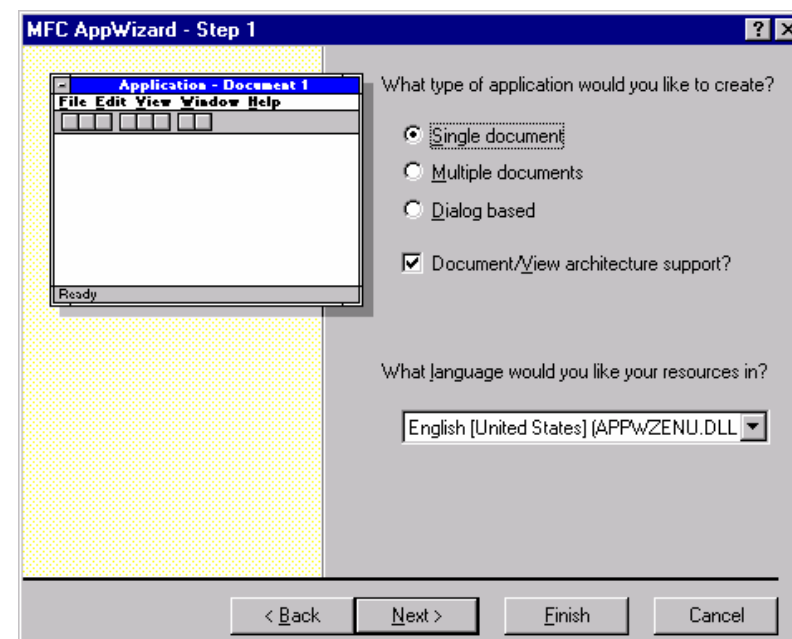


- Ấn định tên các tập tin chứa khai báo và cài đặt của các lớp.
Lưu ý: Chọn lớp CtxtDVFView, khai báo lớp cơ sở là CEditView để màn hình view cho phép soạn thảo. Sau cùng chọn **Finish**.
Ứng dụng nhận được có thể soạn thảo và quản lý dữ liệu văn bản.

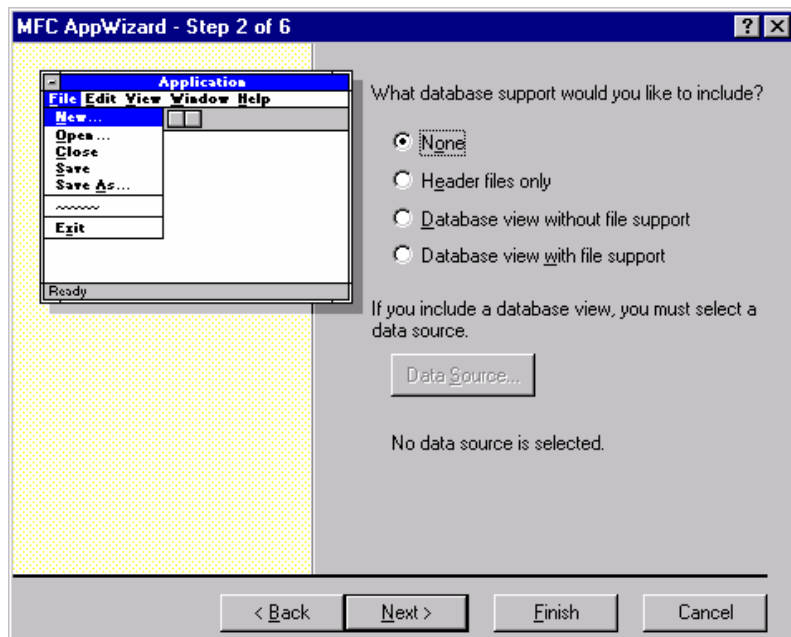
☞ Kế thừa hành vi **Serialize**(CArchive *ar*) của lớp view, dựa trên giá trị trả về từ hành vi **IsStoring**() của đối tượng tham số *ar*, chúng ta có thể tự xử lý đọc/ghi dữ liệu theo cấu trúc lưu trữ riêng.

11.8 **RICH TEXT FORMAT (RTF) DOCUMENT APPLICATION:**

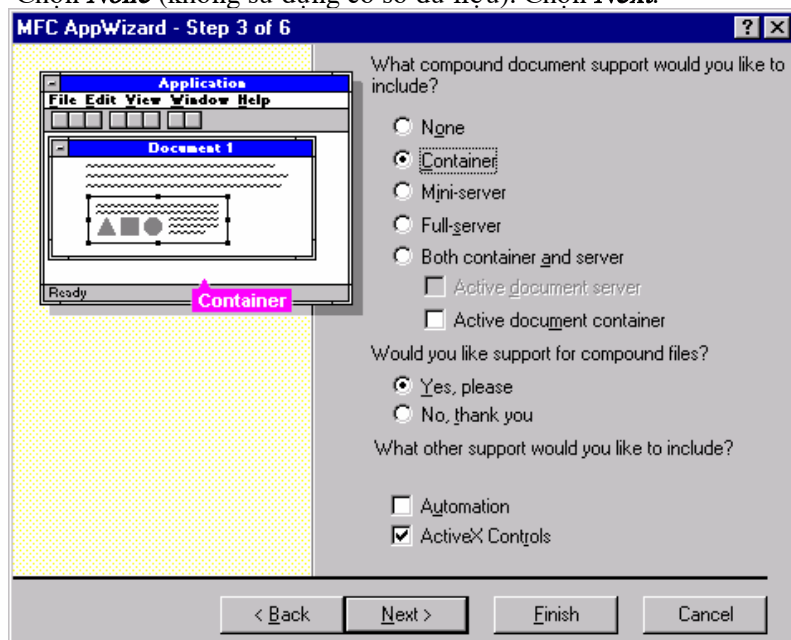
- Chọn **File / New**:
- Trong hộp thoại **New**:
 - **Project type** = *MFC AppWizard*.
 - **ProjectName** = *rtfDVF* (tên dự án).
 - **Location** = Thư mục chứa dự án.
 Sau đó chọn **OK**.



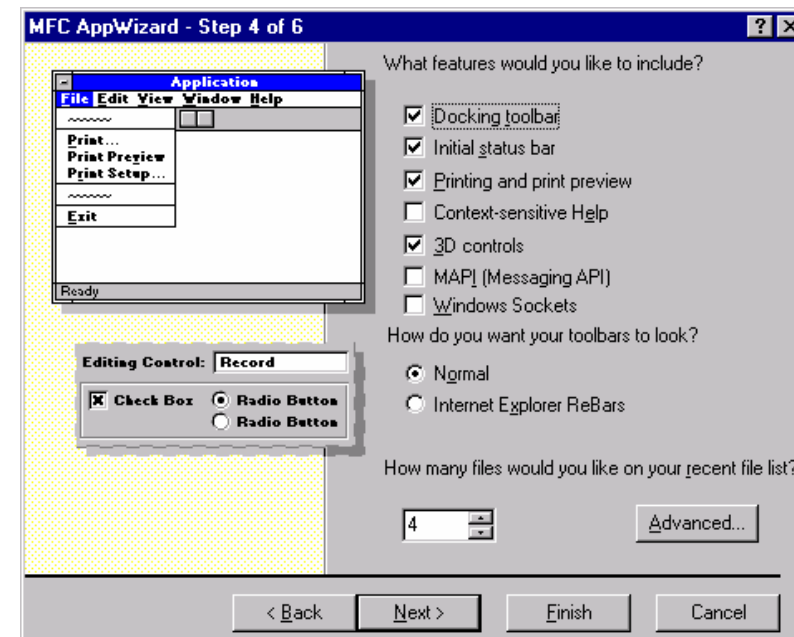
- Ấn định như trên. Chọn **Next**.



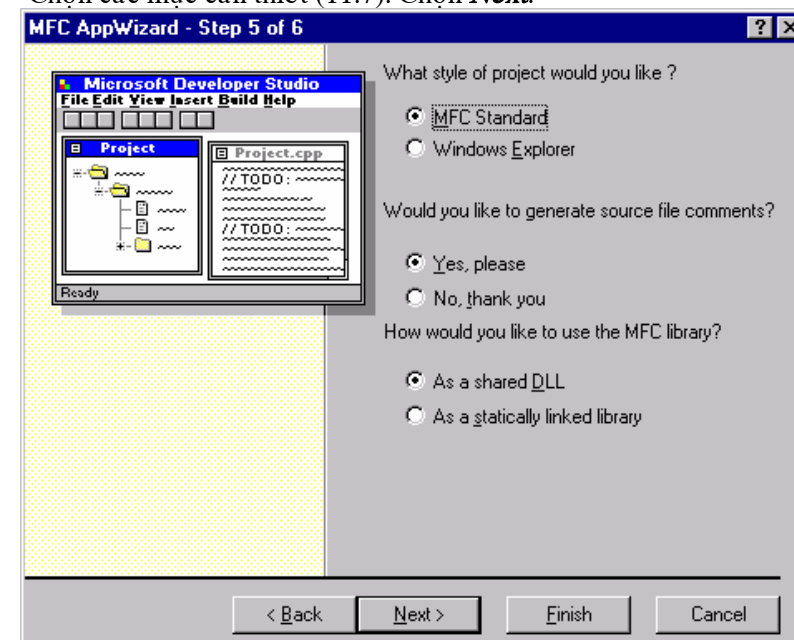
- Chọn **None** (không sử dụng cơ sở dữ liệu). Chọn **Next**.



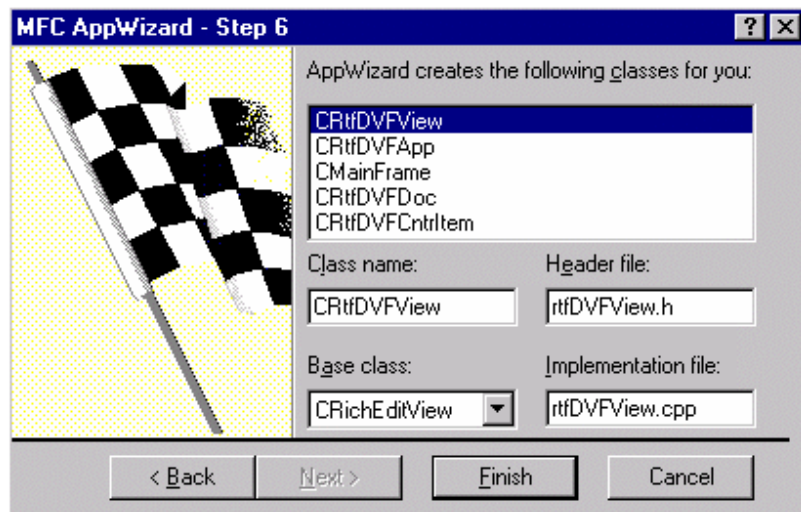
- Ứng dụng sử dụng OLE (Container) từ ứng dụng khác. Chọn **Next**.



- Chọn các mục cần thiết (11.7). Chọn **Next**.



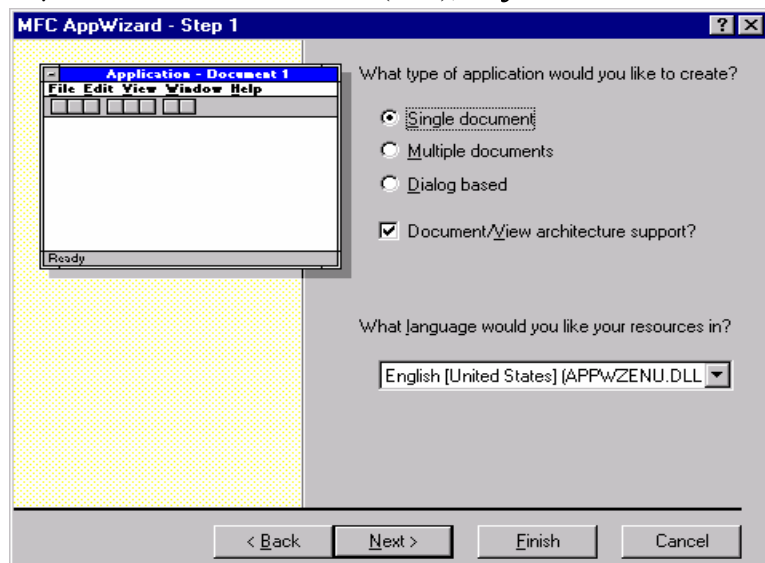
- Chọn các mục cần thiết (11.7). Chọn **Next**.



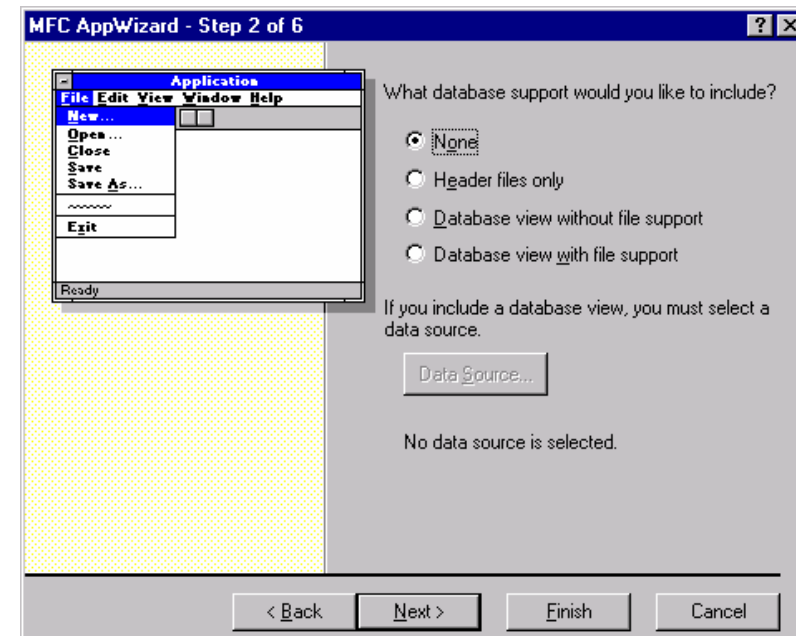
- Ấn định tên các tập tin chứa khai báo và cài đặt của các lớp.
Lưu ý: Chọn lớp CRtfDVFView, khai báo lớp cơ sở là CRichEditView để màn hình view cho phép soạn thảo và liên kết với các đối tượng OLE. Sau cùng chọn **Finish**. Ứng dụng làm việc với dữ liệu rtf.

11.9 HTML DOCUMENT VIEW APPLICATION:

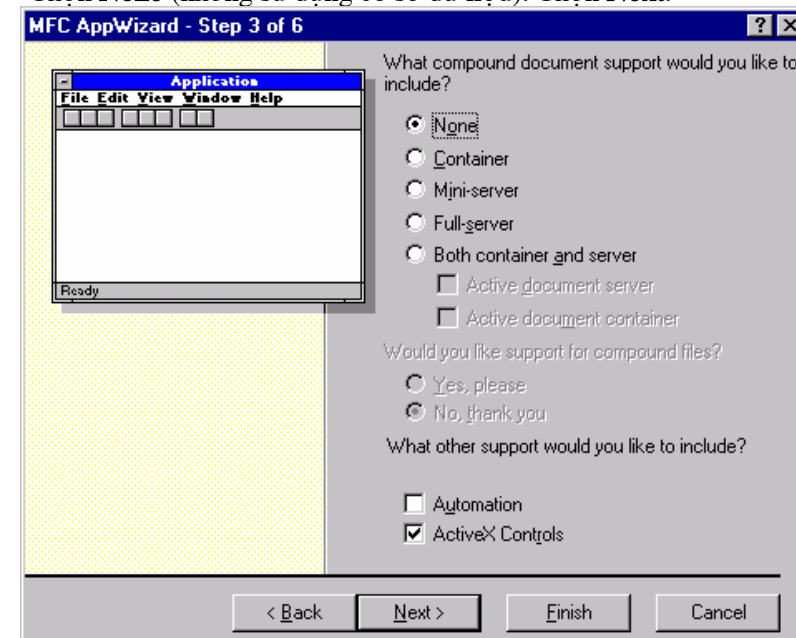
- Chọn **File / New**. Khởi đầu như (11.8); *ProjectName* = *HtmlDVF*.



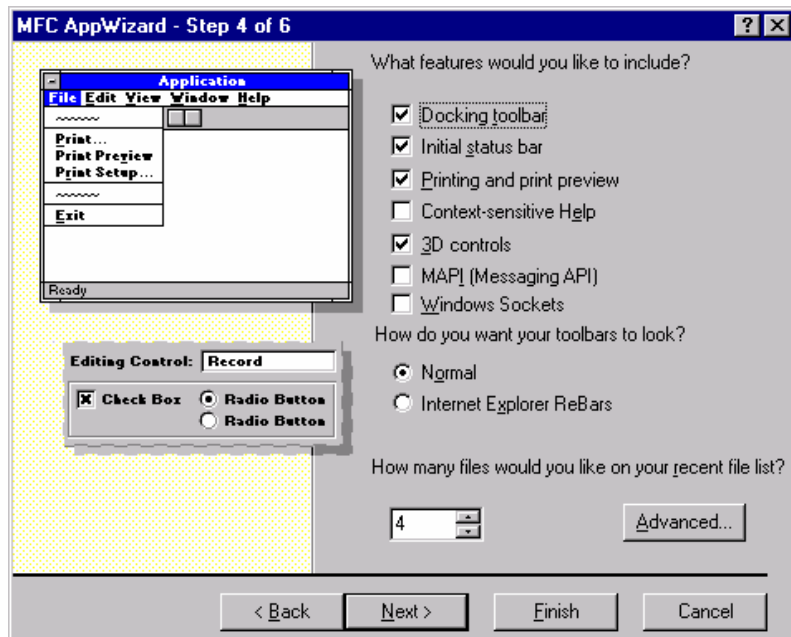
- Ấn định như trên. Chọn **Next**.



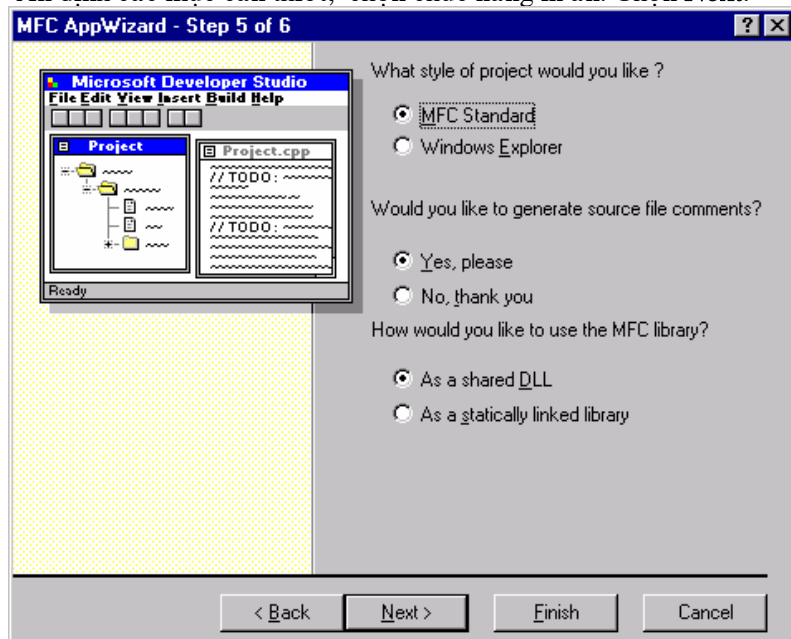
- Chọn **None** (không sử dụng cơ sở dữ liệu). Chọn **Next**.



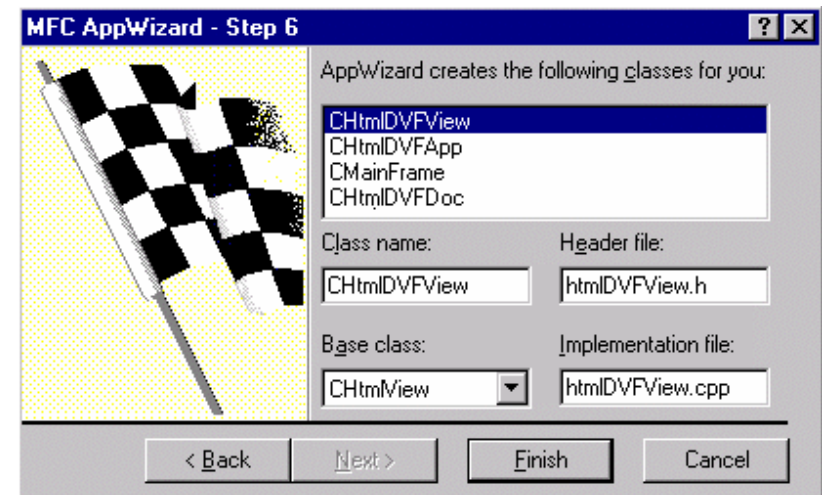
- Chọn **None**. Chọn **Next**.



- Ấn định các mục cần thiết; chọn chức năng in ấn. Chọn **Next**.



- Chọn các mục cần thiết (11.7). Chọn **Next**.



- Ấn định tên các tập tin chứa khai báo và cài đặt của các lớp.
Lưu ý: Chọn lớp CHtmlDVView, khai báo lớp cơ sở là CHtmlView để màn hình view hiển thị được nội dung trang HTML. Chọn **Finish**.
- Ấn định URL: Hành vi **OnInitialUpdate** của lớp CHtmlDVView thực hiện ấn định URL (ví dụ: www.hcmueco.edu.vn) cho **Navigate2**. Ứng dụng nhận được có thể đảm nhận công việc Browser đơn giản.

☞ Để cài đặt thành công cụ như chương trình Internet Explorer cho ứng dụng, ở bước 'Step 4 of 6' ta chọn mục: **Internet Explorer Rebars**.



11.10 MỘT SỐ LỚP VIEW ĐẶC BIỆT:

11.10.1 CListView:

CListView là lớp đối tượng quản lý view dạng danh sách (list).

- **CListView()**; Tạo lập đối tượng view.
- **CListCtrl& GetListCtrl ()**; Trả về đối tượng **CListCtrl** làm cơ sở cho các tác vụ liên quan ListView.

☞ Các hành vi đặc trưng của **CListCtrl**:

- **BOOL SetBkColor(COLOREF cr)**; Đặt màu nền cho listview.
- **COLORREF GetBkColor()**; Trả về giá trị màu nền của listview.

- **CImageList* SetImageList** (
 CImageList* *pImagelist*, // Đối tượng imagelist
 int *imgStyle* // Thông số qui định cách sử dụng
); Chọn đối tượng imagelist chứa ảnh dùng cho các mục của listview.
 Cách sử dụng có thể là:
 LVSIL_NORMAL : Ảnh bình thường.
 LVSIL_SMALL : Ảnh nhỏ.
- **int InsertItem** (
 int *nItem*, // Chỉ số mục được thêm
 LPCTSTR *lpszItem*, // Nội dung thông báo của mục
 int *nImage* // Chỉ số ảnh trong Imagelist mà mục sử dụng
); Thêm một mục vào listview.
- **BOOL DeleteItem** (
 int *nItem* // Số hiệu của mục
); Xóa một mục trong listview.
- **BOOL DeleteAllItems**(); Xóa rỗng listview.
- **BOOL GetItem** (
 LVITEM* *pItem* // Con trỏ đến cấu trúc nhận thông tin
); Lấy thông tin liên quan đến mục có số thứ tự *nItem*.
- **BOOL SetItem** (
 LVITEM* *pItem* // Con trỏ đến cấu trúc chứa thông số
); Đặt thông số cho mục có chỉ số là *pItem->iItem*.
- **BOOL EnsureVisible** (
 int *nItem*, // Chỉ số phần tử cần nhìn thấy.
 BOOL *bPartialIsOK* // =FALSE: Toàn bộ, =TRUE: Một phần
); Cuộn danh sách để nhìn thấy phần tử *nItem* nếu phần tử này không được nhìn thấy trong vùng hiển thị cho phép của view.

11.10.2 CTreeView:

- CTreeView là lớp đối tượng quản lý màn hình view có cấu trúc cây.
- **CTreeView**(); Tạo lập đối tượng treeview.
 - **CTreeCtrl& GetTreeCtrl**(); Trả về đối tượng **CTreeCtrl** làm cơ sở cho các tác vụ liên quan TreeView.
- 📖 Các hành vi đặc trưng của CTreeCtrl:
- **BOOL SetBkColor**(COLOREF *cr*); Đặt màu nền cho treeview.
 - **COLORREF GetBkColor**(); Trả về giá trị màu nền của treeview.
 - **CImageList* SetImageList** (
 CImageList* *pImagelist*, // Con trỏ đối tượng imagelist
 int *imgStyle* // Thông số qui định cách sử dụng
); Chọn đối tượng imagelist chứa ảnh dùng cho các mục của treeview.
 Cách sử dụng có thể là:
 TVSIL_NORMAL : Ảnh dùng cho các mục bình thường.
 TVSIL_STATE : Ảnh dùng cho các mục đặc trưng do người dùng định nghĩa.

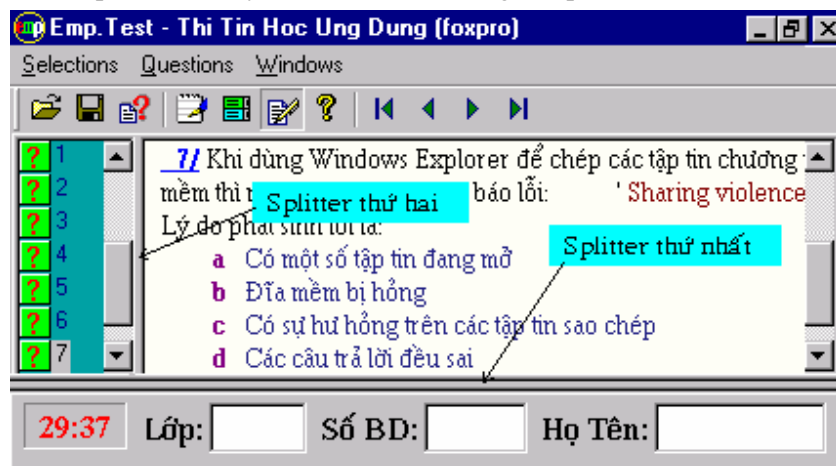
- **UINT GetCount**(); Trả về số mục của treeview.
- **HTREEITEM GetRootItem**(); Trả về handle của phần tử đầu gốc.
- **HTREEITEM GetFirstVisibleItem**(); Trả về handle của phần tử hiển thị đầu tiên trong vùng nhìn thấy của treeview (NULL: Không có).
- **HTREEITEM GetSelectedItem**(); Trả về handle của phần tử đang được chọn.
- **HTREEITEM GetNextVisibleItem**(HTREEITEM *hItem*); Trả về handle của phần tử hiển thị kế sau phần tử có handle là *hItem*.
- **HTREEITEM GetPrevVisibleItem**(HTREEITEM *hItem*); Trả về handle của phần tử hiển thị kế trước phần tử có handle là *hItem*.
- **HTREEITEM InsertItem** (
 LPCTSTR *lpszItem*, // Nội dung thông báo của mục
 int *nImage*, // Chỉ số ảnh dùng cho mục ở trạng thái
 int *nImageSelected*, // bình thường và khi mục được chọn.
 HTREEITEM *parent* = TVI_ROOT, // Con trỏ mục cha
 HTREEITEM *hInsertAfter* = TVI_LAST // Con trỏ mục đứng trước
); Thêm một mục vào treeview, trả về handle của phần tử mới thêm.
- **BOOL DeleteItem** (
 HTREEITEM *hItem* /* handle của mục*/); Xóa mục của treeview.
- **BOOL DeleteAllItems**(); Xóa rỗng nội dung treeview.
- **BOOL GetItem** (
 TVITEM* *pItem* // Con trỏ đến cấu trúc nhận thông tin
); Lấy thông tin của phần tử *pItem->iItem*.
- **BOOL SetItem** (
 TVITEM* *pItem* // Con trỏ đến cấu trúc chứa thông số
); Đặt thông số cho phần tử *pItem->iItem*.
- **BOOL SetItemImage** (
 HTREEITEM *hItem*, // handle của phần tử
 int *nImage*, // Chỉ số ảnh dùng cho mục ở trạng thái


```
int nSelectedImage // bình thường và khi mục được chọn.
); Ấn định chỉ số ảnh trong imagelist dùng cho mục.
```

- **BOOL SetItemText (**
 HTREEITEM *hItem*, // handle của mục
 LPCTSTR *lpszItem* // Nội dung thông báo
); Ấn định lại nội dung thông báo của mục trong treeview.

11.10.3 CSplitterWnd:

Vùng client trong frame window cho phép cài đặt một cửa sổ view duy nhất. Để lồng được nhiều view vào frame ta phải phân chia vùng client của frame. Việc phân chia này được hỗ trợ bởi công cụ splitter window.



Mỗi splitter window cho phép tách vùng client của frame window thành nhiều hàng và cột. Sau đó, mỗi ô (pane) nhận được từ splitter này có thể lại được tách ra thành nhiều hàng và cột bởi một splitter khác. Trong ví dụ trên:

- Splitter thứ nhất tách frame window thành hai dòng và 1 cột.
- Splitter thứ hai tách pane thứ nhất của splitter thứ nhất thành 2 cột và 1 dòng.

