Mục lục

Trang

Bài 1: GIỚI THIỆU CHUNG

Phân bố thời lượng:
- *Số tiết giảng ở lớp: 6 tiết*
- *Số tiết tự học ở nhà: 6 tiết*
- *Số tiết cài đặt chương trình ở nhà: 12 tiết*

1. Mở đầu

- Các ứng dụng của Windows rất dễ sử dụng, nhưng rất khó đối với người đã tạo lập ra chúng. Để đạt được tính dễ dùng đòi hỏi người lập trình phải bỏ ra rất nhiều công sức để cài đặt.
- Lập trình trên Windows khó và phức tạp hơn nhiều so với lập trình trên DOS. Tuy nhiên lập trình trên Windows sẽ giải quyết được một số vấn đề khó khăn trong môi trường DOS như xây dựng giao diện người dùng, quản lý bộ nhớ ảo, độc lập thiết bị vào ra, thâm nhập Internet, khả năng chia sẻ tài nguyên, ...
- Windows cung cấp các hàm để người lập trình thâm nhập các đặc trưng của hệ điều hành gọi là giao diện lập trình ứng dụng (Application Programming Interface – API). Những hàm này được đặt trong các thư viện liên kết động (Dynamic Link Library – DLL). Các chương trình ứng dụng sử dụng chúng thông qua các lời gọi hàm và chỉ chia sẻ được khi trong máy có cài đặt Windows.
- Vài điểm khác biệt giữa lập trình Windows và DOS:

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

2. Các thư viện lập trình của Windows

SDK – Software Development Kit

- Là bộ thư viện lập trình nền tảng của HĐH Windows.
- Cung cấp tất cả các công cụ cần thiết để xây dựng 1 ứng dụng trên Windows.
- Được sử dụng như là thư viện cơ sở để tạo ra những thư viện cao cấp hơn trong những ngôn ngữ lập trình. VD: OWL của BorlandC, MFC của Visual C++,…
- Một số thành phần cơ bản của SDK:
  - Win32 API.
  - GDI/GDI+.
  - Windows Multimedia.
  - OpenGL.
  - DirectX.
  - COM/COM+.
  - ADO (ActiveX Data Object).
  - OLE DB.
  - …

  (Xem thêm MSDN/PlatForm SDK Documentation/Getting started/Content of Platform SDK).
- OWL – Object Windows Library:
  - Là bộ thư viện hướng đối tượng của BorlandC++.
- MFC – Microsoft Foundation Classes:
  - Là bộ thư viện hướng đối tượng của Visual C++.
- Một ứng dụng trên Windows có thể được viết bằng:
  - Thư viện SDK.
  - Một thư viện khác (OWL, MFC,…) phối hợp với SDK.
- ***Các loại ứng dụng:***
  - Win32 Console: ứng dụng 32 bits, với giao diện dạng DOS command line.

- Win32 (SDK): ứng dụng 32 bits, chỉ sử dụng thư viện SDK.
- Win32 DLL: ứng dụng 32 bits, dạng thư viện liên kết động (Dynamic – Linked Library), sử dụng SDK.
- Win32 LIB: ứng dụng 32 bits, dạng thư viện liên kết tĩnh (Static – Linked Library).
- MFC EXE: ứng dụng 32 bits, sử dụng thư viện Microsoft Foundation Class.
- MFC DLL: ứng dụng 32 bits, dạng thư viện liên kết động (Dynamic – Linked Library), sử dụng MFC.
- …

3. Các khái niệm cơ bản

❖ Message:

Trao đổi thông tin giữa chương trình ứng dụng và hệ điều hành.

❖ Thủ tục Window:

Được gọi bởi hệ thống để xử lý các Message nhận được.

❖ Hàng đợi Message:

Mỗi chương trình có 1 hàng đợi Message để chứa các Message. Mỗi chương trình có vòng lặp Message.