Mỗi pane nhận được từ các splitter window cho phép gắn một màn hình view. Như vậy, thông qua các splitter window, frame window có thể chứa nhiều màn hình view đồng thời.

Nhằm tiện việc thao tác với công cụ splitter window, MFC cung cấp lớp đối tượng CSplitterWnd cho phép quản lý các splitter window trong ứng dụng. Các hành vi đặc trưng của lớp CSplitterWnd như sau:

- **CSplitterWnd ();** Tạo lập đối tượng splitter window.

- **BOOL Create (**
 CWnd* *pParentWnd*, // Con trỏ đối tượng cửa sổ cha
 int *nMaxRows*, // Số hàng tối đa của các pane
 int *nMaxCols*, // Số cột tối đa của các pane
 SIZE *sizeMin*, // Kích thước tối thiểu của mỗi pane
 CCreateContext* *pContext*, // Thông số liên kết, lấy từ frame
 DWORD *dwStyle* = WS_CHILD | WS_VISIBLE | WS_HSCROLL |
 WS_VSCROLL | SPLS_DYNAMIC_SPLIT,
 UINT *nID* = AFX_IDW_PANE_FIRST
); Khởi tạo thông số splitter window với số hàng, cột thay đổi được.
- **BOOL CreateStatic (**
 CWnd* *pParentWnd*, // Con trỏ đối tượng cửa sổ cha
 int *nRows*, // Số hàng các pane được tạo
 int *nCols*, // Số cột các pane được tạo
 DWORD *dwStyle* = WS_CHILD | WS_VISIBLE, // Dạng và
 UINT *nID* = AFX_IDW_PANE_FIRST // Số hiệu
); Tạo splitter window với số hàng và cột cố định.
- **virtual BOOL CreateView (**
 int *row*, // Chỉ số hàng và
 int *col*, // chỉ số cột của pane trong splitter
 CRuntimeClass* *pViewClass*, // Cấu trúc chứa thông tin lớp view
 SIZE *sizeInit* // Kích thước khởi đầu của pane
); Cài view vào một pane xác định trong splitter window.
 pViewClass : Con trỏ đối tượng CRuntimeClass quản lý thông tin của lớp view tương ứng tại thời điểm thực thi chương trình. Xem (11.4).
- **void SetColumnInfo (**
 int *col*, // Chỉ số cột trong splitter
 int *cxIdeal*, // Độ rộng mong muốn và
 int *cxMin* // độ rộng tối thiểu (tính bằng pixel)
); Ấn định thông số về độ rộng cho cột trong splitter window.
- **void GetColumnInfo (**
 int *col*, // Chỉ số cột
 int& *cxCur*, // Tham biến chứa độ rộng hiện thời
 int& *cxMin* // Tham biến chứa độ rộng tối thiểu
); Lấy thông tin về độ rộng của cột.
- **void SetRowInfo (**

```

    int row,                // Chỉ số hàng
    int cyIdeal,            // Độ cao mong muốn
    int cyMin               // Độ cao tối thiểu
); Ấn định thông số về độ cao cho hàng trong splitter window.
▪ void GetRowInfo (
    int row,                // Chỉ số hàng
    int& cyCur,             // Tham biến chứa độ cao hiện thời
    int& cyMin              // Tham biến chứa độ cao tối thiểu
); Lấy thông tin về độ cao của hàng.
▪ void RecalcLayout( ); Cập nhật thông số ấn định mới của splitter.
▪ CWnd* GetPane (
    int row, int col        // Chỉ số hàng và cột của pane
); Trả về con trỏ đối tượng view cài trong pane.

```

11.10.4 SỬ DỤNG SPLITTERWND TRONG FRAME WINDOW:

- Khai báo đối tượng thuộc lớp SplitterWnd như là thuộc tính của frame.

```
CSplitterWnd m_splitter;
```

- Dùng hành vi OnCreateClient của frame để tạo các pane và cài view:

```

BOOL CEmpFrame::OnCreateClient (
                                LPCREATESTRUCT lpcs,
                                CCreateContext* pContext )
{
    if (!CFrameWnd::OnCreateClient(lpcs, pContext))
        return FALSE;

    // Dùng đối tượng splitter tách vùng client của frame
    m_splitter.CreateStatic ( this , 1, 2 ); // Ví dụ: 1 hàng, 2 cột

    // Gắn các view tương ứng vào các pane tạo được:
    m_splitter.CreateView ( 0, 0, RUNTIME_CLASS(viewClass1),
                           CSize(120,0), NULL);
    m_splitter.CreateView(0,1, RUNTIME_CLASS(viewClass2),
                           CSize(0,0), NULL);

    ... // Thực hiện các cài đặt khác
    return TRUE;
}

```

11.10.5 CÁC VÍ DỤ THỰC HÀNH:

📖 **Thực hành 1:** Viết ứng dụng như VD30. Tạo splitter trong CEmpFrame với hai view: bên trái là treeview (dùng lớp view kế thừa từ CTreeView), bên phải là list (dùng lớp view kế thừa từ ListView).

- Tạo dự án VD35 tương tự VD30.
- Đăng ký sử dụng CTreeView và CListView:
Trong tập tin *stdafx.h* của dự án, bổ sung chỉ thị:
#include <afxview.h>
- Bổ sung vào dự án hai lớp mới:

CEmpTree kế thừa từ CTreeView

CEmpList kế thừa từ CListView.

Cách thực hiện tương tự như đã làm với CEmpWnd, mục (5.4.1).

- Khai báo đối tượng thuộc tính protected **m_splitter** kiểu CSplitterWnd.
Hành vi OnCreateClient của CEmpFrame thực hiện khởi tạo và cài đặt các pane, view cần thiết:

```



BOOL CEmpFrame::OnCreateClient(LPCREATESTRUCT lpcs,
                                CCreateContext* pContext )
{
    if (!CFrameWnd::OnCreateClient(lpcs, pContext))
        return FALSE;

    m_splitter.CreateStatic(this, 1, 2);
    m_splitter.CreateView( 0, 0, RUNTIME_CLASS(CEmpTree),
                           CSize(120,0), NULL );
    m_splitter.CreateView(0,1, RUNTIME_CLASS(CEmpList),
                           CSize(0,0), NULL );

    return TRUE;
}

```

📖 **Thực hành 2:** Thực hiện ứng dụng tương tự VD35. Tự động thực hiện bổ sung ba phần tử trong treeview, mỗi phần tử có hai phần tử con. Các phần tử đều có hình minh họa cho trạng thái được chọn và không được chọn.

- Tạo dự án VD36 tương tự VD35.
- Bổ sung bitmap resource chứa hai ảnh cùng kích thước:  . Đặt số hiệu cho bitmap resource là IDB_IMGTREE.
- Hành vi OnCreate của CEmpTree đăng ký sử dụng ảnh và bổ sung các phần tử cần thiết:

```
int CEmpTree::OnCreate(LPCREATESTRUCT lpCreateStruct)
```

```

{
    if (CTreeView::OnCreate(lpCreateStruct) == -1)
        return -1;

    static CImageList img;          // use only in this function
    img.Create(IDB_IMGTREE, 16, 2, RGB(255,255,255));

    CTreeCtrl& myCtrl = GetTreeCtrl();    // Based control
    myCtrl.SetImageList(&img, TVSIL_NORMAL);

    HTREEITEM hihi;                  // Add needed items
    hihi = myCtrl.InsertItem("Muc 1", 0, 1);
        myCtrl.InsertItem("Muc 11", 0, 1, hihi);
        myCtrl.InsertItem("Muc 12", 0, 1, hihi);
    hihi = myCtrl.InsertItem("Muc 2", 0, 1);
        myCtrl.InsertItem("Muc 21", 0, 1, hihi);
        myCtrl.InsertItem("Muc 22", 0, 1, hihi);
    hihi = myCtrl.InsertItem("Muc 3", 0, 1);
        myCtrl.InsertItem("Muc 31", 0, 1, hihi);
        myCtrl.InsertItem("Muc 321", 0, 1,
                                myCtrl.InsertItem("Muc 32", 0, 1, hihi));

    return 0;
}

```



- Hành vi kế thừa PreCreateWindow ấn định thông số dạng treeview:

```

BOOL CEmpTree::PreCreateWindow(CREATESTRUCT& cs) {
    cs.style |= TVS_HASLINES|TVS_LINESATROOT|
               TVS_HASBUTTONS;
    return CTreeView::PreCreateWindow(cs);
}

```


▮ Một số thông số ấn định dạng của treeview:

TVS_HASLINES : Treeview có đường nối giữa các mục.
 TVS_LINESATROOT : Đường nối từ gốc.
 TVS_HASBUTTONS : Có nút mở , đóng  mục.
 TVS_SINGLEEXPAND : Cho phép mở 1 mục duy nhất ở mỗi lúc.
 TVS_XXX : Xem MSDN.

THỰC HÀNH:

1. Tạo ứng dụng soạn thảo văn bản (text). Ứng dụng cho phép người dùng đặt password bảo vệ dữ liệu. Chỉ mở được dữ liệu nếu có password hợp lệ.
2. Tương tự bài tập 1 cho ứng dụng RTF view.
3. Viết ứng dụng HTMLView: Thanh công cụ rebar (như IE của Microsoft); Backward, Forward, hộp combobox nhập và ghi nhớ các URL đã nhập.
4. Phát triển VD36 thành ứng dụng cho phép xem cấu trúc ổ đĩa, cây thư mục trên máy như windows explorer.

HD: Xem hàm FindFirstFile() và FindNextFile().

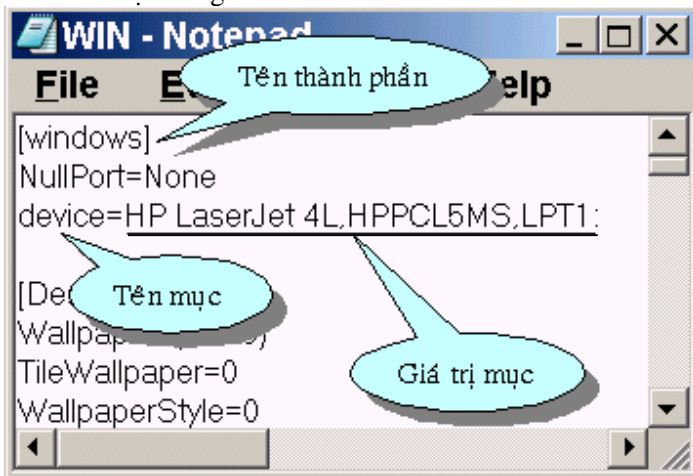
5. Tạo cơ sở dữ liệu Access. Dùng bộ DVF (V = FormView) thích hợp để xem và cập nhật dữ liệu cho các bảng trong cơ sở dữ liệu.
6. Viết ứng dụng RTF View. Trong màn hình view, khi di chuyển chuột lên các nội dung có chỉnh dạng gạch dưới thì thực hiện chuyển dạng con chuột thành , ngược lại chỉnh dạng chuột về dạng mặc nhiên của hệ thống.
7. Viết ứng dụng thi trắc nghiệm đơn giản: Màn hình view rtf cho phép hiển thị văn bản chỉnh dạng và hình ảnh phản ánh nội dung của các câu hỏi và các đáp án lựa chọn. Hiển thị dấu chọn khi người dùng đánh dấu đáp án.
8. Quan sát chương trình tra cứu từ điển. Viết ứng dụng RTF view sử dụng một cửa sổ con kiểu CEdit. Khi thực hiện double trên một từ bất kỳ trong màn hình view rtf thì kích hoạt cửa sổ con và điền từ vừa chọn vào cửa sổ con này. Khi người dùng click vào view, cửa sổ con tự động biến mất.
9. Quan sát chương trình thi trắc nghiệm TOEFL. Viết ứng dụng như bài tập số 8, Khi thực hiện double trên một dòng bất kỳ trong màn hình view rtf thì kích hoạt cửa sổ con và điền nội dung dòng được chọn vào cửa sổ con này, Khi click vào cửa sổ view, nội dung chỉnh sửa được cập nhật vào view.
10. Quan sát Yahoo Messenger!. Thực hiện ứng dụng giao diện rtf như sau:



Một số vấn đề trong windows

12.1 TẬP TIN INI:

Tập tin INI là tập tin văn bản chứa các nội dung phục vụ cho hoạt động của ứng dụng trong môi trường windows phiên bản 3.x và 9x. Các nội dung này được chia thành các thành phần (section) phân biệt theo chức năng hoặc theo nhóm ứng dụng con. Mỗi thành phần chứa các mục với tên gọi xác định và phân biệt với các mục khác trong cùng thành phần. Mỗi mục tương ứng với một giá trị duy nhất. Giá trị của mục được viết ngay sau tên mục, và được ngăn cách với tên mục bằng dấu "=".



Xét ví dụ là tập tin WIN.INI nói trên:

- windows : Tên thành phần (section).
- NullPort : Tên mục (entry).
- None : Giá trị của mục NullPort.

Việc truy xuất giá trị các mục trong tập tin INI được hỗ trợ bởi lớp đối tượng quản lý ứng dụng CWinApp thông qua các thuộc tính, hành vi sau:

- **const char* m_pszAppName** : Lưu chuỗi tên của ứng dụng. Giá trị thuộc tính này có thể được thay đổi được như sau:

```
free( (void*) m_pszAppName ); // Giải phóng vùng nhớ
m_pszAppName = _tcsdup( _T("Tên_mới_của_ứng_dụng") );
```

- **const char* m_pszProfileName** : Lưu đường dẫn và tên tập tin INI sử dụng bởi ứng dụng. Có thể thay đổi giá trị này để ấn định tập tin INI:

```
free((void*)m_pszProfileName); // Hủy bỏ vùng nhớ cấp phát
m_pszProfileName=_tcsdup(_T("ĐườngDẫn-TênTậpTin_INI"));
```

- **BOOL WriteProfileString (**
 LPCTSTR lpszSection, // Tên thành phần
 LPCTSTR lpszEntry, // Tên mục
 LPCTSTR lpszValue // Giá trị của mục (kiểu chuỗi)
); Lưu giá trị kiểu chuỗi của một mục trong thành phần xác định.
- **BOOL WriteProfileInt (**
 LPCTSTR lpszSection, // Tên thành phần
 LPCTSTR lpszEntry, // Tên mục
 int nValue // Giá trị của mục (số nguyên)
); Lưu giá trị kiểu số nguyên của một mục trong thành phần xác định.
- **BOOL WriteProfileBinary (**
 LPCTSTR lpszSection, // Tên thành phần
 LPCTSTR lpszEntry, // Tên mục
 LPBYTE pData, // Vùng đệm chứa giá trị mã
 UINT nBytes // Kích thước vùng đệm
); Lưu khối mã nhị phân của một mục trong thành phần xác định.
- **CString GetProfileString (**
 LPCTSTR lpszSection, // Tên thành phần
 LPCTSTR lpszEntry, // Tên mục. Nếu mục đọc không
 LPCTSTR lpszDefault = NULL // có thì sử dụng giá trị này.
); Trả về giá trị kiểu chuỗi của một mục trong thành phần tương ứng.
- **UINT GetProfileInt (**
 LPCTSTR lpszSection, // Tên thành phần
 LPCTSTR lpszEntry, // Tên mục. Nếu mục đọc không
 int nDefault // có thì sử dụng giá trị này.
); Trả về giá trị kiểu số nguyên của một mục trong thành phần tương ứng.
- **BOOL GetProfileBinary (**
 LPCTSTR lpszSection, // Tên thành phần
 LPCTSTR lpszEntry, // Tên mục
 LPBYTE *pData, // Địa chỉ con trỏ vùng đệm
 UINT *nBytes // Địa chỉ biến nhận kích thước.
); Đọc khối mã nhị phân của một mục vào vùng đệm.

Hành vi trả về giá trị TRUE nếu tác vụ đọc thành công.

Với: *pData* : Địa chỉ biến con trỏ quản lý vùng đệm nhận thông tin.

nBytes : Địa chỉ biến chứa kích thước thông tin đọc được.

☞ Ứng dụng cần giải phóng vùng đệm *pData* khi chấm dứt sử dụng.

☞ Đoạn chương trình sau thực hiện ghi xuống thành phần MY_TEST của tập tin INI của ứng dụng: MyName = Mr.Emp và MyVer = 11.

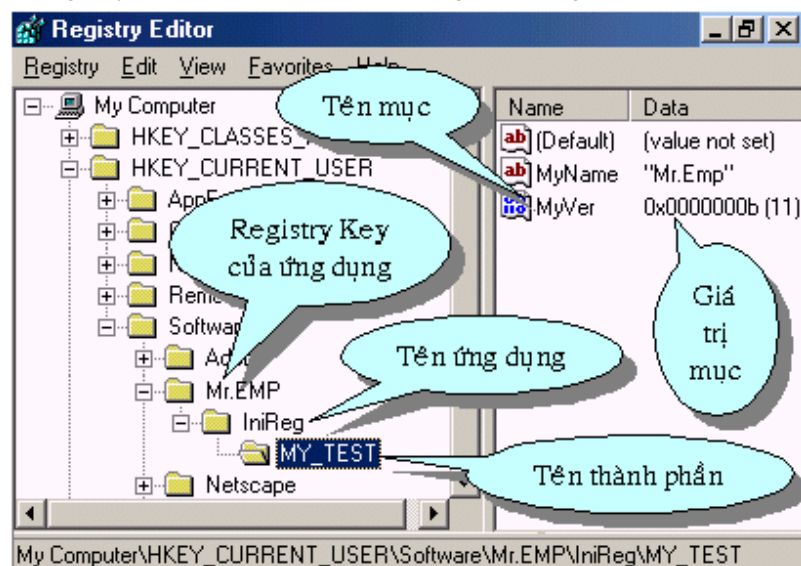
```
CWinApp* pApp = AfxGetApp(); // Đối tượng quản lý ứng dụng
pApp->WriteProfileString ("MY_TEST", "MyName", "Mr.Emp");
pApp->WriteProfileInt ("MY_TEST", "MyVer", 11);
```

☞ Đoạn chương trình sau thực hiện đọc từ thành phần MY_TEST của tập tin INI giá trị hai mục nói trên.

```
CWinApp* pApp = AfxGetApp(); // Đối tượng quản lý ứng dụng
CString myName = pApp->GetProfileString (
    "MY_TEST", "MyName", "Mr.Emp" );
UINT myVer = pApp->GetProfileInt ("MY_TEST", "MyVer", 11);
```

12.2 SYSTEM REGISTRY:

System Registry là cơ sở dữ liệu do windows quản lý, được sử dụng để lưu trữ các nội dung phục vụ cho hoạt động của hệ thống và các ứng dụng. System registry có cấu trúc như sau (chương trình RegEdit.exe).



(System registry ở một máy sử dụng phiên bản Windows-Me)

- Mỗi mục trong cấu trúc cây (tree) gọi là khóa (key).
- Khóa lá (không có con) là thành phần chứa các mục.
- Mỗi mục có một tên để nhận biết và có một giá trị xác định.

☞ Hành vi SetRegistryKey của lớp đối tượng CWinApp cho phép định hướng việc đọc/ghi giá trị các mục lên system registry thay vì sử dụng tập tin INI như (12.1). Hành vi loại protected này có cú pháp như sau:

```
void SetRegistryKey ( UINT lpszRegistryKey );
```

lpszRegistryKey : Thông thường là chuỗi chứa tên hãng phần mềm; ví dụ Netscape. Giá trị này trở thành khóa con của khóa Software thuộc khóa gốc HKEY_CURRENT_USER trong system registry.

☞ Khi ứng dụng thực hiện đọc/ghi giá trị mục, tên của ứng dụng (lưu trong *m_pszAppName* của đối tượng ứng dụng) trở thành khóa con của khóa xác định bởi *lpszRegistryKey*, và các thành phần chứa các mục trở thành khóa con của khóa *m_pszAppName*. Một thứ tự được thiết lập như sau:

```
HKEY_CURRENT_USER\Software\<Tên_hãng_phần_mềm>\
<Tên_ứng_dụng>\<Tên_thành_phần>\<Các_mục>.
```

☞ Thực hiện ứng dụng **IniReg**. Ứng dụng đăng ký sử dụng system registry với khóa "Mr.Emp", đồng thời tiến hành các tác vụ đọc / ghi hai giá trị như ví dụ mục (12.1).

Sau đây là các bước thực hiện dự án của ứng dụng:

- Dùng MFC Wizard tạo dự án **IniReg** với giao diện chính là dialog.
- Hành vi InitInstance của lớp đối tượng quản lý ứng dụng thực hiện đặt lại tên cho ứng dụng và đăng ký sử dụng registry với khóa "Mr.Emp":

```
BOOL CIniRegApp::InitInstance()
{
    free((void*)m_pszAppName); // Giải phóng vùng nhớ
    m_pszAppName= _tcsdup( _T("IniReg") ); // Đặt tên ứng dụng
    SetRegistryKey( _T("Mr.EMP") ); // Đăng ký registry
    CIniRegDlg dlg;
    m_pMainWnd = &dlg;
    dlg.DoModal(); // Thực hiện giao diện
    return TRUE;
}
```

- Thực hiện các bổ sung sau cho lớp dialog giao diện **CIniRegDlg**:
 - Mở dialog resource, cài đặt các control sau:

- Hộp nhập giá trị mục *MyName* Edit IDC_WRITE_NAME
 - Hộp nhập giá trị mục *MyVer* Edit IDC_WRITE_VERSION
 - Hộp hiển thị mục *MyName* Static IDC_READ_NAME
 - Hộp hiển thị mục *MyVer* Static IDC_READ_VERSION
 - Nút chọn thực hiện ghi Button IDC_WRITE
 - Nút chọn thực hiện đọc Button IDC_READ
- Hành vi OnWrite ứng với nút IDC_WRITE lưu giá trị các mục:

```
void CIniRegDlg::OnWrite()
{
    CWinApp* pApp = AfxGetApp();
    CString myName;
    UINT myVer;

    GetDlgItemText(IDC_WRITE_NAME, myName);
    myVer = GetDlgItemInt(IDC_WRITE_VERSION);

    pApp->WriteProfileString("MY_TEST", "MyName", myName);
    pApp->WriteProfileInt("MY_TEST", "MyVer", myVer);
}
```

- Hành vi OnRead ứng với nút IDC_READ đọc giá trị các mục:

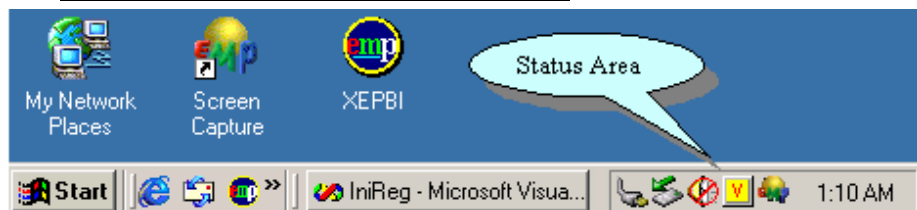
```
void CIniRegDlg::OnRead()
{
    CWinApp* pApp = AfxGetApp();
    CString myName;
    UINT myVer;

    myName = pApp->GetProfileString("MY_TEST",
                                    "MyName", "NoName");
    myVer = pApp->GetProfileInt("MY_TEST", "MyVer", 0);

    SetDlgItemText(IDC_READ_NAME, myName);
    SetDlgItemInt(IDC_READ_VERSION, myVer);
}
```

- Biên dịch và chạy ứng dụng.

12.3 VÙNG STATUS AREA TRÊN TASKBAR:



Taskbar là thanh công cụ đặc biệt của ứng dụng desktop trong windows. Nội dung của taskbar được chia thành 4 thành phần chính:

- **Start menu:** Mục kích hoạt hệ thống menu các ứng dụng.
- **Quick Launch Bar:** Danh mục các ứng dụng thường dùng.
- **Taskbar Buttons:** Danh sách các ứng dụng đang chạy.
- **Status Area:** Chứa icon giao diện của các ứng dụng đang chạy. Ứng dụng có thể nhận được tín hiệu nhập của người dùng khi họ dùng phím hoặc chuột tác động lên icon của ứng dụng trên status area.

Việc cài đặt hoặc hủy bỏ icon giao diện của ứng dụng trên status area được thực hiện thông qua hàm sau:

```
BOOL Shell_NotifyIcon (
    DWORD dwMessage,           // Tác vụ thực hiện
    PNOTIFYICONDATA lpdata     // Cấu trúc chứa thông số liên quan
);
```

Trả về giá trị TRUE nếu tác vụ thực hiện thành công.

dwMessage: Ấn định tác vụ thực hiện.

- NIM_ADD : Tạo icon giao diện của ứng dụng trên status area.
- NIM_MODIFY : Thay đổi thông số liên quan icon giao diện.
- NIM_DELETE : Xóa icon giao diện của ứng dụng trên status area.

lpdata: Địa chỉ cấu trúc NOTIFYICONDATA chứa các thông số.

Các trường trong cấu trúc NOTIFYICONDATA có ý nghĩa như sau:

```
typedef struct _NOTIFYICONDATA {
    DWORD cbSize;           // Kích thước cấu trúc
    HWND hWnd;              // Handle của cửa sổ xử lý message từ icon
    UINT uID;               // Số hiệu của icon trên status area
    UINT uFlags;             // Qui định các thông số có ý nghĩa
    UINT uCallbackMessage;  // Số hiệu message của icon gửi cửa sổ.
    HICON hIcon;            // Handle của icon được sử dụng
    TCHAR szTip[64];        // Nội dung chú thích của icon giao diện.
} NOTIFYICONDATA, *PNOTIFYICONDATA;
```

uFlags: Qui định trường thông số trong cấu trúc có ý nghĩa sử dụng.

NIF_MESSAGE : Trường *uCallbackMessage* được sử dụng.

NIF_ICON : Trường *hIcon* được sử dụng.

NIF_TIP : Trường *szTip* được sử dụng.

uCallbackMessage: Số hiệu message sẽ gửi trả về từ icon giao diện.

Khi người dùng tác động lên icon giao diện của ứng dụng, hệ thống gửi một message đến cửa sổ xử lý liên quan icon với nội dung như sau:

- *message* : Số hiệu message của icon (trong *uCallbackMessage*).
- *wParam* : Số hiệu của icon.
- *lParam* : Chứa các trạng thái của chuột hoặc phím mà người dùng đã sử dụng để tác động lên icon giao diện.

Hành vi WindowProc của cửa sổ xử lý liên quan sẽ tùy nghi xử lý:

```
LRESULT CTaskbarIconDlg::WindowProc ( UINT message,
                                     WPARAM wParam, LPARAM lParam )
{
    if ( message == Icon_uCallbackMessage ) {
        switch ( lParam ) {
            ...           // Xử lý biến cố phím / chuột trong lParam
        }
        return 0 ;
    }
    return CDialog::WindowProc(message, wParam, lParam);
}
```

Giả sử cần thực hiện ứng dụng như sau:



- **Set Icon:** Cài icon của ứng dụng lên status area.
- **Remove Icon:** Xóa icon của ứng dụng khỏi status area.
- Hộp thông báo cho biết các biến cố nhập tác động lên icon.

Các bước thực hiện dự án của ứng dụng:

- Dùng MFC Wizard tạo ứng dụng **TaskbarIcon** có giao diện là dialog.
- Thực hiện các bổ sung sau đây cho lớp dialog CTaskbarIconDlg:
 - Mở dialog resource, cài đặt các control sau:
 - *Hộp thông báo trạng thái icon* Static IDC_ICON_INFO
 - *Nút thực hiện đặt icon* Button IDC_ICON_SET
 - *Nút thực hiện xóa icon* Button IDC_ICON_REMOVE
 - Khai báo thuộc tính protected **m_isIconShow** kiểu luận lý, để ghi nhận tình trạng đặt icon; **m_isIconShow** = FALSE : icon chưa được đặt, **m_isIconShow** = TRUE : icon đã được đặt.
 - Hành vi OnInitDialog chuẩn bị các thông số:

```
BOOL CTaskbarIconDlg::OnInitDialog()
{
    CDialog::OnInitDialog();
    m_isIconShow = FALSE;    // Bắt đầu, icon chưa được đặt.
    return TRUE;
}
```

- Hành vi OnIconSet ứng với nút chọn IDC_ICON_SET thực hiện cài icon giao diện lên status area:

```
void CTaskbarIconDlg::OnIconSet()
{
    if ( m_isIconShow ) return;
    NOTIFYICONDATA dt;
    memset(&dt, 0, sizeof( NOTIFYICONDATA ) );
    dt.cbSize = sizeof(NOTIFYICONDATA);
    dt.uID    = 100;
    dt.hIcon  = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
    dt.hWnd   = this->GetSafeHwnd();
    dt.uCallbackMessage = WM_USER + 10;
    dt.uFlags  = NIF_MESSAGE | NIF_ICON | NIF_TIP;
    strcpy(dt.szTip, "Mr.Emp, hello world !");

    if ( Shell_NotifyIcon( NIM_ADD, &dt ) )
        m_isIconShow = TRUE;    // Đặt icon thành công
}
```

- Hành vi OnIconRemove ứng với nút chọn IDC_ICON_REMOVE thực hiện xóa icon giao diện khỏi status area:


```

void CTaskbarIconDlg::OnIconRemove()
{
    if ( !m_isIconShow ) return;
    NOTIFYICONDATA dt;
    memset(&dt, 0, sizeof(NOTIFYICONDATA));
    dt.cbSize = sizeof(NOTIFYICONDATA);
    dt.uID = 100;
    dt.hWnd = this->GetSafeHwnd();
    if ( Shell_NotifyIcon( NIM_DELETE, &dt ) )
        m_isIconShow = FALSE; // Xóa icon thành công
}

```

- Hành vi WindowProc xử lý thông tin nhập tác động lên icon:

```

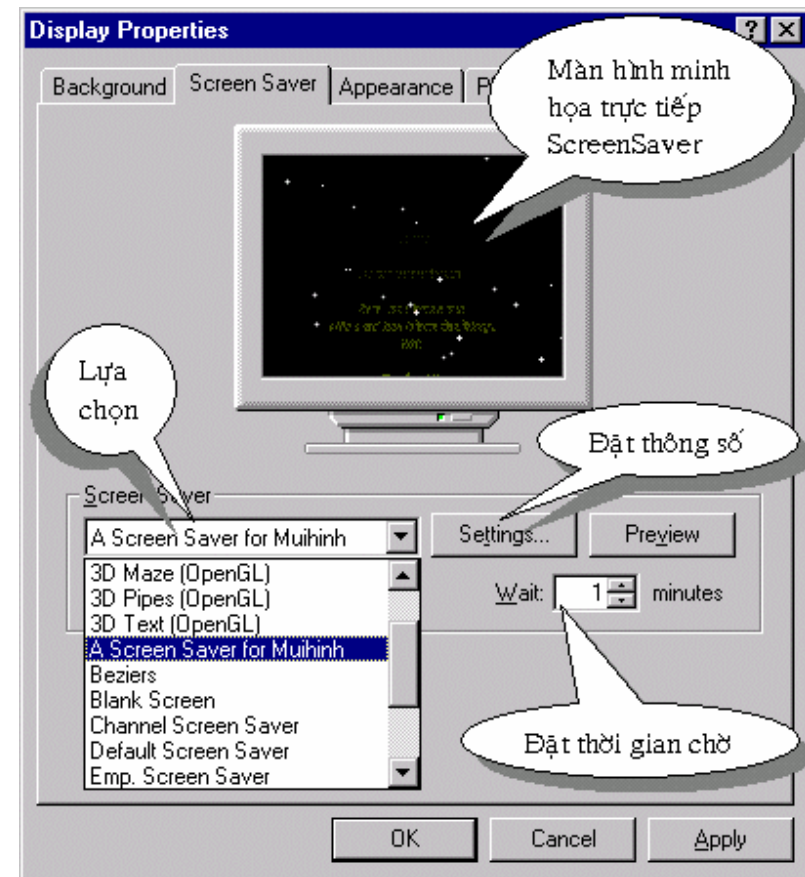
LRESULT CTaskbarIconDlg::WindowProc(UINT message,
    WPARAM wParam, LPARAM lParam)
{
    if ( message == WM_USER + 10 ) {
        // WM_USER + 10 : số hiệu message đăng ký cho icon.
        // Xử lý các thao tác bấm nút chuột (BT) của người dùng.
        switch (lParam) {
            case WM_MOUSEMOVE:
                SetDlgItemText(IDC_ICON_INFO, "Mouse moves!");
                break;
            case WM_LBUTTONDOWN:
                SetDlgItemText(IDC_ICON_INFO, "Left BT down !");
                break;
            case WM_LBUTTONUP:
                SetDlgItemText(IDC_ICON_INFO, "Left BT up !");
                break;
            case WM_RBUTTONDOWN:
                SetDlgItemText(IDC_ICON_INFO, "Right BT down !");
                break;
            case WM_RBUTTONUP:
                SetDlgItemText(IDC_ICON_INFO, "Right BT up !");
                break;
        }
        return 0;
    }
    return CDialog::WindowProc(message, wParam, lParam);
}

```

▪ Biên dịch và chạy ứng dụng.

12.4 ỨNG DỤNG SCREEN SAVER:

ScreenSaver là ứng dụng được lưu trong tập tin chương trình có phần mở rộng .SCR. Để sử dụng ứng dụng ScreenSaver, ta chép tập tin chương trình của ứng dụng vào thư mục hệ thống, sau đó sử dụng chức năng *Desktop* của *Control Panel* (Desktop Properties) cài đặt ứng dụng ScreenSaver cho hệ thống thông qua trang Screen Saver:



☞ Tên tập tin chương trình của ứng dụng ScreenSaver được dùng làm tên ScreenSaver trong danh sách lựa chọn các ScreenSaver. Nếu ta khai báo một hằng chuỗi với số hiệu là 1 trong StringTable resource của ứng dụng thì nội dung hằng chuỗi này (không được nhiều hơn 64 ký tự) được dùng làm tên ScreenSaver trong danh sách nói trên.

⌘ Khi khoảng thời gian mà người dùng ngừng tương tác với hệ thống vượt quá giá trị ấn định *Wait* thì hệ thống tự động thực hiện ứng dụng ScreenSaver.

12.4.1 **Đặc điểm:**

Khác với các ứng dụng thông thường khác, ứng dụng ScreenSaver phải có khả năng ứng xử khác nhau trong các tình huống khác nhau mà ứng dụng được thực hiện. Các tình huống thực hiện có thể xảy ra đối với một ứng dụng ScreenSaver là:

- (a) **Chạy minh họa trực tiếp ScreenSaver trong màn hình con của hộp hội thoại:** Khi người dùng click chọn ứng dụng ScreenSaver trong danh sách các ScreenSaver của hộp hội thoại Display Properties / Screen Saver.
- (b) **Thực hiện chức năng ấn định thông số của ScreenSaver:** Khi người dùng click chọn mục *Setting...* sau khi đã chọn ứng dụng ScreenSaver trong danh sách các ScreenSaver.
- (c) **Thực hiện chức năng đặt Password của ScreenSaver:** Khi người dùng click chọn mục *Password Setting...* sau khi đã chọn ứng dụng ScreenSaver trong danh sách các ScreenSaver. Chỉ sử dụng cho các phiên bản Win9x trở về trước.
- (d) **Chạy minh họa ScreenSaver trong màn hình thực:** Khi người dùng click chọn mục *Preview* sau khi đã chọn ứng dụng ScreenSaver trong danh sách các ScreenSaver. Chế độ chạy này tương tự chế độ chạy thực nhằm giúp người dùng hình dung được hoạt động thực của ScreenSaver.
- (e) **Chạy ScreenSaver:** Khi khoảng thời gian mà hệ thống ngừng tương tác với người dùng lớn hơn khoảng thời gian chờ cho phép. Đây là chế độ chạy thực của ScreenSaver.

12.4.2 **Tham số dòng lệnh (Command Line Parameters):**

Mỗi khi thực hiện ứng dụng ScreenSaver, hệ thống cung cấp thông tin về tình huống cụ thể mà ứng dụng được thực hiện thông qua nội dung tham số dòng lệnh gửi đến cho chương trình của ứng dụng. Ứng dụng ScreenSaver phải lấy thông tin này từ tham số dòng lệnh nhằm lựa chọn xử lý phù hợp.

Thuộc tính *m_lpCmdLine* của đối tượng quản lý tiểu trình chính trong ứng dụng chứa chuỗi tham số dòng lệnh truyền cho ứng dụng. Nội dung của thuộc tính này khác nhau tùy theo tình huống thực hiện ứng dụng khác nhau. Tương ứng các trường hợp (12.4.1), nội dung tham số dòng lệnh có thể là:

- (a) **Chạy minh họa trực tiếp ScreenSaver:** Vùng chạy mẫu là cửa sổ con trong hộp hội thoại Display Properties. Handle của cửa sổ này là một giá trị trong tham số dòng lệnh có nội dung như sau:

`"/p xxxx"`

Trong đó:

- `/p` (hoặc `/P`): Giá trị phản ánh tình huống thực hiện.
- `xxxx`: Các chữ số phản ánh giá trị handle của cửa sổ con.

Cách lấy giá trị handle này từ tham số dòng lệnh như sau:

```
HWND parent;           // Biến chứa handle
sscanf(m_lpCmdLine+3, "%d", &parent);
```

Ở chế độ này, ScreenSaver được thiết kế theo kiểu ứng dụng có màn hình giao diện chính là cửa sổ với nội dung hoạt động như chạy thực. Cửa sổ này được lồng vào vị trí của cửa sổ con nói trên.

- (b) **Thực hiện chức năng ấn định thông số:** Nội dung tham số dòng lệnh như sau:

`"/c xxxx"`

Trong đó:

- `/c` (hoặc `/C`): Giá trị phản ánh tình huống thực hiện.
- `xxxx`: Handle của hộp hội thoại (dialog).

Ở chế độ này, ScreenSaver được thiết kế theo kiểu ứng dụng có màn hình giao diện chính là dialog với các mục nhập cho phép điều chỉnh thông số liên quan đến cách thức hoạt động của ScreenSaver ở chế độ thực. Dialog giao diện là dialog khóa (modal dialog) và đối tượng cửa sổ cha của nó là hộp hội thoại nói trên.

- (c) **Thực hiện chức năng đặt Password:** Nội dung tham số dòng lệnh như sau:

`"/a xxxx"`

Trong đó:

- `/a` (hoặc `/A`): Giá trị nhận diện tình huống thực hiện.
- `xxxx`: Handle của cửa sổ hội thoại.

Ở chế độ này, ScreenSaver được thiết kế theo kiểu ứng dụng có màn hình giao diện chính là dialog. Dialog giao diện có thể là dialog dùng đặt password của hệ thống (thư viện MPR.DLL) hoặc dialog của người dùng.

- (d) **Chạy minh họa ScreenSaver:** Nội dung tham số dòng lệnh như sau:

`"/s"`

Trong đó:

- /s (hoặc /S): Giá trị nhận diện tình huống thực hiện.

Ở chế độ này, ScreenSaver thực hiện xử lý như chạy thực.

- (e) **Chạy ScreenSaver.** Ở chế độ này, ScreenSaver được thiết kế theo kiểu ứng dụng có màn hình giao diện chính là cửa sổ với nội dung hoạt động. Cửa sổ này thường có cùng kích thước và vị trí với cửa sổ desktop của hệ thống.

⦿ Phần lựa chọn trình hướng xử lý của ứng dụng ScreenSaver được cài đặt trong hành vi InitInstance của đối tượng quản lý ứng dụng.

12.4.3 Đặc điểm giao tác với người dùng:

Các ứng dụng ScreenSaver có đặc điểm chung là chấm dứt hoạt động khi nhận được tín hiệu nhập của người dùng (gõ phím, click hay di chuyển chuột). Do đó, chương trình ScreenSaver phải xử lý các message liên quan việc nhập liệu:

- Các message do tác động lên bàn phím:

WM_KEYDOWN, WM_KEYUP

- Các message do tác động lên chuột:

WM_MOUSEMOVE,

WM_LBUTTONDOWN, WM_LBUTTONUP,

WM_RBUTTONDOWN, WM_RBUTTONUP,

WM_MBUTTONDOWN, WM_MBUTTONUP

☞ Thông thường, khi nhận được một trong các message này thì ứng dụng tự động kết thúc.

Trong lúc ScreenSaver hoạt động, cần ngăn cấm người dùng gõ phím Ctrl+Alt+Del để 'qua mặt' ứng dụng. Việc ngăn cấm này thực hiện như sau:

```
UINT oldval;           // Dùng bảo lưu trạng thái
// Bắt đầu, cấm phím:
SystemParametersInfo ( SPI_SETSCREENSAVERRUNNING,
                        1, &oldval, 0 );
... // Phần thực hiện của ScreenSaver

// Cuối cùng, thôi cấm phím:
SystemParametersInfo ( SPI_SETSCREENSAVERRUNNING,
                        0, &oldval, 0 );
```

12.4.4 Thực hiện ứng dụng ScreenSaver đơn giản:

Giả sử cần thực hiện ứng dụng ScreenSaver với nội dung hoạt động là hiển thị câu chào "Hello !". Các bước thực hiện như sau:

- Tạo dự án **ScreenSaver** tương tự dự án VD01.
- Bổ sung lớp CEmpScreenSaverWnd kế thừa từ lớp CWnd cho dự án. Thực hiện cài đặt cho lớp CEmpScreenSaverWnd như sau:
 - Hành vi OnPaint hiển thị thông báo "Hello !" ở chính giữa vùng client của cửa sổ.

```
void CEmpScreenSaverWnd::OnPaint()
{
    CPaintDC dc(this);           // device context để vẽ

    // Do ScreenSaver's works here :
    LOGFONT lf;
    CFont      font, *oldFont;

    memset(&lf, 0, sizeof(LOGFONT));
    strcpy(lf.lfFaceName, "Arial");
    lf.lfHeight = 50; lf.lfWidth = 14;
    font.CreateFontIndirect(&lf);      // Tạo font chữ để sử dụng
    oldFont = dc.SelectObject(&font);  // và lưu font chữ cũ.

    RECT rect;
    dc.GetClipBox(&rect);             // Xóa nền DC
    dc.FillRect(&rect, &CBrush(RGB(0, 128, 128)));

    dc.SetBkMode(TRANSPARENT);
    rect.top += 3; rect.left += 3;
    dc.SetTextColor(RGB(0, 0, 128));
    dc.DrawText("Hello !", 7, &rect,
                DT_CENTER | DT_VCENTER | DT_SINGLELINE);
    rect.top -= 3; rect.left -= 3;
    rect.bottom -= 3; rect.right -= 3;
    dc.SetTextColor(RGB(255, 255, 0));
    dc.DrawText("Hello !", 7, &rect,
                DT_CENTER | DT_VCENTER | DT_SINGLELINE);
    dc.SelectObject(oldFont);         // Khôi phục lại font chữ cũ
}
```

- Các hành vi xử lý message của chuột và bàn phím: OnKeyDown, OnLButtonDown, OnMouseMove, OnRButtonDown thực hiện gửi message đóng cửa sổ:

```
PostMessage( WM_CLOSE, 0, 0 );
```

- Tạo mới dialog resource và lớp CEmpScreenSaverDlg kế thừa từ CDialog sử dụng dialog resource này. Dialog CEmpScreenSaverDlg cho phép người dùng chỉnh sửa các thông số liên quan đến hoạt động của ứng dụng ScreenSaver. Có thể lưu các thông số ấn định bởi người dùng vào system registry (12.2).
- Bổ sung và chỉnh sửa lớp quản lý ứng dụng CEmpApp như sau:

- Bổ sung các thuộc tính protected quản lý hoạt động ScreenSaver:

```
BOOL m_bFullDemo;  
UINT Ctrl_Alt_Del_State;
```

- Hành vi InitInstance lựa chọn xử lý để thực hiện một cách phù hợp với các tình huống hoạt động của ứng dụng:

```
BOOL CEmpApp::InitInstance()  
{  
    m_bFullDemo = FALSE;           // Lưu chế độ minh họa  
    HWND parent;                   // Handle cửa sổ cha  
    RECT rect;                     // Vị trí cửa sổ ứng dụng  
    DWORD exstyle = 0, style;       // Dạng cửa sổ ứng dụng  
    sscanf(m_lpCmdLine + 3, "%d", &parent);  
    switch (m_lpCmdLine[1]) {  
    case 'c':  
    case 'C':  
        // Chức năng ấn định thông số  
        CEmpScreenSaverDlg* main;  
        CWnd* pr;  
        pr = new CWnd();  
        pr->Attach(parent);  
        main = new CEmpScreenSaverDlg(pr);  
        m_pMainWnd = main;  
        main->DoModal();  
        pr->Detach();  
        delete main;  
        delete pr;  
        return FALSE;  
    }
```

```
case 'a':  
case 'A':  
    // Đặt Password (không dùng cho WinNT-Win2000)  
    // Sử dụng hộp Password của hệ thống cho tương thích.  
    typedef VOID (WINAPI *PWDCHANGEPASSWORD) (  
        LPCSTR lpcRegkeyname,  
        HWND hwnd,UINT uiReserved1,  
        UINT uiReserved2 );  
  
    PWDCHANGEPASSWORD PwdChangePassword;  
    HINSTANCE hmpr;  
    hmpr = ::LoadLibrary("MPR.DLL");  
    if (hmpr == NULL) return FALSE;  
    PwdChangePassword = (PWDCHANGEPASSWORD)  
        ::GetProcAddress(hmpr, "PwdChangePasswordA");  
    if (PwdChangePassword != NULL)  
        PwdChangePassword("SCRSAVE",parent,0,0);  
    FreeLibrary(hmpr);  
    return FALSE;  
  
case 'p':  
case 'P':  
    // Xem minh họa trong màn hình con  
    style = WS_CHILD | WS_VISIBLE | WS_DISABLED ;  
    break;  
default:  
    // Chế độ chạy thực của ScreenSaver  
    m_bFullDemo = TRUE;  
    parent = GetDesktopWindow();  
    exstyle= WS_EX_TOOLWINDOW | WS_EX_TOPMOST;  
    style = WS_POPUP | WS_VISIBLE;  
}  
  
// Xác định chế độ chạy cụ thể  
CEmpScreenSaverWnd* main;  
main = new CEmpScreenSaverWnd;  
m_pMainWnd = main;  
  
POINT p1, p2;                       // Góc trái trên, phải dưới  
GetWindowRect (parent, &rect); // Xác định vị trí cửa sổ cha  
p1.x = rect.left; p1.y = rect.top;  
p2.x = rect.right; p2.y = rect.bottom;
```



```

if (!m_bFullDemo) {
    // Chạy trong cửa sổ con :
    // Xác định tọa độ cửa sổ con trong hộp Display Properties
    ScreenToClient ( parent, &p1 );
    ScreenToClient ( parent, &p2 );
}
// Cửa sổ giao diện chính có tọa độ và kích thước thích hợp
main->CreateEx(exstyle, _T("STATIC"),"Emp.ScreenSaver",
    style, p1.x, p1.y, p2.x, p2.y, parent, NULL);
// Dấu con chuột và khóa phím nếu chạy ScreenSaver thực
if (m_bFullDemo) {
    ShowCursor(FALSE);
    main->SetCapture();
    SystemParametersInfo (
        SPI_SETSCREENSAVERRUNNING,
        1,&Ctrl_Alt_Del_State, 0
    );
}
main->UpdateWindow();
return TRUE;
}

```

- Hành vi ExitInstance thực hiện gỡ bỏ các cài đặt:

```

int CEmpApp::ExitInstance()
{
    if (m_bFullDemo) {
        ReleaseCapture();
        ShowCursor(TRUE);
        SystemParametersInfo (
            SPI_SETSCREENSAVERRUNNING,
            0, &Ctrl_Alt_Del_State, 0
        );
    }
    return CWinApp::ExitInstance();
}

```

- Tạo hằng chuỗi tùy ý có số hiệu là 1 trong StringTable resource.
- Biên dịch ứng dụng. Chép tập tin chương trình (.exe) vào thư mục hệ thống và đổi tên tập tin với phần mở rộng là **.SCR**.
- Sử dụng Control Panel / Display / ScreenSaver kiểm tra kết quả.

12.5 ỨNG DỤNG SỬ DỤNG NHIỀU TIỂU TRÌNH:

Việc thiết lập các tiểu trình con hỗ trợ cho tiểu trình chính trong chương trình của ứng dụng cho phép ứng dụng đồng thời đáp ứng nhiều yêu cầu của người dùng. Các tiểu trình hỗ trợ có thể thực hiện các xử lý bên trong (tiểu trình xử lý nội – worker thread) hoặc trực tiếp nhận và thực hiện các yêu cầu của người dùng (tiểu trình giao diện– user interface thread).

12.5.1 Tiểu trình xử lý nội:

Tiểu trình xử lý nội đảm nhận các xử lý tính toán bên trong, không trực tiếp tương tác với người dùng. Việc thiết lập tiểu trình xử lý nội trong chương trình được thực hiện thông qua các nội dung sau đây:

- Xây dựng hàm đảm nhận việc điều khiển toàn bộ hoạt động xử lý của tiểu trình (Thread Procedure). Hàm này có khai báo như sau:

```

UINT MyThreadProc ( LPVOID pParam );

```

pParam : Tham số duy nhất mà hàm điều khiển nhận được khi tiểu trình được kích hoạt.

☞ Khi kết thúc xử lý, hàm phải trả về một giá trị số nguyên phản ánh tình trạng kết thúc của hàm. Thông thường, giá trị 0 trả về cho một kết thúc thành công, các giá trị khác 0 là các qui ước về hiện tượng lỗi.

- Thực hiện khởi động tiểu trình xử lý nội thông qua hàm sau:

```

CWinThread* AfxBeginThread (
    AFX_THREADPROC pfnThreadProc,           // Hàm điều khiển
    LPVOID pParam,                           // Tham số của hàm.
    int nPriority = THREAD_PRIORITY_NORMAL,
    UINT nStackSize = 0,
    DWORD dwCreateFlags = 0,
    LPSECURITY_ATTRIBUTES lpSecurityAttrs = NULL
);

```

☞ Sau đây là bố cục thực hiện toàn bộ công việc trên:

```

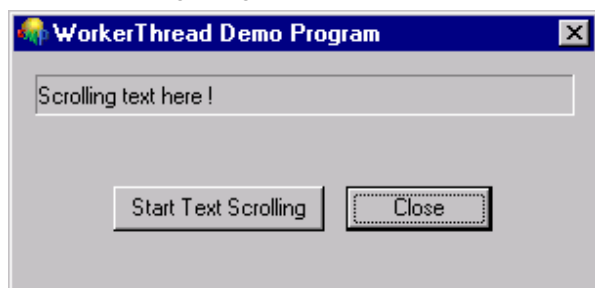
UINT MyThreadProc( LPVOID pParam ) {
    ... // Khai thác nội dung của pParam
    ... // Thực hiện các xử lý cần thiết.

    return ( có_lỗi ) ? 1 : 0;
}

... // Chuẩn bị khởi động tiểu trình
pInfo = new MyInfo ; // Chỉ đến vùng chứa các giá trị thông số
AfxBeginThread( MyThreadProc, pInfo );

```

📁 Giả sử cần thực hiện ứng dụng sau:



- Tiểu trình xử lý nội cài đặt bởi một hàm có nhiệm vụ thực hiện chạy dòng chữ trong hộp thông báo cho đến khi có tín hiệu ngừng.
- Tiểu trình giao diện quản lý dialog nhận yêu cầu người dùng:
 - Mục **Start Text Scrolling** khởi động tiểu trình thực hiện chạy chữ. Mục chọn này sau đó đổi thành **Stop Text Scrolling** để điều khiển ngừng tiểu trình nói trên.
 - Mục **Close** chấm dứt ứng dụng.

Dự án của ứng dụng được thực hiện như sau:

- Tạo dự án **WorkerThread** với giao diện chính là dialog.
- Thực hiện các cài đặt sau cho lớp CWorkerThreadDlg làm giao diện:
 - Mở dialog resource, cài đặt các control sau:
 - Hộp chứa dòng chữ chạy Static IDC_INFO
 - Nút lệnh cho phép chữ chạy / ngừng Button IDOK
 - Thuộc tính public **m_isTextScrolled** kiểu BOOL ghi nhận thông tin về hoạt động chạy chữ.
 - Hành vi OnInitDialog khởi động các thông số:

```
BOOL CWorkerThreadDlg::OnInitDialog()
{
    CDialog::OnInitDialog();
    SetIcon(m_hIcon, TRUE); // Set big icon
    SetIcon(m_hIcon, FALSE); // Set small icon

    m_isTextScrolled = FALSE; // Chưa thực hiện chạy chữ
    return TRUE;
}
```

- Hành vi OnOK ứng với nút lệnh IDOK thực hiện khởi động hoặc thông báo ngừng tiểu trình xử lý chạy chữ:

```
void CWorkerThreadDlg::OnOK()
{
    if (m_isTextScrolled) {
        SetDlgItemText( IDOK, "Start Text Scrolling" );
        // Đặt giá trị thông báo tiểu trình chạy chữ chấm dứt
        m_isTextScrolled = FALSE;
    }
    else {
        SetDlgItemText(IDOK, "Stop Text Scrolling");
        // Đặt giá trị cho phép cho tiểu trình chạy chữ thực hiện
        m_isTextScrolled = TRUE;
        // Khởi động tiểu trình với tham số là dialog giao diện
        AfxBeginThread( TextScrolling, this );
    }
}
```

- TextScrolling là hàm xử lý của tiểu trình chạy chữ. Hàm nhận tham số void* là con trỏ chỉ đến đối tượng dialog giao diện. Có thể khai báo hàm trong phần cài đặt của lớp CWorkerThreadDlg để tiện sử dụng.

```
UINT TextScrolling(void* pParam) {
    // Con trỏ pParam thực chất là con trỏ đối tượng dialog giao diện
    CWorkerThreadDlg* pDlg = (CWorkerThreadDlg*)pParam;
    static CString info = " Welcome to multi-thread programming";
    while (pDlg->m_isTextScrolled) {
        // Giá trị thông báo cho phép tiểu trình tiếp tục thực hiện:
        info = info.Mid(1) + info.Left(1);
        pDlg->SetDlgItemText(IDC_INFO, info);
        Sleep(100); // Tạm nghỉ
    }
    return 0; // Kết thúc tiểu trình xử lý nội
}
```

- Biên dịch và chạy thử ứng dụng.

12.5.2 Tiểu trình giao diện:

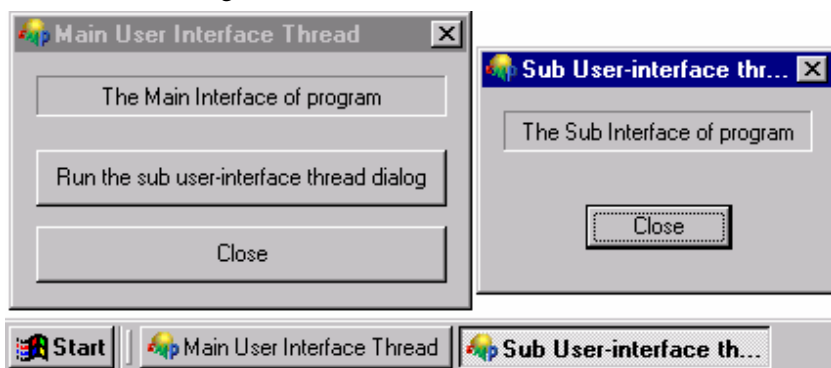
Tiểu trình giao diện có khả năng trực tiếp nhận và xử lý yêu cầu của người dùng một cách độc lập với tiểu trình chính (cũng là tiểu trình giao diện) của ứng dụng. Việc thiết lập tiểu trình giao diện trong chương trình được thực hiện thông qua các nội dung sau đây:

- Chuẩn bị giao diện (cửa sổ hoặc dialog) của tiểu trình giao diện.
- Xây dựng lớp kế thừa từ CWinThread để quản lý tiểu trình giao diện. Sử dụng giao diện trên cho lớp thông qua hành vi InitInstance của lớp.
- Thực hiện khởi động tiểu trình giao diện thông qua hàm sau:

```
CWinThread* AfxBeginThread (
    CRuntimeClass* pThreadClass,
    int nPriority= THREAD_PRIORITY_NORMAL,
    UINT nStackSize= 0,
    DWORD dwCreateFlags= 0,
    LPSECURITY_ATTRIBUTES lpSecurityAttrs= NULL
);
```

pThreadClass: Con trỏ đến cấu trúc quản lý thông tin thi hành của lớp đối tượng quản lý tiểu trình giao diện được kích hoạt. Xem (11.4).

Giả sử có yêu cầu thực hiện ứng dụng với hai giao diện hoạt động đồng hành; giao diện **Sub Interface of program** được kích hoạt khi người dùng click chọn mục **Run the sub user-interface thread dialog** trên dialog giao diện của tiểu trình giao diện chính.



Các bước thực hiện như sau:

- Tạo dự án **UserIntThread** với giao diện chính là dialog.
- Thiết kế dialog resource cho giao diện con. Trên dialog resource này, cài nút thoát với số hiệu IDCANCEL.
- Bổ sung lớp CSubUserDlg kế thừa từ CDialog sử dụng resource trên.
- Bổ sung lớp đối tượng CSubUserThread kế thừa từ CWinThread cho phép quản lý các tiểu trình giao diện con. Đối tượng CSubUserThread nhận đối tượng CSubUserDlg làm cửa sổ giao diện chính thông qua hành vi InitInstance của nó.

```
BOOL CSubUserThread::InitInstance()
{
    CSubUserDlg dlg;
    m_pMainWnd = &dlg;
    dlg.DoModal();
    return TRUE;
}
```

- Thực hiện các cài đặt sau cho lớp dialog CUserIntThreadDlg:
 - Mở dialog resource, cài đặt các control sau:
 - Nút lệnh kích hoạt tiểu trình giao diện con Button IDOK
 - Nút lệnh kết thúc ứng dụng Button IDCANCEL
 - Hành vi OnOK cho nút chọn IDOK kích hoạt tiểu trình giao diện:

```
void CUserIntThreadDlg::OnOK()
{
    AfxBeginThread( RUNTIME_CLASS(CSubUserThread) );
}
```

- Biên dịch và chạy thử ứng dụng.