❖ Handle:

Một giá trị 32 bits không dấu (unsigned) do HĐH tạo ra để làm định danh cho một đối tượng (cửa sổ, file, vùng nhớ, menu,…).

❖ ID (Identifier):

Một giá trị nguyên do ứng dụng tạo ra để làm định danh cho 1 đối tượng (menu item, control).

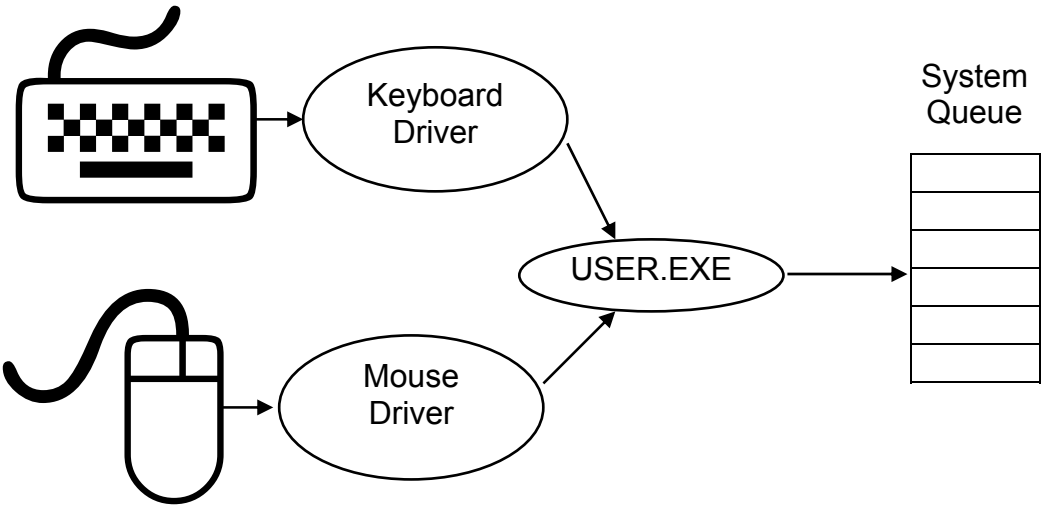
❖ Instance:

Một giá trị nguyên do HĐH tạo ra để định danh 1 thể hiện đang thực thi của ứng dụng.

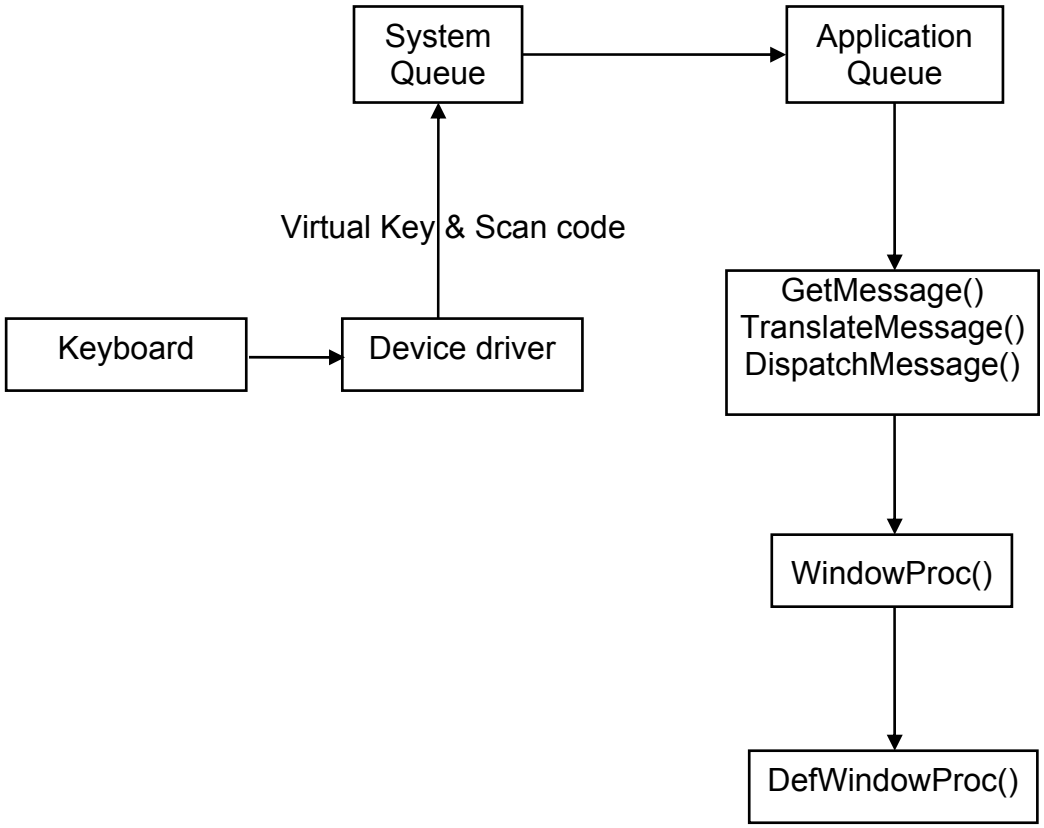
❖ Callback:

Thuộc tính của 1 hàm/ thủ tục sẽ được gọi bởi HĐH, không phải bởi ứng dụng.

4. Lập trình sự kiện (Even driven programming)



Phát sinh các sự kiện và thông điệp



Qui trình xử lí thông điệp

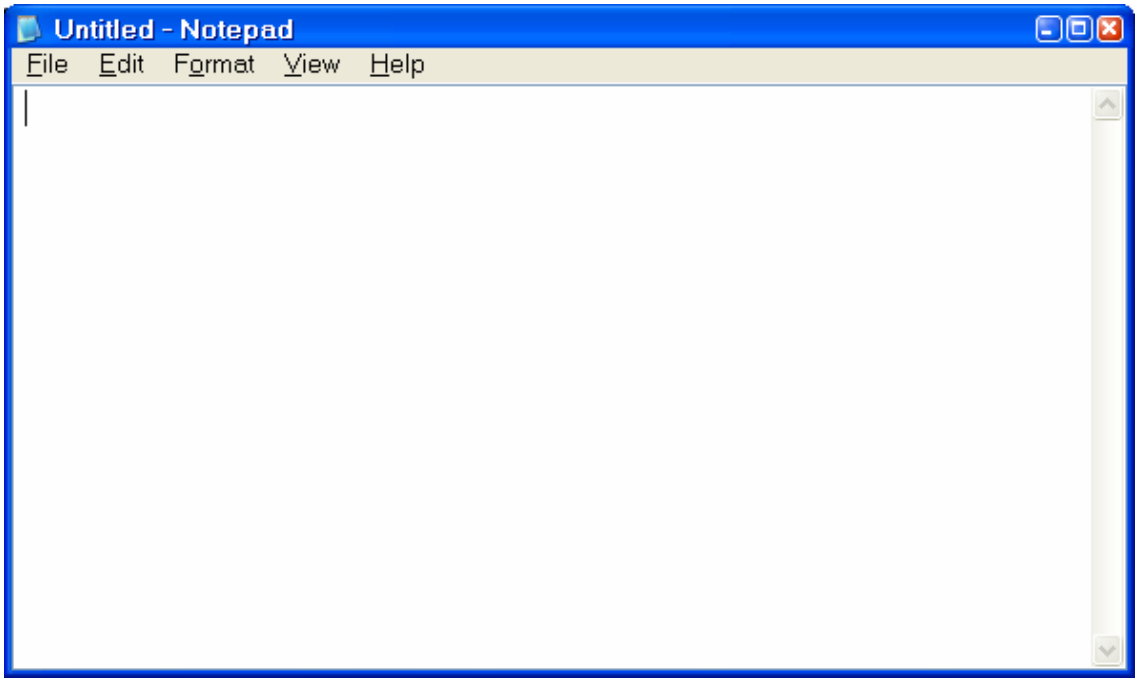
```
MSG msg;
while(GetMessage(&msg,NULL,0,0))
{
        TranslateMessage(&msg);
        DispatchMessage(&msg);
}
return msg.wParam;
```

5. Các thành phần giao diện đồ họa (GUI)

❖ GUI: Graphics User Interface.