12.5.3 Các hàm hỗ trợ:

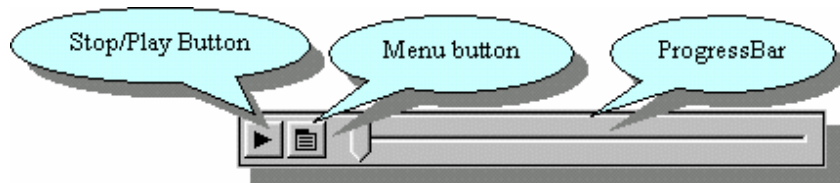
- void **AfxEndThread** (
 - UINT *nExitCode* // Giá trị kết thúc tiểu trình
); Chấm dứt hoạt động của tiểu trình. Hàm chỉ được sử dụng trong phần cài đặt xử lý của tiểu trình.
- BOOL **GetExitCodeThread** (
 - HANDLE *hThread*, // Handle của tiểu trình
 - LPDWORD *lpExitCode* // Con trỏ đến biến chứa kết quả.
); Lấy giá trị kết thúc của một tiểu trình. Trả về giá trị TRUE nếu tác vụ thực hiện thành công. *hThread* của một tiểu trình có thể lấy từ thuộc tính **m_hThread** của đối tượng CWinThread quản lý tiểu trình.

12.6 LẬP TRÌNH MULTIMEDIA VỚI MCI:

MCI (Media Control Interface) cung cấp bộ lệnh cơ bản có tính thích nghi cao với nhiều thiết bị media trong việc thực hiện các nội dung multimedia.

Mỗi thiết bị media sử dụng cho việc thực hiện nội dung multimedia được MCI quản lý thông qua cửa sổ giao diện MCI thuộc lớp MCIWnd. Đây là lớp cửa sổ do MCI đăng ký trước nhằm phục vụ cho mục đích này.

Cửa sổ giao diện MCI như sau:



Sau đây là các hàm MCI sử dụng phổ biến (Vfw.h):

- **HWND MCIWndCreate** (
 - HWND *hwndParent*, // Handle cửa sổ cha của cửa sổ MCI
 - HINSTANCE *hInstance*, // Handle của ứng dụng
 - DWORD *dwStyle*, // Dạng cửa sổ giao diện MCI
 - LPSTR *szFile* // Đường dẫn, tên tập tin media.
); Mở thiết bị media và sử dụng nội dung multimedia trong tập tin tương ứng. Hàm trả về con trỏ cửa sổ MCI quản lý thiết bị được mở. Thông số *dwStyle* của cửa sổ MCI được kết hợp từ các giá trị sau:
 - MCIWNDF_SHOWALL : Hiển thị các mục của cửa sổ.
 - MCIWNDF_NOMENU : Không hiển thị nút chọn menu.
 - MCIWNDF_NOPLAYBAR : Không hiển thị progressbar.
- **LONG MCIWndOpen** (
 - HWND *hwnd*, // Handle cửa sổ MCI
 - LPSTR *szFile* // Đường dẫn, tên tập tin media.
 - DWORD *dwStyle* = 0
); Mở nội dung multimedia mới cho thiết bị media quản lý bởi *hwnd*.
- **LONG MCIWndClose**(HWND *hwnd*); Đóng nội dung multimedia.
- **LONG MCIWndPlay**(HWND *hwnd*); Thực hiện.
- **LONG MCIWndStop**(HWND *hwnd*); Ngừng thực hiện.
- **LONG MCIWndPause**(HWND *hwnd*); Tạm ngừng.
- **LONG MCIWndResume**(HWND *hwnd*); Tiếp tục.
- **VOID MCIWndDestroy**(HWND *hwnd*); Đóng thiết bị media.

☞ Giả sử thiết kế ứng dụng cho phép chọn tập tin multimedia; mục **Play** thực hiện nội dung tập tin, mục **Stop** ngừng thực hiện.

Các bước tiến hành như sau:

- Dùng MFC Wizard tạo dự án **MCI** với giao diện chính là dialog.
- Khai báo sử dụng thư viện MCI trong tập tin STDAFX.H:

```
#include <Vfw.h>
#pragma comment (lib, "Vfw32.lib") // MCI library
```

- Trong lớp dialog giao diện chính: **CMCIDlg**

- Mở dialog resource, cài các control sau:
 - Hộp nhập đường dẫn, tên tập tin Edit IDC_FILE
 - Nút lệnh thực hiện **Play** Button IDC_PLAY
 - Nút lệnh thực hiện **Stop** Button IDC_STOP
- Bổ sung thuộc tính protected **m_mciWnd** kiểu HWND dùng quản lý thiết bị media được sử dụng.
- Hành OnInitDialog khởi động thiết bị media:

```
BOOL CMCIDlg::OnInitDialog()
{
    CDialog::OnInitDialog();
    SetDlgItemText(IDC_FILE, "HappyNewYear.rmi");
    m_mciWnd = MCIWndCreate ( this->GetSafeHwnd(),
                              AfxGetInstanceHandle(),
                              /*invisible*/ ~WS_VISIBLE, NULL );
    return TRUE;
}
```

- Hành vi OnPlay ứng với nút IDC_PLAY thực hiện nội dung media:

```
void CMCIDlg::OnPlay()
{
    char fName[1024]; // Lấy tên tập tin multimedia
    GetDlgItemText(IDC_FILE, fName, 1023);
    if ( MCIWndOpen( m_mciWnd, fName, 0 ) == 0 ) {
        MCIWndPlay( m_mciWnd );
    }
}
```

- Hành vi OnStop ngừng thực hiện:

```
void CMCIDlg::OnStop()
{
    MCIWndStop( m_mciWnd );
}
```

- Biên dịch và chạy thử ứng dụng.

12.7 ẮN ĐỊNH MỘT SỐ TÍNH NĂNG CỦA WINDOWS:

Môi trường windows cho phép người dùng ấn định một số tính năng tiện ích như chế độ tự động ngắt nguồn, tắt máy,... Các tính năng này có thể được cài đặt thông qua chương trình tiện ích hoặc lập trình tự động với hàm sau:

BOOL SystemParametersInfo (

UINT *uiAction*, // Số hiệu của chức năng cần thực hiện
UINT *uiParam*, // Ấn định tùy thuộc *uiAction*
PVOID *pvParam*, // Con trỏ vùng đệm, tùy thuộc *uiAction*
UINT *fWinIni* // Đề nghị cập nhật user profile. = 0: không.

); Lấy hoặc đặt thông số qui định tính năng tương ứng của hệ thống.

Giá trị *uiAction* có thể là:

SPI_GETLOWPOWERTIMEOUT : Lấy thông số low power timeout
SPI_GETPOWEROFFTIMEOUT : Lấy thông số power timeout
SPI_GETSCREENSAVETIMEOUT : Lấy thông số S.saver timeout
SPI_SETLOWPOWERTIMEOUT : Đặt thông số low power timeout
SPI_SETPOWEROFFTIMEOUT : Đặt thông số power timeout
SPI_SETSCREENSAVETIMEOUT : Đặt thông số S.saver timeout.

▫ Khi *uiAction* là giá trị có ý nghĩa lấy thông số:

uiParam : Có giá trị bằng 0

pvParam : Chỉ đến vùng đệm nhận giá trị thông số hiện hành.

▫ Khi *uiAction* là giá trị có ý nghĩa đặt thông số:

uiParam : Giá trị thông số ấn định.

pvParam : = NULL

▫ Đoạn chương trình sau thực hiện cấm tính năng screen saver:

```
UINT oldStatus;  
// Lấy thông số qui định tính năng ScreenSaver, lưu vào oldStatus  
SystemParametersInfo( SPI_GETSCREENSAVETIMEOUT,  
0, &oldStatus, 0 );  
  
// Cấm tính năng ScreenSaver  
SystemParametersInfo( SPI_SETSCREENSAVETIMEOUT,  
0, NULL, 0 );  
  
... // Các xử lý trong điều kiện ScreenSaver bị cấm  
  
// Trả lại ấn định trước đó cho tính năng ScreenSaver : oldStatus  
SystemParametersInfo( SPI_SETSCREENSAVETIMEOUT,  
oldStatus, NULL, 0 );
```

12.8 BẮY (HOOK) MESSAGE (WINDOWS HOOK):

Hook là một khâu trong cơ chế xử lý message của windows mà ứng dụng có thể can thiệp để cài đặt thủ tục xử lý message (hook procedure) trước khi message đó đến được đối tượng xử lý message mà hệ thống điều phối.

12.8.1 Các kiểu hook (Hook Type):

Hook được phân chia thành các kiểu khác nhau tùy thuộc vào kiểu message được hook. Một số kiểu hook phổ biến như sau:

WH_KEYBOARD : Hook các message liên quan bàn phím.
WH_MOUSE : Hook các message liên quan con chuột.
WH_CBT : Hook các thao tác trên cửa sổ giao diện.
WH_CALLWNDPROCRET : Hook giá trị trả về từ hàm xử lý message WindowProc có kiểu CWPRETSTRUCT.
WH_CALLWNDPROC : Hook các message trước khi message được chuyển đến thủ tục WindowProc.

12.8.2 Danh sách hook (Hook Chain):

Danh sách hook là danh sách các con trỏ chỉ đến các thủ tục hook. Mỗi kiểu hook có một danh sách hook riêng. Khi một message phát sinh, nó được chuyển đến danh sách hook liên quan; từ thủ tục hook này đến đến thủ tục hook khác. Mỗi thủ tục hook có thể tùy nghi thực hiện các xử lý khác nhau: ghi nhận message, chỉnh sửa message, hoặc ngăn cấm message không để nó đến được thủ tục hook kế tiếp.

12.8.3 Thủ tục hook (Hook Procedure):

Thủ tục hook là chương trình con chuyên dụng cho việc hook loại message liên quan. Chương trình con xử lý hook có khai báo như sau:

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

Các tham số gửi cho chương trình con này khác nhau theo từng kiểu hook:

- WH_KEYBOARD – *xử lý message bàn phím - KeyboardProc*
code : - HC_ACTION: Message của phím được thực hiện.
- HC_NOREMOVE: Message của phím chưa được lấy khỏi message queue.
wParam: Mã phím liên quan.
lParam : - Các bit 0÷15 : Giá trị cho biết số lần gõ phím
- Các bit 16÷23 : Mã scan code của phím.
- Bit 29 : Bằng 1 nếu phím Alt được nhấn kèm.
- WH_MOUSE – *xử lý message từ con chuột - MouseProc*
code : - HC_ACTION: Message của chuột được thực hiện.
- HC_NOREMOVE: Message của chuột chưa được lấy khỏi message queue.
wParam : Số hiệu message của con chuột

lParam : Con trỏ đến cấu trúc MOUSEHOOKSTRUCT.

```
typedef struct tagMOUSEHOOKSTRUCT {
    POINT pt;           // Vị trí chỉ điểm của con chuột
    HWND hwnd;          // Handle cửa sổ liên quan.
    UINT wHitTestCode;   // Xem (4.2): WM_SETCURSOR
    ULONG_PTR dwExtraInfo; // Các thông tin bổ sung khác.
} MOUSEHOOKSTRUCT, *PMOUSEHOOKSTRUCT;
```

▪ WH_CBT – *Xử lý thao tác trên cửa sổ - CBTProc*:

Quan hệ giữa các thông số như sau:

Code	wParam	lParam
HCBT_ACTIVATE (Kích hoạt cửa sổ)	Handle của cửa sổ	Con trỏ cấu trúc chứa thông tin.
HCBT_CREATEWND (Tạo mới cửa sổ)	Handle của cửa sổ.	Con trỏ cấu trúc chứa thông tin.
HCBT_DESTROYWND (Hủy bỏ cửa sổ)	Handle của cửa sổ.	=NULL
HCBT_MINMAX (Phóng to hay Thu nhỏ cửa sổ)	Handle của cửa sổ.	Word thấp chứa thông số hiển thị cửa sổ (SW_XXX).
HCBT_MOVESIZE (Di chuyển hoặc thay đổi kích thước cửa sổ)	Handle của cửa sổ.	Con trỏ cấu trúc RECT quản lý tọa độ, kích thước mới.

12.8.4 Các dịch vụ liên quan hook:

▪ HHOOK **SetWindowsHookEx** (

int *idHook*, // Kiểu hook
HOOKPROC *lpfn*, // Địa chỉ thủ tục hook
HINSTANCE *hMod*, // Handle của đơn thể chứa hook
DWORD *dwThreadId* // Số hiệu tiểu trình sử dụng hook

); Cài đặt thủ tục hook vào danh sách hook tương ứng. Hàm trả về handle của thủ tục hook được cài đặt trước thủ tục bị chiếm quyền.

dwThreadId: Tiểu trình sử dụng hook; =0: tất cả các tiểu trình.

lpfn: Nếu thủ tục hook sử dụng cho tất cả các tiểu trình thì nên đặt nó trong một tập tin DLL.

hMod: Handle của ứng dụng hoặc DLL chứa thủ tục hook.

▪ LRESULT **CallNextHookEx** (

HHOOK *hbk*, // Handle của thủ tục hook chiếm quyền
int *nCode*, // Chuyển giao giá trị các tham số
WPARAM *wParam*, // mà thủ tục hook chiếm quyền
LPARAM *lParam* // nhận được từ hệ thống

); Thực hiện thủ tục hook bị chiếm quyền, giúp ổn định cho windows.

▪ BOOL **UnhookWindowsHookEx** (

HHOOK *hbk* // Handle của thủ tục hook hủy bỏ.

); Hủy bỏ thủ tục hook trong danh sách hook.

Nếu thủ tục hook được cài trong DLL thì cần sử dụng các hàm sau:

▪ HMODULE **LoadLibrary** (

LPCTSTR *lpFileName* // Đường dẫn, tên tập tin DLL

); Trả về giá trị handle của DLL.

▪ FARPROC **GetProcAddress** (

HMODULE *hModule*, // Handle của DLL chứa thủ tục
LPCSTR *lpProcName* // Tên thủ tục

); Trả về con trỏ của thủ tục tương ứng.

12.8.5 Ứng dụng hook messages của keyboard:

Trong phần này, ta thực hiện ứng dụng hook message của bàn phím. Xử lý hook của ứng dụng bật tiếng beep để thông báo có gõ phím và chuyển message nhận được cho thủ tục xử lý hook đã bị xử lý này chiếm quyền.

Các bước thực hiện như sau:

▪ Dùng MFC Wizard tạo ứng dụng **Hook** với giao diện chính là dialog.

▪ Thực hiện cài đặt các bổ sung cho lớp dialog CHookDlg như sau:

- Trong phần cài đặt, bổ sung biến lưu và hàm xử lý hook:

```
HHOOK oldHook; // Chứa địa chỉ thủ tục hook bị chiếm quyền

LRESULT CALLBACK myHook ( int code, WPARAM wParam,
                           LPARAM lParam) {

    MessageBeep( -1 ); // Thực hiện beep để thông tin
    // Thực hiện thủ tục bị chiếm quyền.
    return CallNextHookEx(oldHook, code, wParam, lParam);
}
```

- Hành vi OnInitDialog xử lý cài đặt thủ tục hook:

```
BOOL CHookDlg::OnInitDialog()
{
    ...
}
```

```

CDialog::OnInitDialog();
oldHook = SetWindowsHookEx (
    WH_KEYBOARD, /* Hook message phím */
    myHook,       /* Thủ tục hook */
    AfxGetInstanceHandle(),
    /* Handle tiến trình chứa thủ tục xử lý hook */
    0             /* Hook tất cả tiến trình */
);
return TRUE;
}

```

- Hành vi OnDestroy hủy bỏ thủ tục hook của ứng dụng:

```

void CHookDlg::OnDestroy()
{
    UnhookWindowsHookEx(oldHook);
    CDialog::OnDestroy();
}

```

- Biên dịch và chạy thử ứng dụng.

12.9 Cài đặt chế độ thực hiện ứng dụng tự động:

SOFTWARE\Microsoft\Windows\CurrentVersion\Run là thành phần đặc biệt của system registry cho phép tự động thực hiện ứng dụng khi khởi động windows thông qua việc cài đặt các mục có giá trị là chuỗi đường dẫn đến chương trình ứng dụng liên quan. Trong đó:

HKEY_LOCAL_MACHINE\ Áp dụng cho mọi người dùng tại host.

HKEY_CURRENT_USER\ Áp dụng cho một người dùng xác định.

📌 Ví dụ: Mục **MyProg = "C:\Game\mci.exe"** cài trong thành phần :

HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Run cho phép thực hiện tự động chương trình **mci.exe** khi khởi động windows.

THỰC HÀNH:

1. Dùng (12.3) và (12.6), thiết kế ứng dụng audio & video player. Ứng dụng có thể thu nhỏ thành icon trên status area để vừa làm việc vừa nghe nhạc.
2. Cài đặt mục trong RUN cho phép thực hiện tự động một ứng dụng tùy ý.
3. HKEY_CURRENT_USER\SOFTWARE\Microsoft\Internet Explorer\TypedURLs là thành phần chứa các URLs trong IE. Viết ứng dụng xóa một URL bất kỳ.

MFC với Internet

13.1 GIAO THỨC TRUYỀN THÔNG TCP/IP:

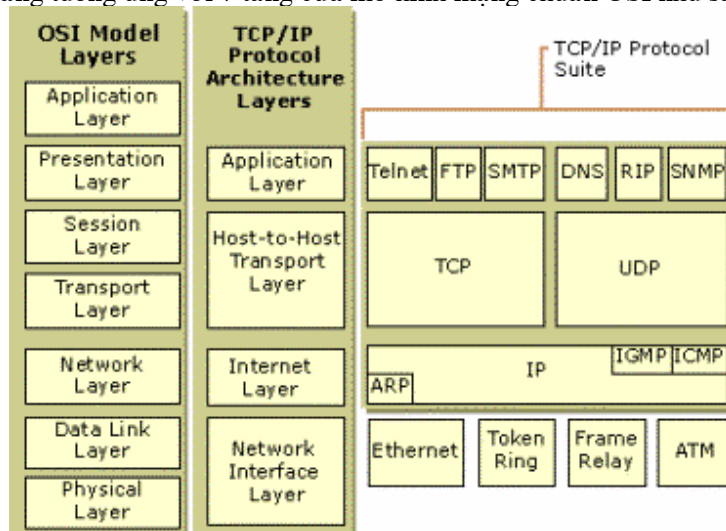
13.1.1 Giới thiệu:

TCP/IP (Transmission Control Protocol/Internet Protocol) bao gồm giao thức truyền thông và các dịch vụ hỗ trợ tác vụ truyền thông giữa các trạm (host) trên hệ thống mạng định vị địa chỉ IP (IP Host Address Internetwork).

TCP/IP ra đời từ năm 1969 bởi cơ quan nghiên cứu các dự án cao cấp thuộc bộ quốc phòng Hoa Kỳ (Department of Defence Advanced Research Projects Agency - DARPA) nhằm mục tiêu xây dựng một giao thức truyền thông chuẩn cho việc phát triển các hệ thống mạng diện rộng (WAN) với cơ chế kết nối truyền thông tốc độ cao trên cơ sở vận dụng các kinh nghiệm từ thành quả phát triển mạng ARPANET, tiền thân của Internet ngày nay.

13.1.2 Kiến trúc của giao thức TCP/IP trên mô hình DARPA:

Mô hình DARPA với giao thức truyền thông TCP/IP là một kiến trúc bao gồm 4 tầng tương ứng với 7 tầng của mô hình mạng chuẩn OSI như sau:



❖ Tầng giao tiếp mạng (Network Interface Layer):

Tầng giao tiếp mạng (tầng truy xuất mạng) đảm nhận nhiệm vụ nhận và gửi các gói chứa thông tin (packet) theo cấu trúc TCP/IP trên thiết bị

kết nối mạng của host. Cấu trúc packet của TCP/IP được thiết kế cho phép không phụ thuộc vào cơ chế truy xuất cũng như kiến trúc khung packet của thiết bị mạng. Nhờ đó, TCP/IP có thể làm việc với nhiều kiểu mạng khác nhau, bao gồm các mạng cục bộ (LAN): Ethernet hoặc Token Ring; mạng diện rộng (WAN): X.25 hoặc Frame Relay. Sự độc lập đó cũng giúp TCP/IP nhanh chóng thích nghi với các công nghệ mạng mới như ATM (Asynchronous Transfer Mode).

❖ Tầng Internet (Internet Layer):

Tầng internet đảm nhận chức năng định vị, đóng gói thông tin và truyền tin định tuyến. Giao thức truyền thông cốt lõi của tầng này là IP, ARP, ICMP, và IGMP.

- Giao thức IP (Internet Protocol): Giao thức truyền thông định tuyến; có nhiệm vụ định vị địa chỉ IP, tách và kết các packet.
- Giao thức ARP (Address Resolution Protocol): Có nhiệm vụ thực hiện hoán chuyển các giá trị địa chỉ một cách tương ứng giữa tầng internet (logic address) và tầng giao tiếp mạng (physic address).
- Giao thức ICMP (Internet Control Message Protocol): Có nhiệm vụ cung cấp các chức năng kiểm soát và thông báo tình hình gửi các IP packet.
- Giao thức IGMP (Internet Group Management Protocol): Có nhiệm vụ quản lý nhóm các IP packet được truyền đến mọi host.

❖ Tầng truyền tải (Transport Layer):

Tầng truyền tải có nhiệm vụ cung cấp cho tầng ứng dụng các dịch vụ truyền thông tin theo dòng và theo gói. Giao thức truyền thông cốt lõi của tầng truyền tải là TCP và UDP.

- **UDP (User Datagram Protocol):** Là giao thức cung cấp dịch vụ truyền thông tin giữa một host với một hay nhiều host khác trên cơ sở đóng gói thông tin và gửi đi theo từng packet độc lập. Giao thức này không thực hiện kiểm tra tình hình nhận thông tin ở host nhận tin nên độ tin cậy thấp, thông tin có thể bị thất lạc.
- **TCP (Transmission Control Protocol):** Là giao thức cung cấp dịch vụ truyền thông tin trên cơ sở xây dựng đường truyền (stream) giữa hai host và thực hiện gửi-nhận thông tin, đồng thời kiểm tra thông tin nhận qua đường truyền này. Giao thức này đảm bảo thông tin được chuyển đến host nhận chính xác và an toàn.

❖ Tầng ứng dụng (Application Layer):

Tầng ứng dụng cung cấp các chức năng khai thác các dịch vụ của các tầng khác, đồng thời định nghĩa các giao thức truyền thông mà ứng dụng của người dùng có thể sử dụng để truyền dữ liệu qua hệ thống mạng. Các giao thức truyền thông phổ biến như sau:

- Giao thức *HTTP* (HyperText Transfer Protocol): Dùng chuyển tải các tập tin tham gia vào nội dung trang WEB (World Wide Web).
- Giao thức *FTP* (File Transfer Protocol): Dùng chuyển tải các tập tin thông thường.
- Giao thức *SMTP* (Simple Mail Transfer Protocol): Dùng chuyển tải nội dung thư tín bao gồm thông điệp và các dữ liệu kèm theo.
- Giao thức *Telnet*: Dùng cho hoạt động thâm nhập host từ xa thông qua các thiết bị đầu cuối (Terminal).

Bên cạnh các giao thức truyền thông nói trên, tầng ứng dụng còn cung cấp các dịch vụ sau:

- Dịch vụ chuyển đổi domain name thành địa chỉ IP tương ứng.
- Dịch vụ cung cấp thông tin định vị địa chỉ IP.
- Dịch vụ quản lý các thiết bị mạng (bộ định tuyến, cầu nối, hub thông minh) nhằm thu thập và trao đổi thông tin quản lý mạng.

13.1.3 Địa chỉ IP:

Địa chỉ IP là giá trị giúp xác định một host duy nhất trên hệ thống mạng. Tất cả các địa chỉ IP đều có dạng thống nhất bao gồm địa chỉ mạng và địa chỉ của host trên mạng đó.

- Địa chỉ mạng (Network address - Network ID): Số hiệu dùng cho một hệ thống mạng các host cùng chung một đặc điểm định tuyến. Các hệ thống mạng kết vào internet phải có địa chỉ mạng phân biệt.
- Địa chỉ host (host address - host ID): Số hiệu dùng cho một host (workstation, server, router, TCP/IP host). Các host trong cùng một hệ thống mạng có cùng địa chỉ mạng nhưng địa chỉ host phải phân biệt.

Mỗi địa chỉ IP có chiều dài 32 bits chia thành 4 bytes (4 octets). Mỗi giá trị nhị phân trong một byte tương ứng với một giá trị thập phân trong đoạn 0 ÷ 255. Bốn giá trị thập phân này được viết ra theo thứ tự và ngăn cách bằng dấu '.' cho ta hình ảnh biểu diễn địa chỉ IP theo dạng số và dấu chấm (dotted decimal notation) mà từ đây ta sẽ gọi tắt là num-dot.

IP: 11000000 10101000 00000011 00011000

Hay: 192.168.3.24

Trên 4 bytes địa chỉ IP, ta có thể chọn một số bytes tùy ý chứa địa chỉ mạng, số bytes còn lại dùng chứa địa chỉ host. Mỗi cách chọn khác nhau tạo thành một lớp địa chỉ IP. Có 3 lớp địa chỉ IP phổ biến:

- Lớp A : 1 byte cho địa chỉ mạng, 3 bytes cho địa chỉ host
- Lớp B : 2 bytes cho địa chỉ mạng, 2 bytes cho địa chỉ host
- Lớp C : 3 bytes cho địa chỉ mạng, 1 byte cho địa chỉ host

Lưu ý:

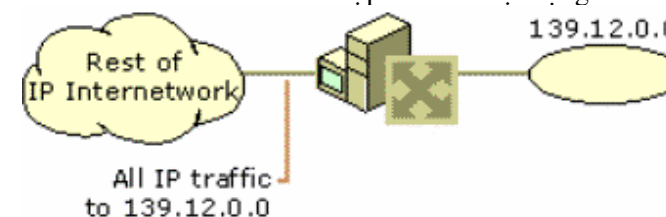
- Giá trị địa chỉ host với tất cả các bits bằng 1 là địa chỉ chỉ cho mọi host trên hệ thống mạng (host broadcast address) chứa host. Không một host nào được sử dụng địa chỉ này.
- Một địa chỉ IP có phần địa chỉ host với tất cả các bits bằng 0 chính là địa chỉ của hệ thống mạng. Không dùng địa chỉ này cho host.

Address	Class First Host ID	Last Host ID
Class A	w.0.0.1	w.255.255.254
Class B	w.x.0.1	w.x.255.254
Class C	w.x.y.1	w.x.y.254

13.1.4 Subnet:

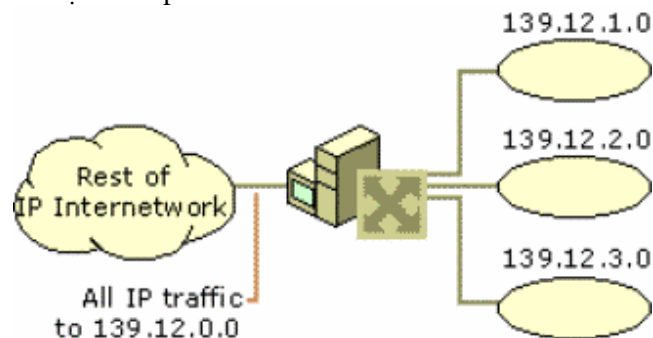
Với số bits dùng cho địa chỉ host trong các lớp địa chỉ, số lượng host của một hệ thống mạng có thể lên đến con số rất lớn (lớp A là 16 triệu). Khi đó, việc gửi một thông điệp lên mạng cho tất cả các host (broadcast) sẽ cần một khoảng thời gian thực hiện không nhỏ, khó đảm bảo xử lý thời gian thực. Hơn nữa, 16 triệu giá trị địa chỉ cho một hệ thống mạng là quá dư thừa.

Nhằm khắc phục hạn chế nói trên, các host cùng hệ mạng được chia thành nhóm nhỏ hơn gọi là mạng con (SubNet). Mỗi subnet tương ứng với một địa chỉ mạng subnet và giới hạn địa chỉ IP các host trực thuộc. Địa chỉ subnet là giá trị hình thành từ một phần bits trong địa chỉ host của địa chỉ IP thuộc hệ mạng ban đầu. Có thể xem Subnet là tập con của hệ mạng.



Hệ thống mạng trong hình trên sử dụng địa chỉ lớp B. Địa chỉ của hệ mạng là 139.12.0.0. Hệ mạng này cho phép xác lập 65535 địa chỉ host. Thực

hiện chia hệ mạng trên thành 256 subnet dựa trên byte thứ ba, ta được các subnet 8-bit địa chỉ lớp B:



☞ Các subnet được tạo thành là: 139.12.1.0, 139.12.2.0 và 139.12.3.0.

13.1.5 Subnet Mask:

Subnet mask là một giá trị 32 bits giúp tách giá trị địa chỉ mạng (hoặc địa chỉ subnet) và địa chỉ host từ một địa chỉ IP bất kỳ (trong một lớp địa chỉ bất kỳ, cách phân chia subnet bất kỳ). Giá trị này được xây dựng như sau:

- Các giá trị bit tương ứng với địa chỉ mạng có giá trị là 1.
- Các giá trị bit tương ứng với địa chỉ host có giá trị là 0.

Giá trị subnet mask cũng được biểu diễn dưới dạng num-dot.

Ta có subnet mask mặc nhiên cho các lớp địa chỉ như sau:

Class	Bits for Subnet Mask	Subnet Mask
Class A	11111111 00000000 00000000 00000000	255.0.0.0
Class B	11111111 11111111 00000000 00000000	255.255.0.0
Class C	11111111 11111111 11111111 00000000	255.255.255.0

Các giá trị subnet mask do người dùng tạo ra tương ứng với mỗi lớp địa chỉ trên có thể khác biệt so với các giá trị mặc nhiên vì chúng chứa cả giá trị mask trên địa chỉ subnet.

Ví dụ: 138.96.58.0 là một địa chỉ subnet 8-bit lớp B. 8 bits địa chỉ host của hệ mạng ban đầu được dùng làm giá trị địa chỉ subnet. Như vậy subnet mask sử dụng tổng cộng 24 bits (255.255.255.0) để định nghĩa địa chỉ mạng subnet. Địa chỉ mạng subnet và giá trị subnet mask tương ứng được biểu diễn theo dạng num-dot như sau:

138.96.58.0, 255.255.255.0

Hay: 138.96.58.0/24 (24-bit mask)

☞ **Xác định địa chỉ mạng:** Để tách địa chỉ mạng từ một địa chỉ IP bất kỳ thông qua giá trị subnet mask, ta sử dụng phép toán AND bits:

Ví dụ: Giả sử IP = 129.56.189.41 và subnet mask = 255.255.240.0

Địa chỉ mạng được xác định như sau:

IP Address : 10000001 00111000 10111101 00101001

Subnet Mask : 11111111 11111111 11110000 00000000

Network ID : 10000001 00111000 10110000 00000000

☞ **Xác định giới hạn địa chỉ IP:** Giả sử địa chỉ mạng là 192.168.0.0.

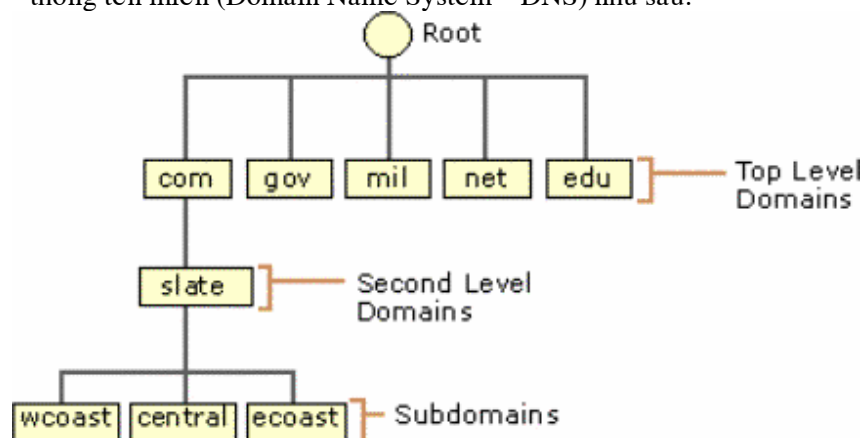
Với subnet 3-bit địa chỉ lớp B của hệ mạng nói trên, ta có 8 trường hợp lựa chọn giá trị cho 3 bits này. Tương ứng với mỗi trường hợp là một giới hạn các địa chỉ IP của các host trong subnet:

Stt	Địa chỉ Subnet theo hệ nhị phân	Giới hạn địa chỉ IP
1	11000000.10101000.00000000.00000001- 11000000.10101000.00011111.11111110	192.168.0.1 - 192.168.31.254
2	11000000.10101000.00100000.00000001- 11000000.10101000.00111111.11111110	192.168.32.1 - 192.168.63.254
3	11000000.10101000.01000000.00000001- 11000000.10101000.01011111.11111110	192.168.64.1 - 192.168.95.254
4	11000000.10101000.01100000.00000001- 11000000.10101000.01111111.11111110	192.168.96.1 - 192.168.127.254
5	11000000.10101000.10000000.00000001- 11000000.10101000.10011111.11111110	192.168.128.1 - 192.168.159.254
6	11000000.10101000.10100000.00000001- 11000000.10101000.10111111.11111110	192.168.160.1 - 192.168.191.254
7	11000000.10101000.11000000.00000001- 11000000.10101000.11011111.11111110	192.168.192.1 - 192.168.223.254
8	11000000.10101000.11100000.00000001- 11000000.10101000.11111111.11111110	192.168.224.1 - 192.168.255.254

13.1.6 Host domain name:

Tên (name) là một giải pháp hữu hiệu cho việc gợi nhớ địa chỉ của host thay vì dùng địa chỉ IP với 4 bytes giá trị khó nhớ nói trên. Tên của host (Host name) là một chuỗi ký tự có chiều dài tối đa 255, có thể chứa mẫu tự, ký số, các ký tự '-' và '.' và có ý nghĩa tương đương với địa chỉ IP trong việc quản lý địa chỉ một host trên hệ thống mạng internet. Có hai dạng phổ biến cho tên của host là nick name và domain name:

- Nick name: Một nhãn được dùng cho một địa chỉ IP duy nhất.
- Domain name: Tên được hình thành từ cấu trúc phân lớp. Cấu trúc phân lớp này được qui định phổ biến thành luật và được gọi là hệ thống tên miền (Domain Name System – DNS) như sau:



Trong đó:

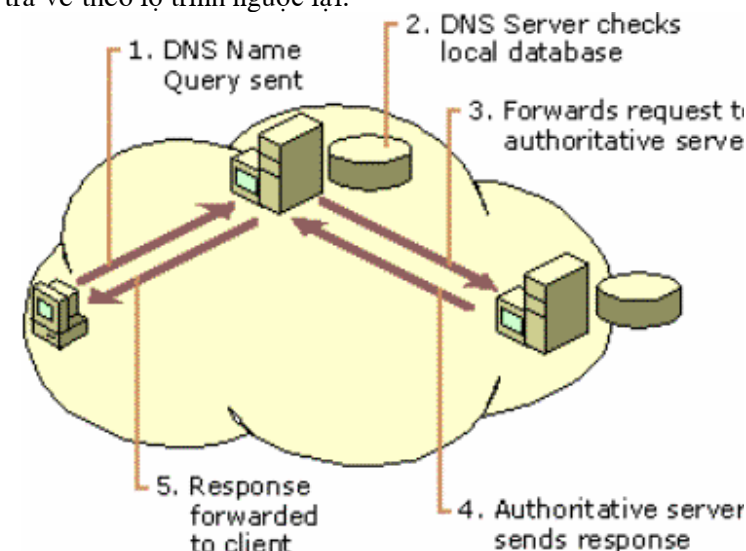
Tên Domain	Ý nghĩa sử dụng
COM	Các tổ chức thương mại
EDU	Các cơ quan giáo dục, nghiên cứu
GOV	Tổ chức chính phủ
MIL	Cơ quan quân sự
NET	cơ quan quản lý mạng chính
ORG	Các tổ chức khác
INT	Các tổ chức quốc tế
<country code>	Các nhánh cho các quốc gia trừ Hoa Kỳ

Một tên miền đầy đủ (Fully Qualified Domain Name – FQDN) chứa đường đi từ gốc đến đối tượng tham chiếu theo trình tự phân cấp nói trên.

Ví dụ: ftpsrv.wcoast.slate.com

Việc chuyển đổi giữa địa chỉ IP và Domain name được thực hiện dựa trên bảng chuyển đổi IP-DomainName do DNS server, một host chuyên dụng của hệ thống mạng, quản lý. Ứng dụng từ một host bất kỳ có thể truy vấn bảng thông tin này thông qua các dịch vụ cung cấp bởi windows socket. Windows socket truyền yêu cầu của ứng dụng đến bộ phận phân giải domain name của giao thức truyền thông TCP/IP. Bộ phận này chuyển yêu cầu đến DNS server. DNS server nhận yêu cầu và thực hiện; nếu

thông tin yêu cầu không xác định được thì nó sẽ chuyển đến DNS server cùng cấp khác, kết quả thực hiện được hoặc không thực hiện được đều được trả về theo lộ trình ngược lại.



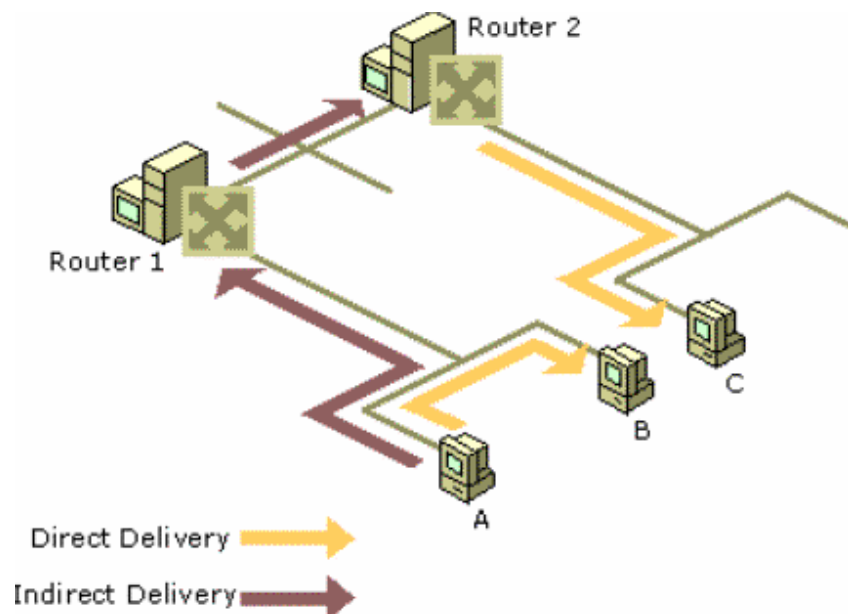
13.1.7 IP Routing:

IP routing là tiến trình xử lý gửi packet đến host nhận dựa trên địa chỉ IP của host. Tiến trình này xảy ra ở host gửi thông tin theo giao thức TCP/IP và ở thiết bị định tuyến (router device) nhằm thực hiện quyết định lựa chọn vị trí mà packet sẽ được chuyển đến.

Để thực hiện quyết định trên, bảng IP tham khảo bảng định tuyến được lưu trữ trong bộ nhớ. Nội dung bảng định tuyến được khởi tạo mặc nhiên khi TCP/IP vừa được khởi động. Các mục bổ sung có thể được thực hiện bởi người quản trị hệ thống mạng (WinNT router table) hoặc thực hiện một cách tự động trong quá trình liên lạc với các bộ định tuyến.

Có hai dạng phổ biến khi gửi packet; gửi trực tiếp và gửi gián tiếp.

- Gửi trực tiếp (Direct delivery): Xảy ra khi host nhận và host gửi được kết nối trực tiếp. Thông tin được đóng gói ở host gửi theo cấu trúc qui định của tầng giao tiếp mạng và được gửi đi.
- Gửi gián tiếp (Indirect delivery): Xảy ra khi host nhận và host gửi được kết nối thông qua một trung gian (bộ định tuyến). Khi đó sẽ có một quá trình gửi gián tiếp từ host đến bộ định tuyến, từ bộ định tuyến trực tiếp đến host nhận (hoặc gián tiếp đến một bộ định tuyến khác).



- A gửi trực tiếp đến B
- A gửi gián tiếp packet đến router1 , router 1 gửi gián tiếp đến router 2, router 2 gửi trực tiếp đến C.

▮ Bảng định tuyến:

Bảng định tuyến được xác lập trên tất cả các host (node, router) và được đặt bởi các giá trị mặc nhiên trong quá trình khởi động của giao thức TCP/IP. Nội dung của bảng chứa thông tin về hệ thống các địa chỉ IP trên mạng, cách kết nối với các địa chỉ ấy.

Mỗi khi một gói thông tin được gửi đi, bảng định tuyến sẽ được sử dụng để xác định:

- *Địa chỉ của host nơi gửi đến:* Nếu gửi trực tiếp thì đó chính là địa chỉ của host nhận packet, ngược lại, là địa chỉ của bộ định tuyến.
- *Giao diện sử dụng để gửi:* Bao gồm thông tin về cấu trúc vật lý và logic của thiết bị kết nối mạng ở nơi gửi và nơi nhận.

Cấu trúc nội dung của một mục trong bảng định tuyến:

[**Network ID, Subnet Mask, Next Hop, Interface, Metric**]

Trong đó:

- Network ID: Địa chỉ mạng tương ứng với tuyến truyền tin.
- Subnet Mask: Giá trị dùng tách địa chỉ mạng từ địa chỉ IP.
- Next Hop: Địa chỉ IP của host trung gian kế tiếp.

- Interface: Thiết bị giao tiếp mạng được sử dụng.
- Metric: Chi phí của tuyến truyền, làm cơ sở cho việc lựa chọn tuyến tối ưu.

Thông tin của mục trong bảng định tuyến qui định đặc điểm tuyến:

- Tuyến kết nối trực tiếp với hệ thống mạng: Giá trị của Next Hop là rỗng hoặc chứa địa chỉ IP của thiết bị giao tiếp mạng.
- Tuyến kết nối trung gian với hệ thống mạng: Giá trị của Next Hop chứa địa chỉ IP của bộ định tuyến trung gian giữa host gửi và host nhận.
- Tuyến kết nối trực tiếp với một host cụ thể: Khi đó Network ID chứa địa chỉ của host và giá trị của subnet mask là 255.255.255.255.
- Tuyến mặc nhiên: Tuyến được sử dụng khi có một tác vụ định tuyến không thành công. Giá trị của network ID là 0.0.0.0 và subnet mask là 0.0.0.0.

Network Address	Netmask	Gateway	Address Interface	Metric	Purpose
0.0.0.0	0.0.0.0	157.55.16.1	157.55.27.90	1	Default Route
127.0.0.0	255.0.0.0	127.0.0.1	127.0.0.1	1	Loopback Network
157.55.16.0	255.255.240.0	157.55.27.90	157.55.27.90	1	DirectlyAttached Network
157.55.27.90	255.255.255.255	127.0.0.1	127.0.0.1	1	Local Host
157.55.255.255	255.255.255.255	157.55.27.90	157.55.27.90	1	Network Broadcast
224.0.0.0	224.0.0.0	157.55.27.90	157.55.27.90	1	Multicast Address
255.255.255.255	255.255.255.255	157.55.27.90	157.55.27.90	1	Limited Broadcast

(Một bảng định tuyến của Windows NT)

▮ Xử lý định tuyến:

Xử lý định tuyến là xử lý thực hiện lựa chọn mục định tuyến trong bảng định tuyến và dùng nó cho việc gửi thông tin. Việc lựa chọn này được thực hiện thông qua các bước sau:

- Trên mỗi mục định tuyến, thực hiện phép toán AND giữa địa chỉ host nhận và giá trị subnet mask. Kiểm tra kết quả này với Network ID để đánh giá độ phù hợp (độ tương tự).
- Chọn ra các mục định tuyến có mức độ phù hợp cao nhất. Trong các mục định tuyến này, chọn ra các mục có chi phí tuyến truyền nhỏ nhất. Cuối cùng, từ các mục định tuyến chọn được, mục định tuyến được sử dụng là mục còn rỗi.

☞ *Xử lý định tuyến khác nhau trên các loại host khác nhau.*

Host gửi: Packet được gửi từ giao thức ở tầng cấp cao hơn của IP (TCP, UDP, ...). Khi đó vai trò IP ở host gửi như sau:

- Đặt giá trị cho trường TTL (Times-To-Live) theo giá trị qui định của ứng dụng tầng cấp cao hoặc lấy giá trị mặc nhiên của hệ thống.
- Xác định tuyến tối ưu cho packet truyền đi.
- Nếu không xác định được tuyến truyền thì thông báo lỗi đến tầng truyền thông ở cấp cao hơn. Ngược lại, thực hiện truyền theo tuyến.

Host định tuyến: Vai trò của IP như sau:

- Kiểm tra checksum của packet. Nếu sai thì hủy packet.
- Kiểm tra địa chỉ IP của host nhận trên packet. Nếu địa chỉ này là địa chỉ một bộ định tuyến thì nội dung packet (trừ phần nội dung IP header) được chuyển cho giao thức tầng cấp cao tương ứng. Ngược lại, giảm giá trị trường TTL đi 1; nếu giá trị trường này bằng 0 thì hủy bỏ packet và gửi thông điệp "ICMP Time Expired-TTL Expired" cho host gửi, ngược lại xác định tuyến gửi và gửi packet.

Host nhận: Vai trò của IP như sau:

- Kiểm tra checksum của packet. Nếu sai thì hủy packet.
- Nếu địa chỉ host nhận ghi trên packet không phải là địa chỉ của host đang xử lý thì hủy packet.
- Gửi nội dung packet (trừ IP header) lên giao thức tầng cao hơn.

13.2 LẬP TRÌNH TCP/IP VỚI WINSOCK:

Winsock (Windows Socket) có xuất xứ từ BSD (Berkeley Software Distribution - UNIX), tương thích với windows qua phiên bản WinSock1.1. Winsock là một giao diện với các dịch vụ xây dựng trên giao thức truyền thông TCP và UDP. Thông qua winsock, ứng dụng có thể triển khai dễ dàng các tác vụ truyền thông trên tầng truyền thông của mô hình mạng.

13.2.1 Port:

Port là khái niệm được diễn tả bằng một giá trị số (số hiệu port) giúp phân biệt các tiến trình trên cùng một host đồng sử dụng giao thức TCP/IP. Các ứng dụng khác nhau sử dụng TCP/IP có thể thực hiện được cùng một lúc trên một host với điều kiện chúng phải sử dụng các số hiệu port khác nhau.

13.2.2 Socket:

Socket là điểm truyền thông đầu cuối cho phép ứng dụng gửi và nhận các packet qua đường truyền mạng. Mỗi socket có một kiểu xác định và gắn liền với một tiến trình xử lý truyền thông. Có hai loại socket:

- **Stream Socket:** Cơ chế truyền dữ liệu theo dòng.
 - **Datagram Socket:** Cơ chế truyền dữ liệu theo packet.
- ☞ Địa chỉ socket là một giá trị bao gồm địa chỉ của host và số hiệu port mà tiến trình xử lý truyền thông liên quan socket đã đăng ký sử dụng.

13.2.3 Một số cấu trúc dữ liệu của Winsock API:

- **Địa chỉ num-dot của host:**

Được biểu diễn bởi một giá trị chuỗi có nội dung là bốn giá trị số có độ lớn bằng byte và được ngăn cách bằng dấu ‘.’

Ví dụ: "127.0.0.1"

- **Cấu trúc khởi động winsock:**

```
typedef struct WSADATA {  
    WORD        wVersion;  
    WORD        wHighVersion;  
    char        szDescription[WSADESCRIPTION_LEN+1];  
    char        szSystemStatus[WSASYS_STATUS_LEN+1];  
    unsigned short iMaxSockets;  
    unsigned short iMaxUdpDg;  
    char FAR *   lpVendorInfo;  
} WSADATA, *LPWSADATA;
```

szDescription: Thông tin về phiên bản winsock đang sử dụng.

szSystemStatus: Thông tin về phiên bản hệ điều hành windows.

- **Cấu trúc chứa thông tin của host:**

```
struct hostent {  
    char FAR * h_name;           // Domain name  
    char FAR * FAR * h_aliases; // Tên dùng thay domain name  
    short h_addrtype;           // Kiểu địa chỉ, với TCP/IP là AF_INET  
    short h_length;             // Kích thước địa chỉ, 4 bytes cho AF_INET  
    char FAR * FAR * h_addr_list; // Danh sách địa chỉ. Phần tử  
                                   // đầu danh sách h_addr_list[0]  
                                   // (hay h_addr) là địa chỉ host.  
} HOSTENT, *PHOSTENT;
```

- **Địa chỉ socket trên một host:**

```
struct sockaddr {
```

```

    unsigned short sa_family; // Họ địa chỉ của host.
    char sa_data[14];         // Địa chỉ (cho phép nhiều loại địa chỉ)
} SOCKADDR ;

```

Cấu trúc này được tương thích bởi SOCKADDR_IN của MicroSoft:

```

struct sockaddr_in {
    short sin_family;           // Có giá trị là AF_INET
    unsigned short sin_port;    // Số hiệu port
    struct in_addr sin_addr;    // Địa chỉ IP 4 bytes của host
    char sin_zero[8];          // Chỉ để tương thích với SOCKADDR
} SOCKADDR_IN ;

```

sin_addr: Địa chỉ IP 4 bytes của host, có các cách diễn tả như sau:

```

struct in_addr {
    union {
        struct {
            unsigned char s_b1, s_b2, s_b3, s_b4;
        } S_un_b;                // Địa chỉ IP bằng 4 bytes
        struct {
            unsigned short s_w1, s_w2;
        } S_un_w;                // Địa chỉ IP bằng 2 words
        unsigned long S_addr;    // Địa chỉ IP bằng 1 long
    } S_un;
};

```

13.2.4 Một số dịch vụ của Winsock API:

- **char FAR * inet_ntoa (**
 struct in_addr in // Địa chỉ IP 4 bytes
); Trả về chuỗi địa chỉ num-dot tương ứng.
- **unsigned long inet_addr (**
 const char FAR *cp // Địa chỉ IP num-dot
); Hàm trả về giá trị kiểu **long** của địa chỉ IP 4 bytes tương ứng.
- **int WSASStartup (** // Hàm trả về giá trị 0 nếu thành công
 WORD version, // Phiên bản winsock
 LPWSADATA pMyInfo // Con trỏ cấu trúc nhận thông tin.
); Khởi động, chuẩn bị cho việc khai thác các dịch vụ của winsock.
 version: Chứa số hiệu phiên bản winsock cần dùng
 - High-order byte = winsock minor version number.

- Low-order byte = winsock major version number.
- Ví dụ: MAKEWORD (1, 1) → Winsock version 1.1.

- **int gethostname (** // Hàm trả về SOCKET_ERROR nếu có lỗi
 char FAR *name, // Vùng đệm chứa thông tin
 int namelen // Kích thước vùng đệm
); Lấy thông tin về tên của host.
- **PHOSTENT gethostbyname (**
 const char FAR *name // Tham số chứa tên host
); Hàm trả về con trỏ đến cấu trúc HOSTENT chứa thông tin của host.
- **int WSACleanup();** Chấm dứt sử dụng dịch vụ winsock.

13.3 MFC VỚI LẬP TRÌNH WINSOCK:

Thư viện sử dụng: **AfxSock.h**

13.3.1 Khởi động Winsock:

BOOL AfxSocketInit (WSADATA* lpwsaData = NULL);

AfxSocketInit đảm nhận việc thực hiện WSASStartup khi bắt đầu ứng dụng và WSACleanup khi ứng dụng kết thúc. Lời gọi AfxSocketInit được thực hiện trong hành vi InitInstance của đối tượng quản lý tiểu trình chính của ứng dụng. Tham số truyền cho hàm có ý nghĩa như với WSASStartup.

13.3.2 Lớp CAsyncSocket:

CAsyncSocket là lớp đối tượng quản lý socket. Bằng sự bao hàm các dịch vụ của winsock API trong các hành vi, lớp CAsyncSocket cho phép tạo ra đối tượng socket hỗ trợ ứng dụng một cách hiệu quả trong việc triển khai các hoạt động truyền thông trên tầng truyền tải của giao thức TCP/IP.

Các hành vi đặc trưng lớp CAsyncSocket như sau:

- **CAsyncSocket();** Khởi tạo đối tượng socket rỗng.
- **BOOL Create (**
 UINT nSocketPort = 0,
 int nSocketType = SOCK_STREAM,
 long lEvent = FD_READ | FD_WRITE | FD_OOB |
 FD_ACCEPT | FD_CONNECT | FD_CLOSE,
 LPCTSTR lpszSocketAddress = NULL
); Khởi tạo thông số đối tượng socket và kết với host đối tác (nếu có).

Trong đó:

- *nSocketPort*: Số hiệu port của socket. Giá trị này có thể xác định bởi người dùng (ví dụ 2050) hoặc đặt bằng 0 để winsock tìm giúp một giá trị phù hợp.

- *nSocketType*: Ấn định giao thức sử dụng; SOCK_STREAM cho TCP hoặc SOCK_DGRAM cho UDP.
- *IEvent*: Bao gồm các thông số liên quan đến các biến cố truyền thông mà ứng dụng muốn winsock phản hồi ngay trên socket quản lý bởi đối tượng socket.
 FD_READ : Nhận được thông tin.
 FD_WRITE : Gửi được thông tin.
 FD_ACCEPT : Đồng ý kết nối.
 FD_CONNECT : Đề nghị kết nối.
 FD_CLOSE : Chấm dứt kết nối.
 FD_OOB : Thông tin ngoài tuyến.
- *lpzSocketAddress*: Chuỗi chứa địa chỉ num-dot của host đối tác mà socket được kết (bind). **INADDR_ANY** là địa chỉ dùng kết với tất cả các host trong hệ thống mạng.

▪ **BOOL Bind** (
 UINT *nSocketPort*, // Số hiệu port
 LPCTSTR *lpzSocketAddress* = NULL // Địa chỉ num-dot của host
); Kết socket với một host xác định.

▪ **BOOL Bind** (
 const SOCKADDR* *IpSockAddr*, // Địa chỉ socket host đối tác
 int *nSockAddrLen* // Kích thước *IpSockAddr*
); Kết socket với một host xác định.

▪ **BOOL SetSockOpt** (
 int *nOptionName*, // Thuộc tính cần ấn định
 const void* *IpOptionValue*, // Địa chỉ biến chứa giá trị ấn định
 int *nOptionLen*, // Kích thước biến chứa giá trị
 int *nLevel* = SOL_SOCKET // Mức đặt thông số cho socket
); Ấn định đặc tính hoạt động của socket.

nOptionName cho phép lựa chọn thuộc tính ấn định của socket :

- SO_BROADCAST : **BOOL** → Gửi message đến mọi host.
- SO_DONTROUTE : **BOOL** → Gửi trực tiếp không qua router.

Ví dụ: Nếu ấn định đặc tính của socket là cho phép gửi message đến mọi host thì hành vi trên được thực hiện với tham số như sau:

```
int nOptionName,           = SO_BROADCAST
const void* IpOptionValue, = Địa chỉ biến luận lý bằng TRUE
int nOptionLen,           = sizeof (BOOL)
```

```
int nLevel                = SOL_SOCKET
▪ BOOL Listen (
    int nConnectionBacklog = 5 // Chiều dài dòng chờ kết nối
); Chờ nhận kết nối từ host đối tác (server chờ các clients).
▪ BOOL Connect (
    LPCTSTR lpzHostAddress, // Địa chỉ num.dot của host đối tác
    UINT nHostPort          // Số hiệu port trên host đối tác
); Xin kết nối với host đối tác (clients kết nối với server).
▪ BOOL Connect (
    const SOCKADDR* IpSockAddr, // Địa chỉ socket host đối tác
    int nSockAddrLen           // Kích thước IpSockAddr
); Xin kết nối với host đối tác (clients kết nối với server).
▪ virtual BOOL Accept (
    CAsyncSocket& rConnectedSocket,
    SOCKADDR* IpSockAddr = NULL,
    int* IpSockAddrLen = NULL
); Chấp nhận kết nối với host đối tác (server chấp nhận kết nối client).
Trong đó:
- rConnectedSocket: Biến chứa đối tượng socket được tạo mới với
    các đặc tính giống như đối tượng chủ thể và
    dùng để quản lý liên kết vừa thiết lập.
- IpSockAddr: Địa chỉ biến kiểu SOCKADDR được dùng để
    nhận địa chỉ host đối tác.
- IpSockAddrLen: Địa chỉ biến được dùng để nhận kích thước của
    địa chỉ trả về trong IpSockAddr.
▪ virtual int Send (
    const void* IpBuf, // Sử dụng cho Stream Socket
    // Địa chỉ vùng đệm chứa dữ liệu truyền
    int nBufLen,       // Kích thước vùng đệm
    int nFlags = 0     // Thông số ấn định đặc tính gửi
); Gửi dữ liệu thông qua một socket được kết nối.
Thông số dùng cho đặc tính gửi:
- MSG_DONTROUTE = Gửi thẳng không qua router.
- MSG_OOB       = Gửi ngoại tuyến (khẩn cấp)
▪ int SendTo (
    const void* IpBuf, // Sử dụng cho Datagram
    // Vùng đệm dữ liệu
    int nBufLen,       // Kích thước vùng đệm
```

```

UINT nHostPort,                // Số hiệu port đối tác
LPCTSTR lpszHostAddress = NULL, // Địa chỉ num-dot đối tác
int nFlags = 0                  // Đặc tính gửi.
); Gửi packet đến host đối tác. Hàm trả về số bytes dữ liệu gửi được.
Để gửi packet đến tất cả các host, đặt lpszHostAddress = NULL.
▪ int SendTo (                  // Sử dụng cho Datagram
    const void* lpBuf,          // Như trên ...
    int nBufLen,
    const SOCKADDR* lpSockAddr, // Địa chỉ socket đối tác
    int nSockAddrLen,           // Kích thước địa chỉ
    int nFlags = 0
); Gửi packet đến host đối tác. Hàm trả về số bytes dữ liệu được gửi.
Để gửi packet đến tất cả các host, giá trị địa chỉ đối tác đặt như sau:
    lpSockAddr->sin_addr.s_addr = htonl ( INADDR_BROADCAST );
▪ virtual int Receive (         // Sử dụng cho Stream Socket
    void* lpBuf,                // Địa chỉ vùng đệm nhận dữ liệu
    int nBufLen,                // Kích thước vùng đệm
    int nFlags = 0              // Đặc tính nhận dữ liệu.
); Nhận dữ liệu thông qua một socket được kết nối.
▪ int ReceiveFrom (             // Sử dụng cho Datagram
    void* lpBuf,                // Địa chỉ vùng đệm chứa dữ liệu
    int nBufLen,                // Kích thước vùng đệm
    CString& rSocketAddress,    // Địa chỉ num-dot đối tác
    UINT& rSocketPort,          // Số hiệu port đối tác
    int nFlags = 0              // Đặc tính gửi
); Nhận dữ liệu từ host đối tác.
▪ int ReceiveFrom (             // Sử dụng cho Datagram
    void* lpBuf,                // Địa chỉ vùng đệm chứa dữ liệu
    int nBufLen,                // Kích thước vùng đệm
    SOCKADDR* lpSockAddr,       // Địa chỉ socket đối tác
    int* lpSockAddrLen,         // Kích thước địa chỉ
    int nFlags = 0              // Đặc tính gửi
); Nhận dữ liệu từ host đối tác.
▪ BOOL AsyncSelect ( long lEvent = FD_READ | FD_WRITE |
    FD_OOB | FD_ACCEPT | FD_CONNECT | FD_CLOSE );
    ☞ Đăng ký biến cố truyền thông mong muốn (xem Create).