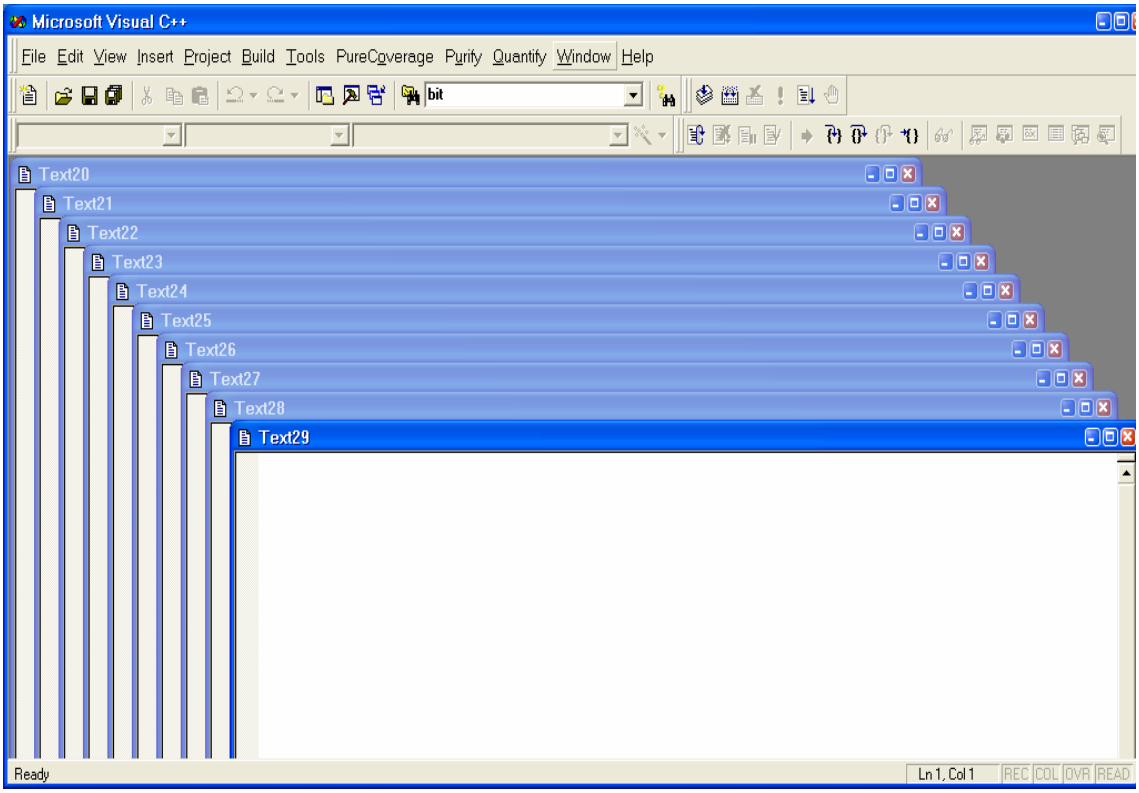
❖ Các dạng GUI cơ bản:

- SDI – Single Document Interface:
  - ✓ Một cửa sổ làm việc.
  - ✓ Cho phép thay đổi kích thước cửa sổ (Resizeable).
  - ✓ Không có các cửa sổ con.
  - ✓ Ví dụ: NotePad, Paint,…

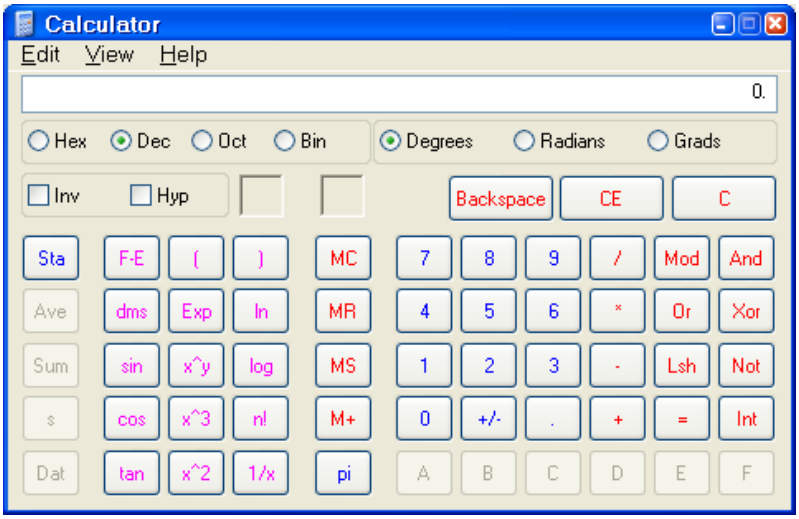


- MDI – Multi Document Interface:
  - ✓ Một cửa sổ làm việc chính (Frame window) và nhiều cửa sổ con (Child window).

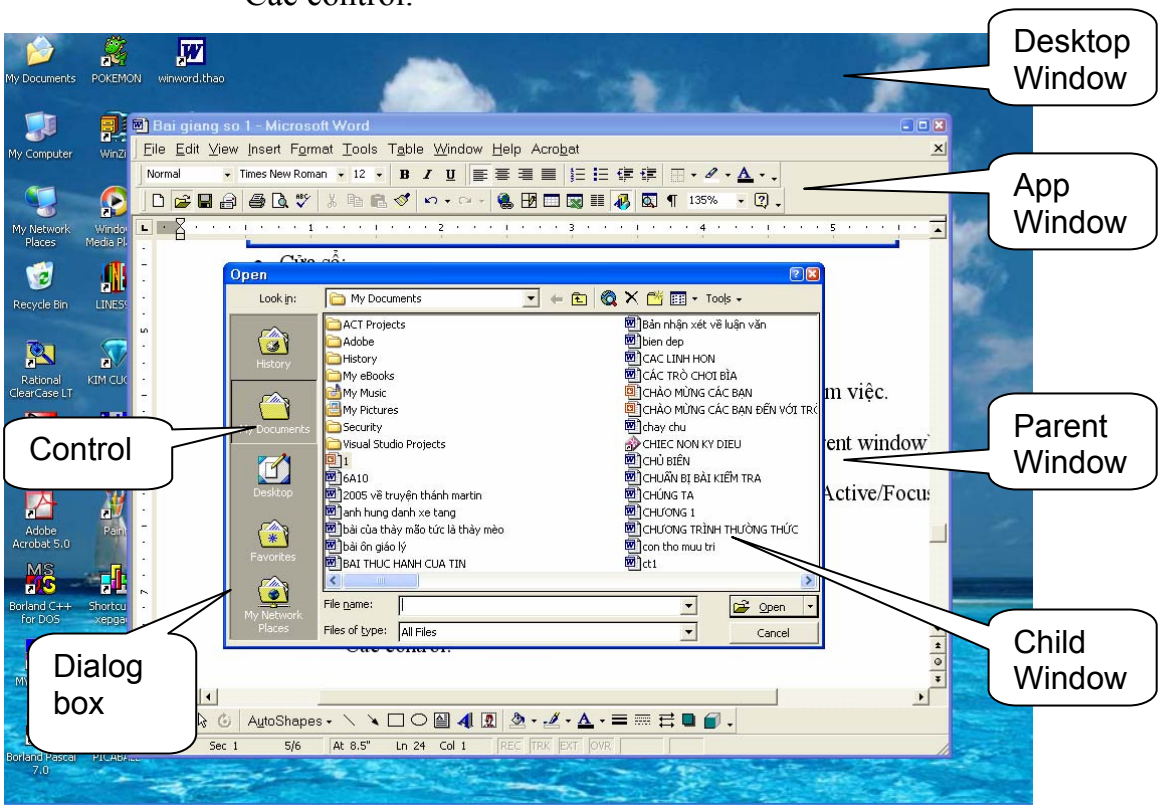
- ✓ Cho phép thay đổi kích thước cửa sổ (Resizeable).
- ✓ Cho phép Maximize/Minimize/Close các cửa sổ con.
- ✓ Ví dụ: Word, Excel, VC++,…