```

- virtual void **OnConnect** (
 - int nErrorCode // Mã lỗi
); Nhận được yêu cầu xin kết nối của host đối tác.
- virtual void **OnAccept** (
 - int nErrorCode // Mã lỗi, = 0 is OK
); Nhận được sự đồng ý kết nối của host đối tác.
- virtual void **OnSend** (
 - int nErrorCode
); Socket đã sẵn sàng cho việc gửi dữ liệu.
- virtual void **OnReceive** (
 - int nErrorCode
); Dữ liệu đã sẵn sàng chờ nhận thông qua socket.
- virtual void **OnOutOfBandData** (
 - int nErrorCode
); Có dữ liệu khẩn cấp sẵn sàng chờ nhận thông qua socket.
- virtual void **OnClose** (
 - int nErrorCode
); Socket chuẩn bị chấm dứt hoạt động.
- BOOL **ShutDown** (
 - int nHow // 0 = receive, 1 = send ; 2 = both
); Chấm dứt tác vụ truyền thông tương ứng trên socket.
- virtual void **Close**(); Hủy bỏ socket.

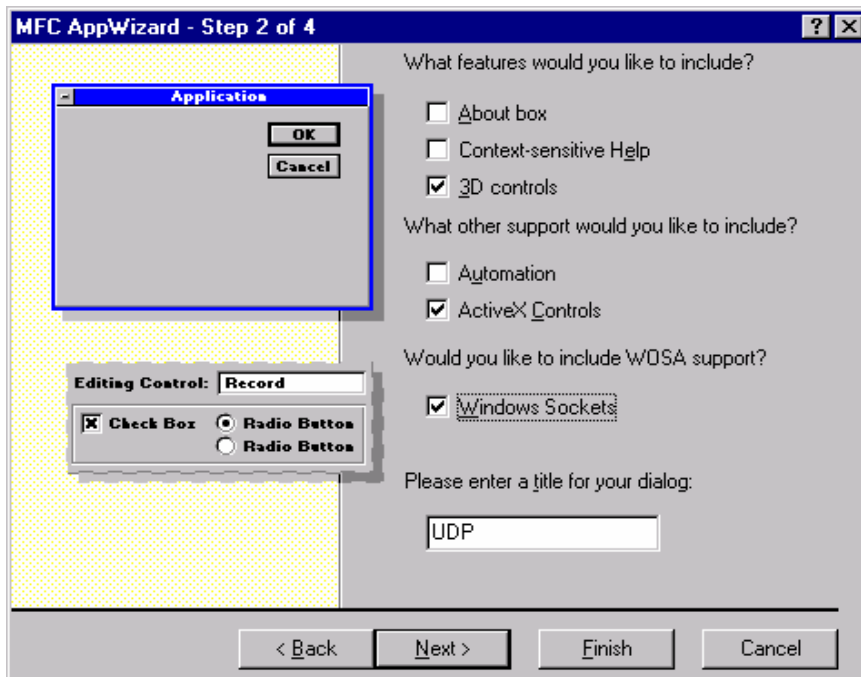
13.4 LẬP TRÌNH WINSOCKET CHO GIAO THỨC UDP:

Trong phần này ta sử dụng giao thức UDP để thực hiện truyền thông điệp giữa hai hay nhiều host. Ứng dụng thiết kế cho phép người dùng soạn thảo nội dung thông điệp, tùy ý chọn gửi đến host xác định hoặc tất cả các host.



Cách thực hiện như sau:

- Dùng MFC Wizard tạo dự án **Udp** với cửa sổ giao diện chính là dialog.
- Ở bước '**Step 2 of 4**': Chọn hỗ trợ Windows Sockets:



- Chọn Finish hoàn để hoàn tất việc khởi tạo dự án.

Tiếp tục thực hiện các bổ sung sau :

- Bổ sung lớp CEmpUdp kế thừa lớp CAsyncSocket của MFC. Thực hiện các cài đặt cho lớp như sau:

- Khai báo các hằng, số hiệu port sử dụng trong ứng dụng:

```
const BUF_LEN      = 512;      // Chiều dài vùng đệm
const SENT_PORT    = 2051;     // port gửi thông tin
const RECEIV_PORT  = 2050;     // port nhận thông tin
```

- Các đối tượng thuộc tính sử dụng trong lớp:

```
char      *m_Buffer;    // Vùng đệm chứa thông tin nhận-gửi
CDialog   *m_Parent;    // Con trỏ đối tượng dialog liên quan
```

- Hành vi khởi tạo của lớp nhận tham số là con trỏ đối tượng dialog giao diện liên quan; thực hiện lưu giá trị con trỏ vào thuộc tính của lớp và đăng ký vùng nhớ làm vùng đệm nhận-gửi thông tin:

```
CEmpUdp::CEmpUdp(CDialog *parent)
{
    m_Parent = parent;
    m_Buffer = new char[BUF_LEN + 1];
```

```
}
- Hành vi hủy bỏ thực hiện giải phóng vùng nhớ đã xin cấp phát:
CEmpUdp::~CEmpUdp()
{
    delete m_Buffer;
}
```

- Hành vi kế thừa OnSend lấy thông tin nhập trong hộp nhập có số hiệu là IDC_EDIT của dialog liên quan và thực hiện gửi đến host (nếu địa chỉ xác định) hoặc gửi cho tất cả host trên mạng:

```
void CEmpUdp::OnSend(int nErrorCode)
{
    CString peerAddr;
    m_Parent->GetDlgItemText(IDC_EDIT, m_Buffer, BUF_LEN);
    ((CUDPDlg*)m_Parent)->m_ip.GetWindowText(peerAddr);
    if (peerAddr == "0.0.0.0") {
        // Send to everyone
        BOOL toAll = TRUE;
        SetSockOpt(SO_BROADCAST, &toAll, sizeof(BOOL));
        SendTo(m_Buffer, BUF_LEN, RECEIV_PORT);
    }
    else
        SendTo(m_Buffer, BUF_LEN, RECEIV_PORT, peerAddr);
}
```

- Hành vi kế thừa OnReceive nhận thông tin gửi đến và hiển thị trong hộp thông báo có số hiệu là IDC_READ của dialog liên quan:

```
void CEmpUdp::OnReceive(int nErrorCode)
{
    CString peerAddr;      // Địa chỉ host đối tác và
    UINT port;             // số hiệu port sử dụng
    ReceiveFrom(m_Buffer, BUF_LEN, peerAddr, port);
    sprintf(m_Buffer, "%s\n(from %s)", m_Buffer, peerAddr);
    m_Parent->SetDlgItemText(IDC_READ, m_Buffer);
    AsyncSelect(FD_READ);  // Duy trì việc nhận thông tin
}
```

- Mở resource của dialog giao diện, bổ sung hộp edit (IDC_EDIT) cho phép nhập nội dung thông tin gửi, hộp thông báo (IDC_READ) hiển thị thông tin nhận, và hộp nhập IP với biến m_IP dùng nhập địa chỉ IP.

- Thực hiện các bổ sung cho lớp CUDPDlg quản lý dialog giao diện:
 - Khai báo các đối tượng thuộc tính quản lý socket nhận và gửi thông tin của ứng dụng:

```
CEmpUdp *m_sentUDP;    // Quản lý socket gửi thông tin
CEmpUdp *m_receivUDP;  // Quản lý socket nhận thông tin
```

- Hành vi OnInitDialog: Cấp phát và khởi tạo thông số cho các đối tượng CEmpUdp:

```
m_sentUDP = new CEmpUdp(this);
m_receivUDP = new CEmpUdp(this);
m_sentUDP->Create(SENT_PORT, SOCK_DGRAM, 0);
m_receivUDP->Create(RECEIV_PORT, SOCK_DGRAM, FD_READ);
```

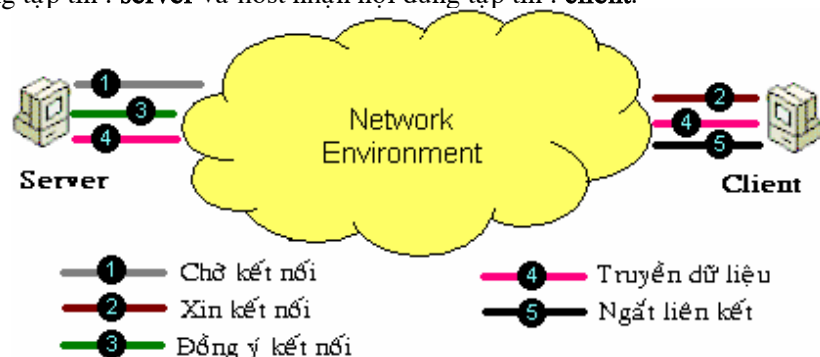
- Hành vi OnOK (tương ứng nút chọn OK) thực hiện gửi thông điệp:

```
void CUDPDlg::OnOK()
{
    m_sentUDP->AsyncSelect(FD_WRITE); // Đăng ký gửi
}
```

- Biên dịch và chạy thử ứng dụng.

13.5 LẬP TRÌNH WINSOCKET CHO GIAO THỨC TCP:

Trong phần này, ta sử dụng giao thức TCP để thực hiện truyền dữ liệu là nội dung của một tập tin. Việc truyền dữ liệu diễn ra giữa host quản lý nội dung tập tin : **server** và host nhận nội dung tập tin : **client**.



Thứ tự thực hiện các tác vụ kết nối và truyền dữ liệu giữa host server và host client như sau:

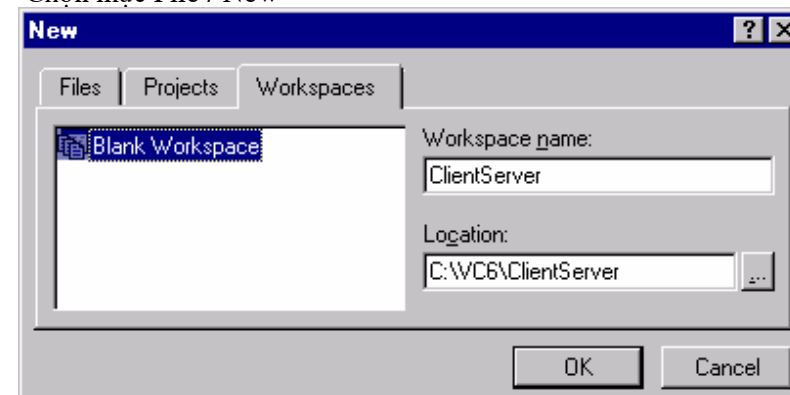
Server	Client
// Khai báo socket	// Khai báo socket

CSocket sockSrvr;	CSocket sockClient;
// Tạo socket ^{1,2} sockSrvr.Create(nPort);	// Tạo socket ² sockClient.Create();
// Chờ nhận kết nối sockSrvr.Listen();	
	// Xin phép kết nối ^{3,4} sockClient.Connect(strAddr,nPort);
// Chuẩn bị socket kết nối CSocket sockRecv; // Đồng ý kết nối ⁵ sockSrvr.Accept(sockRecv);	
// Truyền dữ liệu ⁶	// Truyền dữ liệu ⁶
// Kết thúc kết nối ⁷ . sockSrvr.Close();	sockClient.Close()

Cách thức hiện:

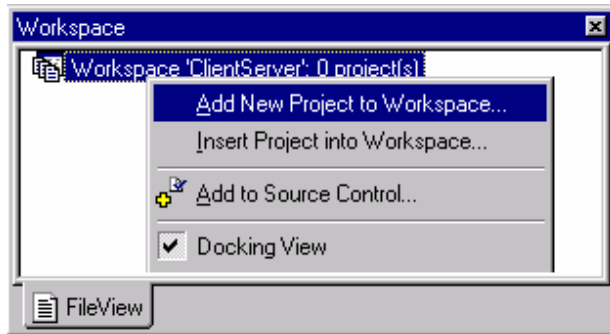
- Tạo tập dự án rỗng (Blank Workspace):

- Chọn mục File / New



- Thực hiện các ấn định như trên. Chọn **OK**.

- Bổ sung vào Workspace dự án **TcpServer**. Đây là dự án của ứng dụng server có nhiệm vụ chờ kết nối và cung cấp dữ liệu cho các client.
 - Trong màn hình Workspace, right-click trên mục Workspace.



- Chọn **Add New Project to Workspace**.
- Dùng Wizard tạo dự án **TcpServer** với giao diện chính là dialog, và có sử dụng windows socket. Cách thực hiện tương tự (13.4).

Thực hiện các công việc sau cho dự án TcpServer:

- Bổ sung lớp CEmpTcpWriter kế thừa CAsyncSocket của MFC. Lớp bổ sung này đảm nhận các chức năng truyền nội dung tập tin cho client được kết nối (OnSend), và chấm dứt kết nối khi đã truyền xong (Close). Cài đặt của lớp CEmpTcpWriter như sau:

- Khai báo kiểu cấu trúc cho vùng đệm chứa dữ liệu gửi:

```
const BUFFER_LEN = 1024; // Kích thước vùng chứa dữ liệu
typedef struct _tagBuf {
    int    length;           // Kích thước dữ liệu thực
    char  info[BUFFER_LEN]; // Vùng chứa dữ liệu
} BUFFER;
```

- Các thuộc tính protected:

```
BOOL    m_isBusy; // Trạng thái phục vụ.
CDialog* m_Parent; // Đối tượng dialog liên quan
```

- Hành vi tạo lập nhận tham số là con trỏ đối tượng dialog liên quan, và xác định trạng thái sẵn sàng:

```
CEmpTcpWriter::CEmpTcpWriter( CDialog* parent )
{
    m_Parent = parent; // Đối tượng dialog liên quan
    m_isBusy = FALSE;  // Sẵn sàng nhận dữ liệu
}
```

- Hành vi OnSend thực hiện gửi dữ liệu cho client liên kết:

```
void CEmpTcpWriter::OnSend( int nErrorCode )
{
    static    BOOL fileOK = TRUE;
```

```
static    BOOL fileReady = FALSE;
static    CFile file;
static    longtotal;
BUFFER buf;
m_isBusy = TRUE;           // Đã kết nối
buf.length = 0;           // Dữ liệu nhận thực sự
if ( fileOK && (!fileReady) ) {
    m_Parent->SetDlgItemText(IDC_INFO, "\nTransferring...");
    fileReady = TRUE;
    CString fileName;       // Tên tập tin được gửi
    // IDC_FILE : Số hiệu hộp nhập tên tập tin trên dialog
    m_Parent->GetDlgItemText(IDC_FILE, fileName);
    fileOK = file.Open(fileName, CFile::modeRead);
    total = 0;              // Chứa kích thước thông tin gửi
}
if (!fileOK ||
    ( buf.length = file.Read( buf.info, BUFFER_LEN ) )
    < BUFFER_LEN )
    fileReady = FALSE;      // Không có file hoặc đã gửi hết
Send( &buf, BUFFER_LEN + sizeof(int) );
total += buf.length;
if (fileReady) {
    Sleep( 300 );           // Thời gian nghỉ cho client
    AsyncSelect( FD_WRITE ); // Gửi tiếp
}
else {
    if ( fileOK ) file.Close();
    Close();
    m_isBusy = FALSE;       // Sẵn sàng nhận dữ liệu
    CString result;
    result.Format( "\nReady to connect to clients !\n"
                  "%d (bytes) transfered completed", total );
    m_Parent->SetDlgItemText( IDC_INFO, result );
}
}
```

- Bổ sung lớp CEmpTcpServer kế thừa CAsyncSocket đảm nhận các chức năng nhận kết nối (accept), và khởi tạo đối tượng gửi dữ liệu:

- Các thuộc tính protected của lớp CEmpTcpServer:

```
CEmpTcpWriter* m_pDoSendObject; // Đối tượng gửi dữ liệu
```

```
CDialog* m_Parent; // Dialog liên quan
```

- Hành vi tạo lập nhận tham số là con trỏ đối tượng dialog liên quan:

```
CEmpTcpServer::CEmpTcpServer( CDialog* parent )
{
    m_Parent = parent; // Con trỏ đối tượng dialog liên quan
    m_pDoSendObject = new CEmpTcpWriter( m_Parent );
}
```

- Hành vi hủy bỏ giải phóng đối tượng m_pDoSendObject:

```
CEmpTcpServer::~CEmpTcpServer( )
{
    delete m_pDoSendObject;
}
```

- Hành vi OnAccept chấp nhận kết nối nếu đang rồi:

```
void CEmpTcpServer::OnAccept( int nErrorCode )
{
    if ( m_pDoSendObject->m_isBusy )
        return ; // Đang bận phục vụ
    SOCKADDR_IN Addr;
    int Len = sizeof(SOCKADDR_IN);
    if (Accept(*m_pDoSendObject, (SOCKADDR*) &Addr, &Len))
        // Khởi động hoạt động gửi dữ liệu trên m_pDoSendObject
        m_pDoSendObject->AsyncSelect(FD_WRITE);
}
```

- Thực hiện các cài đặt bổ sung cho lớp đối tượng dialog CTcpServerDlg làm giao diện chính của ứng dụng server:

- Mở dialog resource, cài đặt các control sau:
 - *Hộp nhập tên tập tin* : Edit Số hiệu IDC_FILE
 - *Hộp hiển thị trạng thái hệ thống* : Static IDC_INFO
 - *Hộp hiển thị IP của host làm server* : Static IDC_HOSTIP
- Đối tượng thuộc tính protected làm nhiệm vụ truyền thông:

```
CEmpTcpServer* m_Tcp;
```

- Hành vi OnInitDialog của dialog khởi tạo thông số cho dialog và các thông số cần thiết cho hoạt động truyền thông với m_Tcp:

```
BOOL CTcpServerDlg::OnInitDialog()
{
    CDialog::OnInitDialog(); SetIcon( m_hIcon, TRUE );
    SetDlgItemText( IDC_FILE, "C:/AUTOEXEC.BAT" );
}
```

```
// Xác định địa chỉ IP của server host
char name[1024];
gethostname( name, 1024 );
PHOSTENT phost = gethostbyname( name );
SetDlgItemText( IDC_HOSTIP,
    inet_ntoa(*(in_addr*)phost->h_addr) );
m_Tcp = new CEmpTcpServer( this );
m_Tcp->Create( PORT_NO ); // Khởi tạo thông số
m_Tcp->Listen(); // Chờ kết nối
return TRUE;
}
```

- Hành vi OnDestroy thực hiện giải phóng đối tượng truyền thông

```
void CTcpServerDlg::OnDestroy()
{
    CDialog::OnDestroy();
    delete m_Tcp;
}
```

- Tạo dự án **TcpClient**. Thao tác tương tự dự án **TcpServer**.

Thực hiện các công việc sau đây với dự án TcpClient :

- Bổ sung lớp CEmpTcpReader kế thừa CAsyncSocket đảm nhận chức năng xin kết nối (Connect), và nhận dữ liệu (OnReceive) từ server:

- Thuộc tính public chứa trạng thái truyền thông:

```
BOOL m_isBusy; // = TRUE : Đang bận
```

- Thuộc tính protected chứa con trỏ đối tượng dialog liên quan:

```
CDialog* m_Parent;
```

- Hành vi tạo lập nhận tham số là đối tượng dialog liên quan:

```
CEmpTcpReader::CEmpTcpReader(CDialog* parent)
{
    m_isBusy = FALSE; // Sẵn sàng làm việc
    m_Parent = parent;
}
```

- Hành vi OnReceive nhận dữ liệu và lưu xuống tập tin:

```
void CEmpTcpReader::OnReceive(int nErrorCode)
{
    static BOOL fileOK = TRUE;
    static BOOL fileReady = FALSE;
    static CFile file;
    static long total;
```

```

BUFFER  buf;
Receive(&buf, BUFFER_LEN + sizeof(int));
if ( fileOK && (!fileReady) ) {
    m_Parent->SetDlgItemText(IDC_INFO, "\nDownloading...");
    fileReady = TRUE;
    CString fileName;          // Tên tập tin chứa dữ liệu nhận
    // IDC_FILE : Số hiệu hộp nhập tên tập tin trên dialog
    m_Parent->GetDlgItemText( IDC_FILE, fileName );
    fileOK = file.Open ( fileName,
                        CFile::modeWrite | CFile::modeCreate );
    total = 0;          // Kích thước dữ liệu nhận được
}
if ( fileOK && buf.length > 0 ) {
    file.Write( buf.info, buf.length );
    total += buf.length;
    if ( buf.length < BUFFER_LEN )
        fileReady = FALSE;    // Đã nhận đủ dữ liệu từ server
}
else
    fileReady = FALSE;
if (fileReady)
    CAsyncSocket::OnReceive(nErrorCode);    // Nhận tiếp
else {
    m_isBusy = FALSE;          // Trở lại trạng thái sẵn sàng @
    Close();
    if (fileOK) file.Close();
    CString result;
    result.Format(  "\nReady to connect to server !\n"
                    "%d (bytes) download completed.", total );
    m_Parent->SetDlgItemText(IDC_INFO, result);
}
}

```

- Thực hiện các cài đặt bổ sung cho lớp đối tượng dialog CTcpClientDlg làm giao diện chính của ứng dụng client:

- Mở dialog resource, cài đặt các control sau:
 - *Hộp nhập tên tập tin* : Edit IDC_FILE
 - *Hộp hiển thị trạng thái hệ thống* : Static IDC_INFO
 - *Hộp nhập địa chỉ IP của server* : IPControl IDC_HOSTIP

- *Nút lệnh download tập tin từ server* : Button IDOK
- Tạo thuộc tính biến *m_serverIP* cho hộp nhập IDC_HOSTIP.
- Đối tượng thuộc tính *protected* làm nhiệm vụ truyền thông:

```
CEmpTcpReader *m_readTcp;
```

- Hành vi *OnInitDialog* của dialog khởi tạo thông số cho dialog và các thông số cần thiết cho hoạt động truyền thông với *m_readTcp*:

```

BOOL CTcpClientDlg::OnInitDialog()
{
    CDialog::OnInitDialog();
    SetIcon(m_hIcon, TRUE);    // Set big icon
    SetIcon(m_hIcon, FALSE);   // Set small icon
    m_serverIP.SetWindowText( "127.0.0.1" );    // local host
    SetDlgItemText( IDC_FILE, "C:/Autoexec.000" );
    m_readTcp = new CEmpTcpReader( this );
    return TRUE;
}

```

- Hành vi *OnDestroy* giải phóng đối tượng truyền thông:

```

void CTcpClientDlg::OnDestroy()
{
    CDialog::OnDestroy();
    delete m_readTcp;
}

```

- Hành vi *OnOK* ứng với nút chọn download bắt đầu nhận dữ liệu:

```

void CTcpClientDlg::OnOK()
{
    CString s;
    m_serverIP.GetWindowText(s);
    if ( m_readTcp->m_isBusy || s == "0.0.0.0" ) return;
    m_readTcp->m_isBusy = TRUE;    // Trạng thái bận @
    m_readTcp->Create();
    m_readTcp->Connect(s, PORT_NO);
}

```

- Biên dịch các dự án và thử nghiệm trên một hoặc nhiều máy.
- ☞ Chọn mục : **Project / Set Active Project** để ấn định dự án làm việc trong tập dự án (Workspace) như trường hợp nói trên.
- ☞ Sử dụng CSocket và CSocketFile cho tác vụ truyền tập tin nói trên.

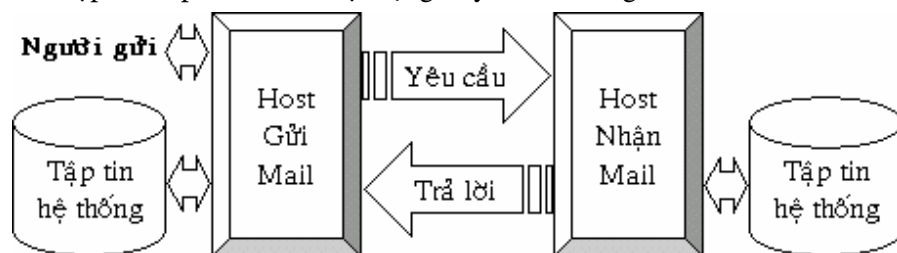
Lớp CSocket:

CSocket là lớp kế thừa CAsyncSocket mà các hành vi gửi và nhận dữ liệu của nó tự động thực hiện cơ chế chờ cho tác vụ trước đó hoàn tất. Như vậy, sẽ không gặp lỗi tác vụ nghẽn (WSAEWOULDBLOCK) khi thực hiện truyền dữ liệu với CSocket. Điều này rất quan trọng với các ứng dụng truyền tải dữ liệu có kích thước lớn như ứng dụng (13.5).

Trong phần nội dung tiếp theo, CSocket sẽ được sử dụng để thay thế cho CAsyncSocket trong các ví dụ minh họa.

13.6 TCP VỚI SMTP (SIMPLE MAIL TRANSFER PROTOCOL):

SMTP là giao thức đơn giản và hiệu quả cho việc truyền tải e-mail từ host gửi đến host nhận mail; ví dụ từ một mail client đến một mail server. Quá trình truyền tải mail được thực hiện trên cơ sở thiết lập một kênh liên lạc hai chiều giữa hai host. Host gửi lần lượt phát các chỉ thị kết nối và gửi dữ liệu. Host nhận tiếp nhận dữ liệu và trả lời để host gửi có cơ sở xác định thao tác thích hợp kế tiếp. Mô hình hoạt động truyền tải mail giữa hai host như sau:



- Khi nhận được yêu cầu của người dùng, host gửi phát tín hiệu đến host nhận để xin kết nối.
- Mỗi khi nhận được một xác nhận đồng ý của host nhận, host gửi tiếp tục chuyển phần nội dung kế tiếp của mail.
- Quá trình trên kết thúc khi mail đã gửi xong hoặc có sự trả lời với nội dung từ chối từ phía host nhận.

13.6.1 Quy ước giao tác giữa ứng dụng gửi mail và ứng dụng nhận mail:

- Thiết lập liên kết: Đầu tiên, ứng dụng gửi mail trên host gửi sử dụng giao thức TCP kết nối với ứng dụng nhận mail trên host nhận thông qua địa chỉ IP của host nhận và số hiệu port của ứng dụng nhận mail.
 - Gửi: Mở socket và liên lạc với ứng dụng nhận mail (port = 25).
 - Nhận: 220 <Domain> <ServerName> ready

☞ Sau khi nhận được tín hiệu trả lời như trên từ ứng dụng nhận mail, ứng dụng gửi mail tiếp tục thực hiện các tác vụ sau đây theo thứ tự.

- ‘Trình diện’ với ứng dụng nhận mail:

- Gửi: HELO <SP> <domain> <CRLF>

Trong đó: SP : Ký tự khoảng trắng
CRLF : \r\n // return & newline in C

- Nhận: OK : 250 BBN-UNIX.ARPA

- ERROR : Số hiệu lỗi (kết nối bị từ chối).

Nếu nhận được trả lời “250 <Server Name>” từ ứng dụng nhận mail, ứng dụng gửi mail có thể tiếp tục các công việc kế tiếp.

- Đăng ký địa chỉ người gửi mail:

- Gửi: MAIL <SP> FROM:<địa_chỉ_nơi_gửi> <CRLF>

- Nhận: OK : 250 OK

- ERROR : Đăng ký không được chấp nhận → kết thúc.

- Đăng ký địa chỉ người nhận mail: Nếu mail được gửi cho nhiều địa chỉ thì thực hiện thao tác này nhiều lần cho các địa chỉ nhận đó.

- Gửi: RCPT <SP> TO:<địa_chỉ_nơi_nhận> <CRLF>

- Nhận: OK : 250 OK

- ERROR : 550 Failure Info

- Chuẩn bị truyền nội dung của mail:

- Gửi: DATA <CRLF>

- Nhận: OK : 354 Intermediate reply

- ERROR : 550 Failure

- Truyền nội dung của mail: Có thể lặp nhiều lần thao tác sau đây tùy theo số đoạn văn bản trong nội dung của mail.

- Gửi: Nội dung đoạn văn bản thứ (i) của mail

- Nhận: Không có trả lời của host nhận.

- Chấm dứt truyền nội dung của mail:

- Gửi: <CRLF> . <CRLF>

- Nhận: 250 OK

- Kết thúc kết nối với host nhận:

- Gửi: QUIT <CRLF>

- Nhận: 221 Terminated Info

Chấm dứt liên kết giữa hai ứng dụng gửi mail và nhận mail.

13.6.2 Thiết kế ứng dụng gửi mail:

Trong phần này, ta xây dựng ứng dụng sử dụng giao thức TCP để kết nối và gửi mail lên một mail server. Nội dung mail và các ấn định liên quan được thể hiện trên dialog giao diện chính của ứng dụng.

- Dùng MFC Wizard tạo dự án với cửa sổ giao diện chính là dialog, có sử dụng winsock (xem mục 13.4). Đặt tên cho dự án là **SMTP**.
- Thực hiện các cài đặt sau đây cho dialog giao diện (CSMTPDlg):
 - Mở dialog resource, cài đặt các control sau:
 - *Hộp nhập địa chỉ mail server* : Edit IDC_SERVER_IP
 - *Hộp nhập số hiệu port* : Edit IDC_SERVER_PORT
 - *Hộp nhập địa chỉ người nhận* : Edit IDC_MAIL_TO
 - *Hộp nhập tiêu đề của mail* : Edit IDC_MAIL_SUBJECT
 - *Hộp soạn thảo nội dung mail* : Edit IDC_MAIL
 - *Nút lệnh thực hiện gửi mail* : Button IDOK
 - Hành vi OnInitDialog thực hiện các khởi tạo cần thiết:

```

BOOL CSMTPDlg::OnInitDialog()
{
    CDialog::OnInitDialog();
    SetIcon(m_hIcon, TRUE);
    SetIcon(m_hIcon, FALSE);

    SetDlgItemText(IDC_SERVER_IP, "omail.hcmueco.edu.vn");
    SetDlgItemText(IDC_SERVER_PORT, "25");
    SetDlgItemText(IDC_MAIL_TO, "emp@hcmueco.edu.vn");
    SetDlgItemText(IDC_MAIL_SUBJECT,
                    "A Lesson of Mr.Emp");
    SetDlgItemText(IDC_MAIL, "Hello Mr.Emp!\r\n\r\n"
                             "It's nice to meet You.\r\n\r\n"
                             "Thanks for your lessons.");

    return TRUE;
}

```

- Hành vi kiểm tra lỗi trên giá trị nhận được từ ứng dụng nhận mail:
Cài đặt hành vi protected: GetErrorCode cho lớp CSMTPDlg:

```

UINT CSMTPDlg::GetErrorCode(char *sReply)
{
    UINT rs = 0;
    sscanf(sReply, "%d", &rs);
    return rs;    // Trả về số hiệu lỗi ghi trong dữ liệu nhận được
}

```

- Hành vi OnOK ứng với nút chọn IDOK thực hiện gửi mail:

```

void CSMTPDlg::OnOK()
{
    char    s[1025];
    UINT    port;
    CSocket m_sock;

    GetDlgItemText(IDC_SERVER_IP, s, 1024);
    port = GetDlgItemInt(IDC_SERVER_PORT);
    m_sock.Create();
    if (!m_sock.Connect(s, port)) return;
    if (!m_sock.Receive(s, 1024) || GetErrorCode(s) != 220)
        return;

    strcpy(s, "HELO Mr.Emp\r\n");
    m_sock.Send(&s, strlen(s));
    if (!m_sock.Receive(s, 1024) || GetErrorCode(s) != 250)
        return;

    // Địa chỉ người gửi
    strcpy(s, "MAIL FROM:Mr.Emp\r\n");
    m_sock.Send(&s, strlen(s));
    if (!m_sock.Receive(s, 1024) || GetErrorCode(s) != 250)
        return;

    // Địa chỉ người nhận
    strcpy(s, "RCPT TO:");
    GetDlgItemText(IDC_MAIL_TO, s + strlen(s), 1024);
    strcpy(s + strlen(s), "\r\n");
    m_sock.Send(&s, strlen(s));
    if (!m_sock.Receive(s, 1024) || GetErrorCode(s) == 550)
        return;

    // Chuẩn bị truyền nội dung mail
    strcpy(s, "DATA\r\n");
    m_sock.Send(&s, strlen(s));
    if (!m_sock.Receive(s, 1024) || GetErrorCode(s) != 354)
        return;

    // Chủ đề mail
    strcpy(s, "Subject:");
    GetDlgItemText(IDC_MAIL_SUBJECT, s + strlen(s), 1024);
    strcpy(s + strlen(s), "\r\n");
    m_sock.Send(&s, strlen(s));

    // Nội dung mail

```



```

GetDlgItemText(IDC_MAIL, s, 1024);
sprintf(s, "%s%s", s, "\r\n");
m_sock.Send(&s, strlen(s));

// Kết thúc
strcpy(s, "\r\n.\r\n");
m_sock.Send(&s, strlen(s));

strcpy(s, "QUIT\r\n");
m_sock.Send(&s, strlen(s));

m_sock.Close();
MessageBox("The Mail was sent OK !", "Send mail");
}

```

13.7 TCP VỚI POP3 (Post Office Protocol - Version 3):

POP3 là giao thức cho phép mail client host kết nối với mail server host để lấy thông tin về hộp mail của người dùng trên mail server. Hoạt động truy xuất này nhằm tải mail của người dùng về client host và xóa mail ấy khỏi hộp mail của họ trên mail server. Tương tự SMTP, quá trình tải mail từ server của POP3 được thực hiện trên cơ sở thiết lập một kênh liên lạc hai chiều giữa client và server. Client gửi lần lượt gửi các chỉ thị kết nối và nhận dữ liệu. Mail server tiếp nhận yêu cầu của client và trả lời hoặc gửi dữ liệu.

13.7.1 Quy ước giao tác giữa hai ứng dụng mail client và mail server:

- Thiết lập liên kết: Đầu tiên, ứng dụng mail client sử dụng giao thức TCP kết nối với ứng dụng mail server thông qua địa chỉ IP của mail server host và số hiệu port của ứng dụng mail server.
 - Gửi: Mở socket và liên lạc với ứng dụng mail server (port=110).
 - Nhận: +OK ok_message
-ERR error_message
- ☞ Sau khi nhận được tín hiệu trả lời 'OK' từ ứng dụng mail server, ứng dụng mail client tiếp tục thực hiện các tác vụ sau theo thứ tự.

Đăng ký tài khoản truy cập mail server:

- Khai báo user name:
 - Gửi: USER <SP> <tên_người_sử_dụng> <CRLF>
 - Nhận: +OK ok_message
-ERR error_message → kết thúc.
- Khai báo password:
 - Gửi: PASS <SP> <nội_dung_password> <CRLF>
 - Nhận: +OK ok_message

-ERR error_message → kết thúc.

Sau khi đăng nhập thành công, có thể tùy ý thực hiện các công việc sau:

- Lấy thông tin về hộp mail:
 - Gửi: STAT <CRLF>
 - Nhận: +OK <SP> <number_of_mail> <SP> <mail_total_size>
-ERR error_message
- Lấy thông tin chi tiết của các mail trong hộp mail:
 - Gửi: LIST <CRLF>
 - Nhận: +OK <SP> Mailbox scan listing follows
-ERR error_message
- Đọc mail:
 - Gửi: RETR <SP> <số_hiệu_mail_được_đọc> <CRLF>
 - Nhận: +OK kích_thước_message octets
... message do mail server gửi về, kết thúc bởi dấu '.'
-ERR error_message
- Xóa mail:
 - Gửi: DELE <SP> <số_hiệu_mail_xóa> <CRLF>
 - Nhận: +OK message <n> deleted
-ERR error_message
- Chấm dứt liên kết với mail server:
 - Gửi: QUIT <CRLF>
 - Nhận: + OK goodbye_message
- ERR error_message

13.7.2 Thiết kế ứng dụng nhận mail:

Trong phần này, ta xây dựng ứng dụng sử dụng giao thức TCP để kết nối và lấy mail từ một POP mail server. Thông số liên quan mail server và thông tin nhận được sẽ được thể hiện trên dialog giao diện chính của ứng dụng.

- Dùng MFC Wizard tạo dự án với cửa sổ giao diện chính là dialog, có sử dụng winsock (xem mục 13.4). Đặt tên cho dự án là **POP3**.
- Thực hiện các cài đặt sau cho dialog giao diện chính (CPOP3Dlg):
 - Mở dialog resource, cài đặt các control sau:

- Hộp nhập địa chỉ mail server	: Edit	IDC_SERVER_IP
- Hộp nhập số hiệu port	: Edit	IDC_SERVER_PORT
- Hộp nhập tên người truy cập	: Edit	IDC_USER_NAME
- Hộp nhập password truy cập	: Edit	IDC_USER_PASSWORD

- Hộp nhập số hiệu mail nhận về : Edit IDC_MAIL_NO
- Hộp hiển thị nội dung mail : Edit IDC_MAIL
- Nút lệnh lấy thông tin về mail : Button IDOK
- Nút lệnh thực hiện nhận mail : Button IDC_READ_MAIL
- Hành vi OnOK ứng với nút **OK** thực hiện lấy thông tin về hộp mail:

```
void CPOP3Dlg::OnOK()
{
    char    s[1025];
    UINT    port;
    CSocket m_sock;

    GetDlgItemText(IDC_SERVER_IP, s, 1024);
    port = GetDlgItemInt(IDC_SERVER_PORT);
    m_sock.Create();
    if (!m_sock.Connect(s, port)) return;
    if ( !m_sock.Receive(s, 1024) || !isOK(s) ) return;

    strcpy(s, "USER ");           // Đăng nhập
    GetDlgItemText(IDC_USER_NAME, s + strlen(s), 1024);
    strcat(s, "\r\n");
    m_sock.Send(&s, strlen(s));
    if ( !m_sock.Receive(s, 1024) || !isOK(s) ) return;

    strcpy(s, "PASS ");           // Đăng ký password
    GetDlgItemText(IDC_USER_PASSWORD, s+strlen(s), 1024);
    strcat(s, "\r\n");
    m_sock.Send(&s, strlen(s));
    if ( !m_sock.Receive(s, 1024) || !isOK(s) ) return;

    strcpy(s, "LIST\r\n");        // Lấy thông tin hộp mail
    m_sock.Send(&s, strlen(s));
    if ( (port = m_sock.Receive(s, 1024)) < 1 || !isOK(s) )
        return;

    s[port] = '\0';
    SetDlgItemText(IDC_MAIL, s+4); // Bỏ qua +OK<SP>

    // Kết thúc...
    strcpy(s, "QUIT\r\n");
    m_sock.Send(&s, strlen(s));
    m_sock.Close();
}
```

- Hành vi protected : isOK kiểm tra trả lời của mail server trên dữ liệu nhận được sau mỗi tác vụ liên lạc:

```
BOOL CPOP3Dlg::isOK(char *result)
{
    return ( result[0] == '+' &&
            result[1] == 'O' &&
            result[2] == 'K' );    // +OK : Thành công
}
```

- Hành vi OnReadMail (nút chọn IDC_READ_MAIL) tải mail về:

```
void CPOP3Dlg::OnReadMail()
{
    char    s[10241];
    UINT    port;
    CSocket m_sock;

    GetDlgItemText(IDC_SERVER_IP, s, 1024);
    port = GetDlgItemInt(IDC_SERVER_PORT);
    m_sock.Create(); if (!m_sock.Connect(s, port)) return;
    if ( !m_sock.Receive(s, 1024) || !isOK(s) ) return;

    strcpy(s, "USER ");           // Đăng nhập
    GetDlgItemText(IDC_USER_NAME, s + strlen(s), 1024);
    strcat(s, "\r\n");
    m_sock.Send(&s, strlen(s));
    if ( !m_sock.Receive(s, 1024) || !isOK(s) ) return;

    strcpy(s, "PASS ");           // Hợp lệ ?
    GetDlgItemText(IDC_USER_PASSWORD, s+strlen(s), 1024);
    strcat(s, "\r\n");
    m_sock.Send(&s, strlen(s));
    if ( !m_sock.Receive(s, 1024) || !isOK(s) ) return;

    strcpy(s, "RETR ");           // Tải mail về
    GetDlgItemText(IDC_MAIL_NO, s + strlen(s), 1024);
    strcat(s, "\r\n");
    m_sock.Send(&s, strlen(s));
    if ( (port = m_sock.Receive(s, 10240)) == 0 || !isOK(s) )
        return;

    s[port] = '\0'; SetDlgItemText(IDC_MAIL, strstr(s, "From"));

    strcpy(s, "QUIT\r\n");        // Kết thúc
    m_sock.Send(&s, strlen(s)); m_sock.Close();
}
```

13.8 TCP VỚI HTTP & FTP:

HTTP và FTP là các giao thức tầng ứng dụng được xây dựng dựa trên giao thức TCP ở tầng truyền tải (mô hình DARPA).

- HTTP: Giao thức phục vụ mô hình ứng dụng Client/Server, dùng phổ biến trong việc truyền tải các nội dung HTML (www) giữa client và server. Trong đó ứng dụng client kết nối với ứng dụng server thông qua cổng TCP xác định, thông thường là 80.
- FTP: Giao thức truyền tải tập tin. Ứng dụng client kết nối với ứng dụng server thông qua hai cổng TCP; một dùng điều khiển truyền, một dùng truyền dữ liệu; thông thường là 20 và 21.
- ☞ MFC hỗ trợ rất mạnh cho việc khai thác các giao thức nói trên thông qua các lớp đối tượng trong thư viện **AfxInet.h**.

13.8.1 CInternetSession:

CInternetSession là lớp đối tượng quản lý một giao tác xác định với mạng internet, kể cả việc thực hiện kết nối với proxy server.

Mỗi giao tác internet cho phép thực hiện các tác vụ liên quan đến tập tin trên các internet server thông qua các dịch vụ đặc biệt như FTP, HTTP.

- **CInternetSession** (
LPCTSTR *pstrAgent* = NULL,
DWORD *dwContext* = 1,
DWORD *dwAccessType* =
INTERNET_OPEN_TYPE_PRECONFIG,
LPCTSTR *pstrProxyName* = NULL,
LPCTSTR *pstrProxyBypass* = NULL,
DWORD *dwFlags* = 0
); Hành vi tạo lập và khởi tạo thông số cho đối tượng giao tác internet.
pstrAgent: Tên ứng dụng. Một số ứng dụng dùng tên làm giá trị nhận diện đối với server.
dwContext: Số hiệu nhận diện diễn tiến của tác vụ.
dwAccessType: Kiểu truy xuất.
INTERNET_OPEN_TYPE_PRECONFIG :
Theo ấn định mặc nhiên của hệ thống.
INTERNET_OPEN_TYPE_DIRECT :
Trực tiếp không thông qua proxy.
INTERNET_OPEN_TYPE_PROXY :
Thông qua proxy.

- pstrProxyName*: Proxy. Sử dụng cho truy xuất thông qua proxy.
- pstrProxyBypass*: Danh sách địa chỉ các server được truy xuất trực tiếp (dùng cho ấn định truy xuất qua proxy).
- dwFlags*: Thông số về chế độ tối ưu trong truy xuất server.
INTERNET_FLAG_DONT_CACHE :
Không sử dụng cache ở tất cả các host liên quan đến các tác vụ tập tin của giao tác.
INTERNET_FLAG_OFFLINE :
Sử dụng cache cho các tác vụ tập tin.

- CFtpConnection* **GetFtpConnection** (
LPCTSTR *pstrServer*, // Địa chỉ FTP server
LPCTSTR *pstrUserName* = NULL, // Tên đăng nhập
LPCTSTR *pstrPassword* = NULL, // password hợp lệ
INTERNET_PORT *nPort* = // số hiệu port kết nối */
INTERNET_INVALID_PORT_NUMBER,
BOOL *bPassive* = FALSE // Cơ chế kết nối
); Trả về con trỏ đối tượng quản lý liên kết với FTP server.
bPassive: Qui định cơ chế chọn port cho kết nối.
- TRUE: Do người dùng chọn.
- FALSE: Do FTP Server chọn (là cơ chế mặc nhiên).
- CHttpConnection* **GetHttpConnection** (
LPCTSTR *pstrServer*, // Địa chỉ HTTP server
INTERNET_PORT *nPort* = // số hiệu port */
INTERNET_INVALID_PORT_NUMBER,
LPCTSTR *pstrUserName* = NULL, // Tên đăng nhập
LPCTSTR *pstrPassword* = NULL // password hợp lệ
); Trả về con trỏ đối tượng quản lý liên kết với HTTP server.

13.8.2 CInternetFile:

CInternetFile là lớp đối tượng quản lý một tập tin trên mạng internet.

- **CInternetFile**(); Tạo lập đối tượng quản lý tập tin.
- virtual LONG **Seek** (
LONG *lOffset*, // Cự ly dời tương đối
UINT *nFrom* // Vị trí làm mốc.
); Ấn định vị trí con trỏ (đầu đọc/ghi dữ liệu) trong tập tin.
Các giá trị dùng cho *nFrom* có thể là:
- CFile::current : Tính từ vị trí hiện hành của con trỏ tập tin.

- CFile::begin : Tính từ đầu tập tin.
- CFile::end : Tính từ cuối tập tin.
- virtual UINT Read (
 - void* lpBuf, // Địa chỉ vùng đệm chứa dữ liệu đọc
 - UINT nCount // Kích thước dữ liệu đọc vào
); Đọc dữ liệu từ tập tin.
- virtual void Write (
 - const void* lpBuf, // Địa chỉ vùng đệm chứa dữ liệu ghi ra
 - UINT nCount // Kích thước dữ liệu được ghi
); Ghi dữ liệu ra tập tin.
- virtual BOOL ReadString (
 - CString& rString // Biến chứa chuỗi đọc vào
); Đọc một chuỗi từ tập tin.
- virtual void WriteString (
 - LPCTSTR pstr // Biến chứa chuỗi được ghi ra
); Ghi một chuỗi ra tập tin.
- virtual void Close(); Đóng tập tin, chấm dứt mọi tác vụ liên quan.

13.8.3 CFtpConnection:

CFtpConnection là lớp đối tượng quản lý một liên kết với FTP server. Đối tượng này có thể nhận được từ đối tượng quản lý giao tác với mạng internet thông qua hành vi GetFtpConnection với thông số thích hợp (13.8.1). Các hành vi đặc trưng của lớp đối tượng CFtpConnection như sau:

- BOOL SetCurrentDirectory (
 - LPCTSTR pstrDirName // Đường dẫn của thư mục.
); Ấn định thư mục làm việc mặc nhiên trên FTP server.
- BOOL GetCurrentDirectory (
 - CString& strDirName // Biến chứa kết quả
); Lấy đường dẫn của thư mục làm việc trên FTP server.
- BOOL RemoveDirectory (
 - LPCTSTR pstrDirName // Đường dẫn thư mục
); Xóa thư mục trên FTP server.
- BOOL CreateDirectory (
 - LPCTSTR pstrDirName // Đường dẫn và tên thư mục
); Tạo mới thư mục trên FTP server.
- BOOL Rename (
 - LPCTSTR pstrExisting, // Đường dẫn và tên cũ của thư mục

- LPCTSTR pstrNew // Tên mới của thư mục
); Đổi tên thư mục trên FTP server.
- BOOL Remove (
 - LPCTSTR pstrFileName // Đường dẫn và tên tập tin
); Xóa tập tin trên FTP server.
- CInternetFile* OpenFile (
 - LPCTSTR pstrFileName, // Đường dẫn và tên tập tin được mở
 - DWORD dwAccess = GENERIC_READ,
 - DWORD dwFlags = FTP_TRANSFER_TYPE_BINARY,
 - DWORD dwContext = 1
); Mở tập tin trên FTP server. Trả về con trỏ đối tượng CInternetFile quản lý tập tin được mở.

dwAccess : Chế độ mở tập tin, chọn một trong các chế độ mở sau:

 - GENERIC_WRITE : Mở để ghi.
 - GENERIC_READ : Mở để đọc.

dwFlags : Kiểu nội dung tập tin.

 - FTP_TRANSFER_TYPE_ASCII : Mã ASCII
 - FTP_TRANSFER_TYPE_BINARY : Mã nhị phân
- BOOL PutFile (
 - LPCTSTR pstrLocalFile, // Đường dẫn và tên tập tin ở client
 - LPCTSTR pstrRemoteFile, // Đường dẫn và tên tập tin ở server
 - DWORD dwFlags = // Kiểu nội dung tập tin */
 - FTP_TRANSFER_TYPE_BINARY,
 - DWORD dwContext = 1 // Số hiệu nhận diện tác vụ
); Tải một tập tin từ host làm việc lên FTP server (upload).
- BOOL GetFile (
 - LPCTSTR pstrRemoteFile, // Đường dẫn tên tập tin trên server
 - LPCTSTR pstrLocalFile, // Đường dẫn tên tập tin trên client
 - BOOL bFailIfExists = TRUE, // Chế độ xử lý trùng tên tập tin.
 - DWORD dwAttributes = FILE_ATTRIBUTE_NORMAL,
 - DWORD dwFlags = // Kiểu nội dung tập tin */
 - FTP_TRANSFER_TYPE_BINARY,
 - DWORD dwContext = 1
); Tải một tập tin từ FTP server về host làm việc.

bFailIfExists : Qui định cách thực hiện khi tên dùng cho tập tin ghi trùng với tên một tập tin đã có trên host làm việc.

TRUE → Thông báo lỗi.

FALSE → Tự động ghi chồng lên nội dung đã có.

dwAttributes : Thuộc tính tập tin. Có thể kết hợp các giá trị sau:

FILE_ATTRIBUTE_ARCHIVE : Tập tin được cập nhật.

FILE_ATTRIBUTE_DIRECTORY : Thư mục.

FILE_ATTRIBUTE_NORMAL : Tập tin bình thường.

FILE_ATTRIBUTE_HIDDEN : Tập tin ẩn mật.

FILE_ATTRIBUTE_READONLY : Tập tin chỉ đọc.

FILE_ATTRIBUTE_SYSTEM : Tập tin hệ thống.

FILE_ATTRIBUTE_TEMPORARY : Tập tin sử dụng tạm thời.

- virtual void **Close**(); Kết thúc liên kết với FTP server.

13.8.4 **CFtpFindFile**:

CFtpFindFile là lớp đối tượng quản lý một công cụ tìm kiếm tập tin trên FTP server. Các hành vi đặc trưng của lớp như sau:

- **CFtpFileFind**(CFtpConnection* *pConnection*); Tạo lập và khởi tạo thông số cho đối tượng tìm kiếm tập tin.

- virtual BOOL **FindFile** (LPCTSTR *pstrName* = NULL, // Dạng tên tập tin cần tìm
DWORD *dwFlags* = INTERNET_FLAG_RELOAD
); Khởi động quá trình tìm kiếm tập tin. Trả về giá trị TRUE nếu khởi điểm tìm kiếm thành công.

Có thể sử dụng ?, * để ẩn định dạng tên của tập tin cần tìm.

dwFlags = INTERNET_FLAG_DONT_CACHE không sử dụng cache.

- virtual BOOL **FindNextFile**(); Tìm tập tin kế tiếp thỏa điều kiện. Trả về giá trị TRUE nếu tác vụ tìm kiếm thành công, nếu ngược lại thì xem như đã tìm hết các tập tin thỏa điều kiện tìm kiếm.

- CString **GetFileURL**(); Trả về chuỗi đường dẫn của tập tin vừa tìm được theo yêu cầu.

Đoạn chương trình sau thực hiện tìm kiếm các tập tin với tên có phần mở rộng là 'txt' trên FTP server:

```
CInternetSession sess(_T("FTP - Example")); // Quản lý giao tác
CFtpConnection* pConnect = NULL; // Liên kết với server
try {
    pConnect = sess.GetFtpConnection("ftp.hcmueco.edu.vn");
    CFtpFileFind TimFile ( pConnect ); // Đối tượng tìm kiếm
```

```
BOOL TimThay = TimFile.FindFile(_T("*.TXT"));
while (TimThay) {
    TimThay = TimFile.FindNextFile();
    ... // Xử lý tập tin tìm thấy
}
}
catch (CInternetException* pEx) {
    ... // Có lỗi
}
if (pConnect != NULL) pConnect->Close();
delete pConnect;
```

13.8.5 **CHttpConnection**:

CHttpConnection là lớp đối tượng quản lý một liên kết với HTTP server. Đối tượng này có thể nhận được từ đối tượng quản lý giao tác với mạng internet thông qua hành vi GetHttpConnection với thông số thích hợp (13.8.1).

Lớp CHttpConnection có hành vi đặc trưng sau:

- CHttpFile* **OpenRequest** (int *nVerb*,
LPCTSTR *pstrObjectName*,
LPCTSTR *pstrReferer* = NULL,
DWORD *dwContext* = 1,
LPCTSTR* *pstrAcceptTypes* = NULL,
LPCTSTR *pstrVersion* = NULL,
DWORD *dwFlags* = INTERNET_FLAG_EXISTING_CONNECT
); Trả về con trỏ đối tượng CHttpFile quản lý tập tin dữ liệu cần truy xuất trên HTTP server.

nVerb: Ấn định loại tác vụ truy xuất đối với tập tin dữ liệu.

CHttpConnection::HTTP_VERB_POST : Ghi dữ liệu

CHttpConnection::HTTP_VERB_GET : Đọc dữ liệu

CHttpConnection::HTTP_VERB_DELETE : Xóa tập tin dữ liệu.

pstrObjectName : Đường dẫn và tên tập tin dữ liệu trên HTTP server.

pstrReferer : Địa chỉ (root) của tập tin dữ liệu.

dwContext : Số hiệu nhận diện tác vụ truy xuất.

pstrAcceptTypes : Chứa các kiểu dữ liệu mà client mong muốn khai thác trong nội dung của tập tin dữ liệu được truy xuất. Ví dụ: "text/*" = Chỉ lấy nội dung text, không nhận dữ liệu là hình ảnh, mã,...

pstrVersion: Giao thức sử dụng, mặc nhiên là "HTTP/1.0"
dwFlags: Tối ưu trong truy xuất
 INTERNET_FLAG_RELOAD : Không đọc từ cache.
 INTERNET_FLAG_DONT_CACHE : Không để lại cache.

13.8.6 CHttpFile:

CHttpFile là lớp đối tượng quản lý một tập tin dữ liệu độc lập đang được truy xuất trên HTTP server.

- **BOOL QueryInfo** (
 - DWORD *dwInfoLevel*, // Thông số truy vấn
 - LPVOID *lpvBuffer*, // Vùng đệm nhận kết quả
 - LPDWORD *lpdwBufferLength*, // Chiều dài vùng đệm
 - LPDWORD *lpdwIndex* = NULL
); Truy vấn thông tin liên quan đến tập tin dữ liệu. Trả về giá trị TRUE nếu tác vụ truy vấn thành công. Thông số truy vấn có thể là:
 - HTTP_QUERY_FLAG_REQUEST_HEADERS : Thông tin header.
 - HTTP_QUERY_LAST_MODIFIED : Ngày cập nhật cuối cùng.
 - **BOOL AddRequestHeaders** (
 - CString& *str*, // Nội dung thông số yêu cầu
 - DWORD *dwFlags* = HTTP_ADDREQ_FLAG_ADD_IF_NEW
); Cập nhật một yêu cầu mới vào danh sách các yêu cầu đối với tập tin dữ liệu. Các yêu cầu này sẽ được gửi lên HTTP server thực hiện. Nội dung mỗi yêu cầu phải kết thúc bằng <CRLF>. Xem (RFC2616):
 - **Accept** : Đề nghị kiểu dữ liệu truy xuất trong tập tin.
 Cú pháp: "Accept: (media-range [accept-params],)* "
 Ví dụ: "Accept: audio/*", "Accept: text/plain"
 - **User-Agent** : Tên, dùng nhận diện ứng dụng yêu cầu.
 Cú pháp: "User-Agent" ":" Name(product | comment)*" "
 Ví dụ: "User-Agent: Mr.EMP"
 - **Content-Type** : Kiểu nội dung tập tin dữ liệu.
 Cú pháp: "Content-Type: media-type*" "
 Ví dụ: "Content-Type: text/html; charset=ISO-8859-4"
- dwFlags*: Thông số ấn định cơ chế cập nhật:
- HTTP_ADDREQ_FLAG_COALESCE : Trộn các nội dung của cùng một loại yêu cầu:
- "Accept: text/*" và "Accept: audio/*" → "Accept: text/*, audio/*"

HTTP_ADDREQ_FLAG_REPLACE : Thay thế nội dung trước bởi nội dung sau có cùng loại yêu cầu.

"Accept: text/*" và "Accept: audio/*" → "Accept: audio/*"

HTTP_ADDREQ_FLAG_ADD : Như _REPLACE và tự động bổ sung nếu loại yêu cầu mới là chưa có.

- **BOOL SendRequest** (
 - CString& *strHeaders*, // Nội dung yêu cầu.
 - LPVOID *lpOptional* = NULL, // Các diễn giải của client (không bắt buộc)
 - DWORD *dwOptionalLen* = 0
); Gửi yêu cầu liên quan tập tin dữ liệu lên HTTP server. Nếu không chỉ ra nội dung yêu cầu thì mặc nhiên dùng yêu cầu ấn định bởi hành vi AddRequestHeaders.
- **BOOL SendRequestEx** (
 - LPINTERNET_BUFFERS *lpBuffIn*, // Kích thước dữ liệu
 - LPINTERNET_BUFFERS *lpBuffOut*, // Vùng đệm chứa dữ liệu
 - DWORD *dwFlags* = HSR_INITIATE,
 - DWORD *dwContext* = 1
); Ghi thông tin lên tập tin dữ liệu trên HTTP server.
- **BOOL EndRequest** (
 - DWORD *dwFlags* = 0,
 - LPINTERNET_BUFFERS *lpBuffIn* = NULL,
 - DWORD *dwContext* = 1
); Ngừng tác vụ ghi trên tập tin dữ liệu của HTTP server.
- virtual CString **GetFileURL**(); Trả về đường dẫn và tên của tập tin dữ liệu trên HTTP server.
- virtual void **Close**(); Kết thúc truy xuất tập tin dữ liệu.

Ví dụ: Đoạn chương trình sau thực hiện kết nối với HTTP server. Đọc và hiển thị nội dung của tập tin /index.html trên server này.

```

CInternetSession session("HTTP - Example");
CHttpFile* file=NULL;
CString strServer = "www.hcmueco.edu.vn";
CString strObject("/index.html");
INTERNET_PORT port = INTERNET_DEFAULT_HTTP_PORT;
CString strHeaders (
    "Accept: text/*\r\n"
    "User-Agent: Mr.Emp\r\n"
    "Content-Type: application/x-www-form-urlencoded\r\n");
CHttpConnection* server = NULL;
try {
    server=session.GetHttpConnection(strServer, port);
    file=server->OpenRequest (
        CHttpConnection::HTTP_VERB_POST,strObject );
}
catch (CInternetException* pEx) {
    ... // Có lỗi
}
if (server==NULL) return;
if (file!=NULL) {
    file->AddRequestHeaders(strHeaders);
    file->SendRequest();           // Xác lập yêu cầu truy xuất
    CString line;                 // Sử dụng cho đọc chuỗi
    while (file->ReadString(line)) {
        ... // Xử lý nội dung đọc được
    }
    file->Close();
    delete file;
}
server->Close();
delete server;

```

13.8.7 Thực hiện ứng dụng FTP client đơn giản:

Trình tự tiến hành các xử lý của ứng dụng như sau:

- Thiết lập giao tác internet, tạo liên kết FTP từ giao tác này.
- Tạo đối tượng CFtpFileFind từ liên kết FTP, tìm tập tin trên ftp server.
- Tạo đối tượng CInternetFile từ tập tin tìm được. Thực hiện các tác vụ đọc ghi một cách thích hợp.
- Đóng các đối tượng.

Sau đây là các bước thực hiện dự án của ứng dụng nói trên:

- Dùng MFC Wizard, tạo dự án **Ftp** với giao diện chính là dialog.
- Bổ sung chỉ thị: **#include <afxinet.h>** cho tập tin stdafx.h của dự án.
- Thực hiện các cài đặt sau cho lớp dialog CFtpDlg của dự án:
 - Mở dialog resource, cài đặt các control sau:
 - *Hộp nhập địa chỉ FTP server* : Edit IDC_URL
 - *Hộp hiển thị danh sách các tập tin* : ListBox IDC_FILE_LIST
 - *Nút lệnh thực hiện tìm kiếm tập tin* : Button IDOK
 - Tạo biến **m_fileList** cho control IDC_FILE_LIST.
 - Hành vi OnOK ứng với nút chọn IDOK thực hiện tìm tất cả các tập tin trong thư mục gốc của FTP server và đưa vào danh sách:

Cài đặt tương tự ví dụ mục (13.8.4)

Phần cài đặt của **Xử lý tập tin tìm thấy** như sau:

```

m_fileList.ResetContent();
while (TimThay) {
    bWorking = TimFile.FindNextFile();
    m_fileList.AddString(TimFile.GetFileURL());
}

```

- Biên dịch và chạy ứng dụng.

13.8.8 Thực hiện ứng dụng HTTP client đơn giản:

Trình tự tiến hành các xử lý của ứng dụng như sau:

- Thiết lập giao tác internet, tạo liên kết HTTP từ giao tác này.
- Tạo đối tượng CHttpFile cho tập tin quan tâm trên HTTP server.
- Sử dụng đối tượng CHttpFile để QueryInfo hoặc ấn định truy xuất với AddRequestHeaders và SendRequest trên nội dung tập tin.
- Truy xuất dữ liệu: Read, ReadString hoặc Write, WriteString.

Sau đây là các bước thực hiện dự án của ứng dụng nói trên:

- Dùng MFC Wizard, tạo dự án **Http** với giao diện chính là dialog.
- Bổ sung chỉ thị: **#include <afxinet.h>** cho tập tin stdafx.h của dự án.
- Thực hiện các cài đặt sau cho lớp dialog CHttpDlg của dự án:
 - Mở dialog resource, cài đặt các control sau:
 - *Hộp nhập địa chỉ HTTP server* : Edit IDC_URL
 - *Hộp hiển thị nội dung tập tin* : Edit IDC_SOURCE
 - *Nút lệnh thực hiện tải tập tin về* : Button IDOK

- Hành vi OnOK ứng với nút chọn IDOK thực hiện tải nội dung tập tin từ địa chỉ nhập trong hộp nhập:

Cài đặt tương tự ví dụ mục (13.8.6)

Phần cài đặt của **Xử lý nội dung đọc được** như sau:

```
CString line, info;
info.Empty();
while (file->ReadString(line)) {
    info = info + line;
    info = info + "\r\n";
}
SetDlgItemText(IDC_SOURCE, info);
```

- Biên dịch và chạy ứng dụng.

THỰC HÀNH:

1. Xây dựng ứng dụng CHAT với một trong các giao thức TCP và UDP.
2. Xây dựng ứng dụng FTP explorer (xem windows explorer).
3. Xây dựng ứng dụng chơi bài tiến lên trên mạng (mô hình client/server).
4. Xây dựng ứng dụng thi trắc nghiệm trên mạng.
 - Ứng dụng giám thị là trung tâm kết nối với các ứng dụng kiểm tra chạy trên các host làm bài. Ứng dụng giám thị điều khiển đồng bộ các ứng dụng kiểm tra.
 - Ứng dụng kiểm tra chờ chỉ thị từ ứng dụng giám thị, tạo giao diện làm bài với người dùng, thường xuyên thông báo tình trạng bài làm của người dùng cho ứng dụng giám thị quản lý và bảo lưu.
 - Khi bắt đầu, ứng dụng giám thị chủ động kết nối, ứng dụng làm bài chờ kết nối để lấy thông tin về ứng dụng giám thị và các thông tin khác về đề thi, thông tin bảo lưu nếu trước đó có sự cố.
5. Quan sát ứng dụng Yahoo Messenger!. Kết hợp hai kiểu ứng dụng rtf view (11.8) và ứng dụng TCP (13.5) để thực hiện ứng dụng tương tự.
6. Viết ứng dụng mail server và mail client đơn giản.
7. Viết ứng dụng ftp server đơn giản.
8. Viết ứng dụng http server đơn giản (RFC1945).
9. Viết ứng dụng voice chat đơn giản.

Một số vấn đề lập trình hướng đối tượng

A.1 **LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG (OOP):**

Lập trình hướng đối tượng (Object-Oriented Programming) là phương pháp lập trình giải quyết bài toán dựa trên sự phối hợp các đối tượng khách quan trong không gian bài toán. Các đối tượng này hoạt động và tương tác lẫn nhau để đưa bài toán về trạng thái mong muốn.

Phương pháp tiếp cận này cho phép tách không gian bài toán thành các tập hợp nhiều đối tượng, các đối tượng có tính độc lập tương đối với nhau. Do đó, từ một công việc lớn phức tạp ta có thể phân chia thành nhiều công việc nhỏ đơn giản và dễ thực hiện hơn, đồng thời việc điều chỉnh bổ sung tính năng cho mỗi đối tượng cũng không ảnh hưởng đến hoạt động của các đối tượng khác. Nhờ đó, khi có nhu cầu phát triển bài toán thì ta không phải xây dựng lại từ đầu mà có thể dựa trên những nội dung đã có. Đây chính là một ưu điểm lớn so với lập trình cấu trúc.

A.2 **CÁC KHÁI NIỆM:**

A.2.1 **Lớp (Class):**

Lớp là một tập hợp các đối tượng có cùng một số tính chất khảo sát. Các tính chất này có thể là trạng thái (thuộc tính) hay hoạt động của đối tượng.

Ví dụ:

Lớp cá là tập hợp bao gồm những động vật có cùng đặc điểm: vây, mang, máu lạnh và có các hành vi: sống dưới nước, thở bằng mang, đẻ trứng.

A.2.2 **Đối tượng (Object):**

Đối tượng là một thể hiện cụ thể của lớp. Một đối tượng thuộc lớp nào sẽ có những thuộc tính, hành vi của lớp đó.

Sự khác nhau về cách thể hiện thuộc tính, hành vi của đối tượng là cơ sở phân biệt đối tượng với các đối tượng khác trong cùng một lớp.

A.2.3 **Thuộc tính (Attribute):**

Thuộc tính là giá trị phản ánh trạng thái của đối tượng tại một thời điểm xác định, giúp thể hiện đối tượng. Đối tượng có một hoặc nhiều thuộc tính.

A.2.4 **Hành vi (Method):**

Hành vi là khả năng ứng xử của đối tượng với môi trường xung quanh, nó tác động và thay đổi thuộc tính của bản thân đối tượng và các đối tượng liên quan. Hành vi thể hiện mặt hoạt động của đối tượng.

Ví dụ:

Hoạt động sống của con người là quá trình vận động thay đổi bản thân con người và tác động (cũng như bị tác động) đến môi trường xung quanh.

Ta có mô hình của đối tượng như sau:

Đối tượng = Thuộc tính + Hành vi

A.2.5 **Chương trình (Program):**

Dưới góc độ OOP, chương trình là chuỗi thao tác phối hợp của một hay nhiều đối tượng nhằm đạt đến một kết cục mong muốn. Các thao tác này của các đối tượng phải được dàn dựng trước, ta có thể gọi đó là kịch bản.

Chương trình = Tập hợp các đối tượng + Kịch bản

A.3 **ĐẶC ĐIỂM LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG:**

- **Tính khách quan (Objective):**

Một lớp đối tượng hình thành là xuất phát từ một chủ thể khách quan với những thuộc tính và hành vi thể hiện bản chất và chức năng của nó. Kịch bản của chương trình vận dụng những yếu tố khách quan này một cách hợp lý để đạt được mục đích của chương trình.

- **Tính đóng gói (Encapsulation):**

Việc bao hàm các thuộc tính và hành vi trong đối tượng giúp đối tượng có thể hoạt động phối hợp trong độc lập với các lớp đối tượng khác.

- **Tính kế thừa (Inheritance):**

Việc định nghĩa một lớp đối tượng dựa trên các lớp đối tượng đã có gọi là sự kế thừa. Các lớp đã có gọi là lớp cơ sở (based class), lớp được định nghĩa gọi là lớp kế thừa (derived class). Bên cạnh việc kế thừa, lớp kế thừa có thể được bổ sung những hành vi và thuộc tính cần thiết để phục vụ cho nhu cầu mới phát sinh. Cơ chế đó đã tạo sự phát triển cho ứng dụng mà đối tượng tham gia.

- **Tính đa hình (Polymorphism):**

Cơ chế triển khai nhiều cách thể hiện khác nhau cho một hành vi theo nguyên tắc xây dựng lớp cơ sở chứa hành vi cần triển khai và các lớp kế thừa. Mỗi lớp kế thừa sẽ phát triển hành vi đó theo một cách khác nhau. Một đối tượng đại diện lớp cơ sở sẽ được kiến tạo từ lớp kế thừa phù hợp để có cách thể hiện hành vi cần triển khai theo ý muốn. Cơ chế này tạo khả năng ứng xử phong phú của đối tượng.

A.4 PHÂN LOẠI THUỘC TÍNH VÀ HÀNH VI:

Các thuộc tính và hành vi của đối tượng có thể được chia thành ba loại:

- **Public:** Các thuộc tính, hành vi được đối tượng thể hiện bên ngoài.
- **Protected:** Các thuộc tính, hành vi ẩn chứa bên trong đối tượng, hỗ trợ các hoạt động của đối tượng và có thể truyền lại cho các lớp đối tượng kế thừa.
- **Private:** Các thuộc tính, hành vi ẩn chứa bên trong đối tượng, hỗ trợ các hoạt động của đối tượng và không thể truyền lại cho các lớp đối tượng kế thừa.

A.5 CÁC HÀNH VI ĐẶC BIỆT:

- **Hành vi tạo tập:** Hành vi được thực hiện ngay khi đối tượng vừa hình thành. Hành vi này dùng để cài đặt các xử lý khởi tạo các giá trị thuộc tính của đối tượng.

Một lớp đối tượng có thể có một hoặc nhiều hành vi tạo lập. Tên hành vi tạo lập trùng với tên lớp. Các hành vi tạo lập được phân biệt lẫn nhau bởi cấu trúc các tham số của chúng.

- **Hành vi hủy bỏ:** Hành vi được thực hiện trước khi đối tượng chấm dứt sự tồn tại của nó.

Một lớp đối tượng có một hành vi hủy bỏ duy nhất. Tên hành vi hủy bỏ trùng với tên của lớp nhưng được bắt đầu bằng dấu '~'. Hành vi hủy bỏ không có tham số.

Cả hành vi tạo lập và hủy bỏ đều không có kiểu trả về.

A.6 KHAI BÁO LỚP, ĐỐI TƯỢNG TRONG C++:

A.6.1 Khai báo lớp:

Lớp đối tượng được khai báo trong C++ như sau:

```
// Phần khai báo của lớp:
```

```
class Tên_lớp {
public:
    [ Khai báo các thuộc tính, hành vi kiểu public ]
protected:
    [ Khai báo các thuộc tính, hành vi kiểu private ]
private:
    [ Khai báo các thuộc tính, hành vi kiểu private ]
};

// Phần cài đặt của lớp:
kiểu Tên_lớp: Tên_hành_vi_1( [ Danh_sách_tham_số ] ) {
    ... // Các xử lý trong hành vi
}

...
kiểu Tên_lớp: Tên_hành_vi_n( [ Danh_sách_tham_số ] ) {
    ... // Các xử lý trong hành vi
}
```

☞ Nội dung phần khai báo và cài đặt của lớp được đặt trong hai tập tin khai báo (.h) và cài đặt (.cpp). Trong đó:

- Tập tin .H: Chứa toàn bộ phần khai báo của lớp và các chỉ thị định hướng biên dịch (nếu cần).
- Tập tin .CPP: Chứa toàn bộ phần cài đặt của lớp. Ở đầu tập tin này khai báo chỉ thị sử dụng tập tin .H:
#include "Tên_tập_tin_h_liên_quan"

📖 **Thực hành 1:** Xét bài toán quản lý chuỗi, không gian bài toán là các giá trị kiểu chuỗi của C cần quản lý và truy xuất.

☞ Dưới góc độ OOP, lớp đối tượng chuỗi giúp biểu diễn chuỗi có các thuộc tính và hành vi như sau:

- **Thuộc tính:** char* **info** được sử dụng để xin cấp phát vùng nhớ chứa nội dung chuỗi.
- **Hành vi:**
 - Hành vi tạo lập với một tham số là giá trị chuỗi làm nội dung khởi đầu cho đối tượng chuỗi.
 - Hành vi hủy bỏ giải phóng vùng nhớ **info**.
 - Hành vi lấy địa chỉ vùng chứa nội dung chuỗi.

Lớp chuỗi được khai báo trong C++ như sau:

```
// Tập tin String1.h
class CString1 {
public:
    CString1 ( char* s );    // Hành vi tạo lập
    ~CString1 ( void );     // Hành vi hủy bỏ
    char* GetInfo( void );
protected:
    char* info;
};
```

```
// Tập tin String1.cpp
CString1::CString1( char* s ) {
    info = new char[ strlen( s ) + 1 ];
    strcpy( info, s );
}

CString1::~CString1( ) {
    delete info;
}

char* CString1::GetInfo( void ) {
    return info;
}
```

A.6.2 Khai báo đối tượng:

Đối tượng được khai báo nhờ lớp tương ứng. Lớp có thể xem như là một kiểu dữ liệu và đối tượng chính là biến ứng với kiểu đó.

Cú pháp khai báo biến đối tượng trong C++ như sau:

Tên_lớp Tên_biến_đối_tượng [*Danh sách giá trị làm tham số*] ;

A.6.3 Sử dụng đối tượng trong chương trình:

Thực hiện hành vi của biến đối tượng (các hành vi public) như sau:

Tên_biến_đối_tượng.Tên_hành_vi [*Danh sách giá trị làm tham số*] ;

```
// Chương trình Main.cpp sử dụng lớp CString1 khai báo ở trên.
#include "String1.h"
void main( void ) {
    CString1 a( "Vo Van A" );
    printf( "Gia tri chuoi luu la: %s ", a.GetInfo() );
}
```

A.7 KẾ THỪA TRONG C++:

C++ cho phép định nghĩa lớp kế thừa từ một hoặc nhiều lớp cơ sở. Cú pháp thực hiện khai báo này như sau:

```
class Tên_lớp_kế_thừa: [ public | private ] Tên_lớp_cơ_sở1
                        [, [public | private] Tên_lớp_cơ_sở2 [, ... ]
{
    ... // Các khai báo bổ sung của lớp kế thừa
};
```

Đặc điểm kế thừa qui định mức độ kế thừa của lớp kế thừa từ lớp cơ sở. Có hai kiểu khác nhau của đặc điểm kế thừa và có ý nghĩa như sau:

Thuộc tính lớp cơ sở	Thuộc tính nhận được cho lớp kế thừa	
public	public	private
protected	protected	private
private	Không truy xuất được	Không truy xuất được
Đặc điểm kế thừa	public	Private

Ví dụ: Khai báo lớp CString1B kế thừa hoàn toàn từ lớp CString1.

```
class CString1B : public CString1 { };
```

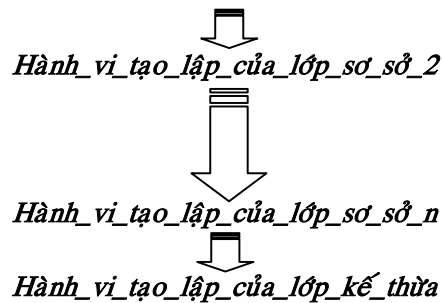
A.7.1 Kế thừa hành vi tạo lập:

Mỗi hành vi tạo lập bổ sung của lớp kế thừa chỉ được phép sử dụng một hành vi tạo lập duy nhất từ một lớp cơ sở. Khai báo có cú pháp như sau:

```
Hành_vi_tạo_lập_của_lớp_kế_thừa( [ Danh sách tham số ] ) :
    Hành_vi_tạo_lập_của_lớp_sơ_sở_1( [ Các giá trị tham số ] )
    [, Hành_vi_tạo_lập_của_lớp_sơ_sở_2( [ Các giá trị tham số ] )
    [, ... ] ] {
    ...
};
```

Khi một đối tượng thuộc lớp kế thừa hình thành, xử lý trong hành vi tạo lập của lớp kế thừa và xử lý trong các hành vi tạo lập của các lớp cơ sở mà lớp đăng ký kế thừa sẽ được thực hiện theo thứ tự như sau:

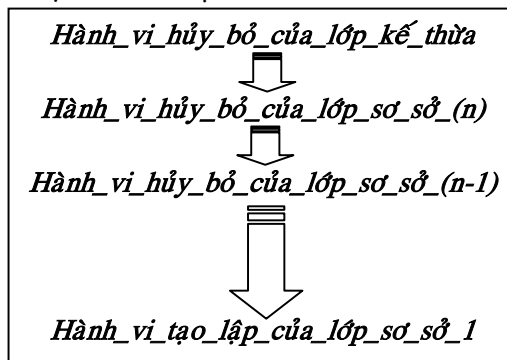
Hành_vi_tạo_lập_của_lớp_sơ_sở_1



A.7.2 Kế thừa hành vi hủy bỏ:

Hành vi hủy bỏ của lớp kế thừa (mặc nhiên thuộc loại public) tự động kế thừa hành vi hủy bỏ của lớp cơ sở mà không cần phải khai báo.

Khi một đối tượng thuộc lớp kế thừa chấm dứt hoạt động, xử lý trong hành vi hủy bỏ của của lớp kế thừa và xử lý trong các hành vi hủy bỏ của các lớp cơ sở sẽ được thực hiện theo thứ tự như sau:



A.7.3 Thực hiện hành vi lớp cơ sở:

Hành vi của lớp kế thừa được phép chủ động sử dụng hành vi của lớp cơ sở của nó. Cách thực hiện hành vi của lớp cơ sở như sau:

```
Tên_lớp_cơ_sở::Tên_hành_vi( [ Danh_sách_giá_trị_tham_số ] );
```

🔖 **Thực hành 2:** Mở rộng không gian bài toán thực hành 1, với lớp đối tượng CString2 kế thừa từ CString1 và bổ sung hành vi tạo lập nhận tham số là một đối tượng CString1.

Lớp CString2 được khai báo trong C++ như sau:

```
// Tập tin String2.h
#include "String1.h"
```

```
class CString2 : public CString1 {
public:
    // Hành vi tạo lập bổ sung nhận một tham số là đối tượng CString1
    CString2( CString1* s ) : CString1( s->GetInfo() ) { }
};
```

```
// Chương trình Main2.cpp sử dụng lớp CString2 khai báo ở trên.
#include "String2.h"
void main( void ) {
    CString2 a( "Vo Van B" );
    CString2 b( &a );
    printf( "Gia tri chuoi luu la: %s ", b.GetInfo() );
    // Kết quả nhận được: Gia tri chuoi luu la: Vo Van B
}
```

A.8 KHAI BÁO HÀNH VI TOÁN TỬ SỐ HỌC:

Khai báo hành vi toán tử số học là hình thức thay thế hiện hành vi theo dạng hàm (kiểu đại số) thành dạng toán tử (kiểu số học).

Có hai loại hành vi toán tử:

- Hành vi toán tử hai ngôi: Là hành vi có một tham số đối tác.

Khai báo:

Kiểu Tên_lớp::operator Ký_hiệu_hành_vi([Danh_sách_tham_số]);

- Hành vi toán tử một ngôi: Là hành vi không có tham số đối tác nào. Hành vi này có thể được sử dụng cho phép chuyển kiểu đối tượng.

Khai báo:

Tên_lớp::operator Ký_hiệu_hành_vi(void);

🔖 **Thực hành 3:** Mở rộng không gian bài toán thực hành 1, với các toán tử gán (=), cộng (+), và chuyển kiểu lớp đối tượng chuỗi về kiểu chuỗi của C(char*) thay cho hành vi GetInfo.

🔗 Tạo lớp kế thừa CString3 từ lớp cơ sở CString1. Thực hiện bổ sung các hành vi toán tử sau đây:

- Toán tử chuyển kiểu: **char ***
- Toán tử gán (=) với tham số là một giá trị kiểu chuỗi của C.
- Toán tử cộng (+) với tham số là một giá trị kiểu chuỗi của C và trả về con trỏ đến đối tượng chuỗi mới.

Lớp CString3 được khai báo trong C++ như sau:

```
// Tập tin String3.h
```

```
#include "String1.h"
class CString3 : public CString1 {
public:
    CString3( char* s ) : CString1( s ) { }
    // Toán tử chuyển kiểu
    operator char* ( void ) {
        return info;
    }
    void operator = ( char* s );
    CString3* operator + ( char* s );
};
```

```
// Tập tin String3.cpp
#include "String3.h"
void CString3::operator = ( char* s ) {
    delete info;
    info = new char[ strlen( s ) + 1 ];
    strcpy( info, s );
}

CString3* CString3::operator + ( char* s ) {
    char* ketqua;
    ketqua = new char[ strlen(info) + strlen( s ) + 1 ];
    sprintf( ketqua, "%s%s", info, s );
    CString3* kq = new CString3( ketqua );
    delete ketqua;
    return kq;
}
```

```
// Chương trình Main3.cpp sử dụng lớp CString3 khai báo ở trên.
#include "String3.h"
void main( void ) {
    CString3 s1( "Chao Cac " );
    CString3 s2( "Ban" );
    CString3 *s = (s1 + s2);
    printf( "Cong hai chuoi la: %s", *s );
    delete s;
    // Kết quả nhận được: Cong hai chuoi la: Chao Cac Ban
}
```

A.9. CON TRỎ this:

this là con trỏ dùng tham chiếu đến bản thân đối tượng trong nội dung cài đặt của nó. Hành vi toán tử = của lớp CString3 có thể viết như sau:

```
void CString3::operator = ( char* s ) {
    delete this->info;
    this->info = new char[ strlen( s ) + 1 ];
    strcpy( this->info, s );
}
```

A.10 HÀNH VI virtual:

Hành vi virtual được cài đặt trong lớp cơ sở nhằm mục đích triển khai nhiều cách thể hiện khác nhau của hành vi này trên các lớp kế thừa.

Trong ví dụ sau, lớp A được cài đặt với hành vi khung Action. Trong các lớp B và C kế thừa từ A, hành vi này được thể hiện khác nhau như sau:

- Lớp B : Bật tiếng beep của PC speaker.
- Lớp C : Bật hoặc tắt ký tự x.

```
class A {
public:
    void Action( void ) { } // Hành vi khung cơ sở
}

class B : public A {
public:
    void Action( void ) {
        // Cách thể hiện Action của lớp B
        putch( 7 );
    }
}

class C : public A {
public:
    void Action( void ) {
        // Cách thể hiện Action của lớp C ; bật, tắt chữ x
        static char ch = 'x';
        putch(x);
        ch = ( ch == 'x' )? ' ': 'x' ;
    }
}

void main ( void ) {
```

```

A* a[5];           // 5 phần tử kiểu lớp A
for ( int i = 0; i<5; i++ ) {
    if ( i % 2 )
        a[ i ] = new B();    // Khởi tạo ngẫu nhiên theo kiểu lớp B
    else
        a[ i ] = new C();    // hoặc lớp C
}
while ( !kbhit() ) {
    for ( i = 0; i < 5 ; i++ )
        a[ i ]->Action( );    // Hành động theo cách của lớp B hoặc C
    delay( 100 );             // Tạm dừng
}
for ( i = 0; i < 5 ; i++ )
    delete a[ i ];
}

```

A.11 THUỘC TÍNH VÀ HÀNH VI TÍNH:

Thuộc tính và hành vi tính là những thuộc tính và hành vi có thể khai thác được mà không cần sử dụng đối tượng cụ thể của lớp.

Trong phần khai báo của lớp:

```

static Kiểuthuộc tính Thuộc tính;
static Kiểutrả về Hành vi ( [ Danh_sách_tham_số ] );

```

Trong phần cài đặt của lớp:

```

Kiểuthuộc tính Tênlớpởhữu::Thuộc tính = Giá trị khởi đầu;
Kiểutrả về Tênlớpởhữu::Hành vi ( [ Danh_sách_tham_số ] ) {
    ...           // Nội dung cài đặt của hành vi
}

```

Lưu ý: Hành vi static chỉ được phép sử dụng các thuộc tính static và hành vi static của lớp.

Trong ví dụ sau, ta cài đặt lớp Duongtron với thuộc tính bán kính, thuộc tính pi, và hành vi Chuvi có hai thể hiện như sau:

- Chuvi(float R) : Trả về chu vi của một đường tròn có bán kính R.
 - Chuvi(void) : Trả về chu vi của đường tròn quản lý bởi đối tượng.
- Có thể sử dụng hành vi Chuvi(R) mà không dùng đối tượng thuộc lớp.

Lớp Duongtron được khai báo như sau:

```

class Duongtron {
protected:
    float R;
    static float pi;
public:
    Duongtron( float bankinh ) {
        R = bankinh;
    }

    float Chuvi( void )
    {
        return R * R * pi;
    }

    static float Chuvi( float bankinh )
    {
        return R * R * pi;
    }
};
float Duongtron::pi = 3.14159;

```

// Chương trình minh họa sử dụng lớp Duongtron

```

void main( void ) {
    // Dùng hành vi tính Chuvi( . ) của Duongtron để tính chu vi
    // của một đường tròn bất kỳ
    printf( "Chu vi duong tron ban kinh R = 10 la : %0.2f" ,
        Duongtron::Chuvi( 10 ) );

    // Dùng hành vi Chuvi() để tính chu vi của đối tượng Duongtron
    Duongtron a( 20 );
    printf( "Chu vi duong tron ban kinh R = 20 la : %0.2f", a.Chuvi() );
}

```

- ☞ Nếu thuộc tính **pi** của Duongtron không phải là thuộc tính static thì hành vi Chuvi(float Bankinh) của Duongtron sẽ không hợp lệ vì hành vi này đã sử dụng một thuộc tính không phải static.