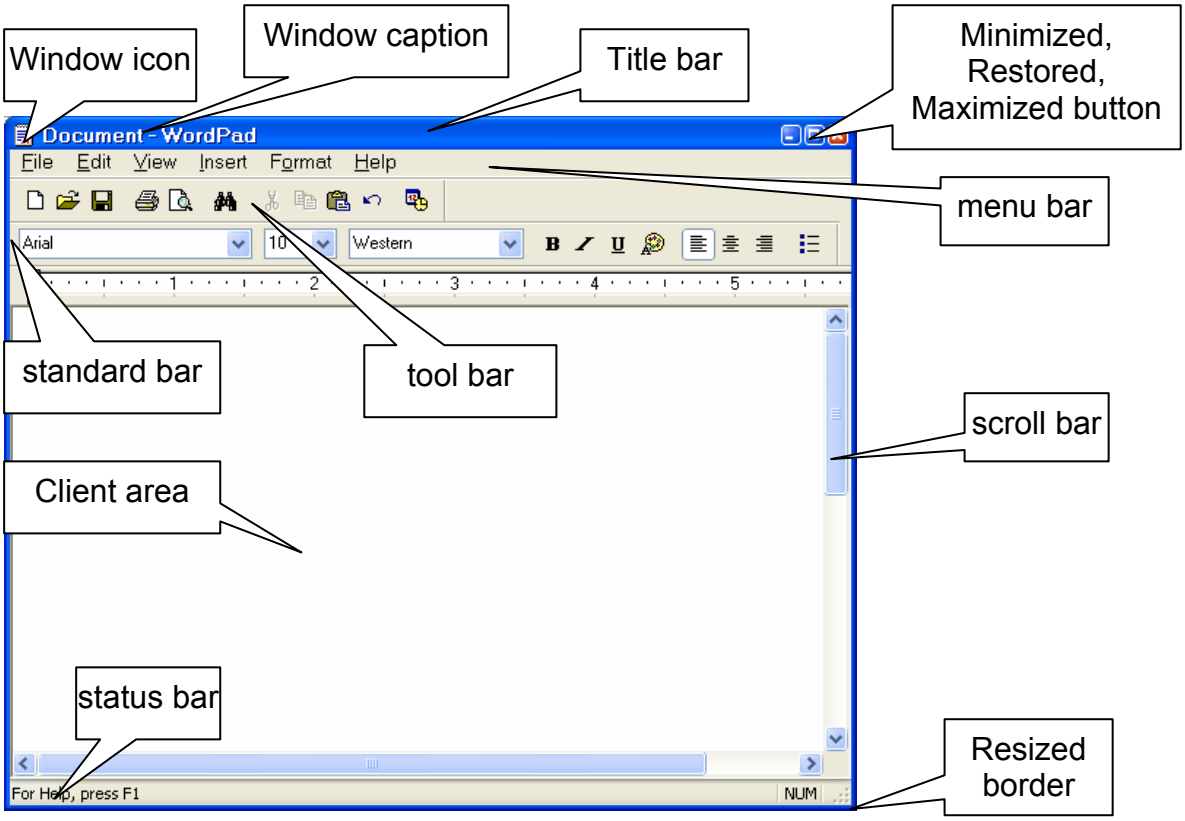


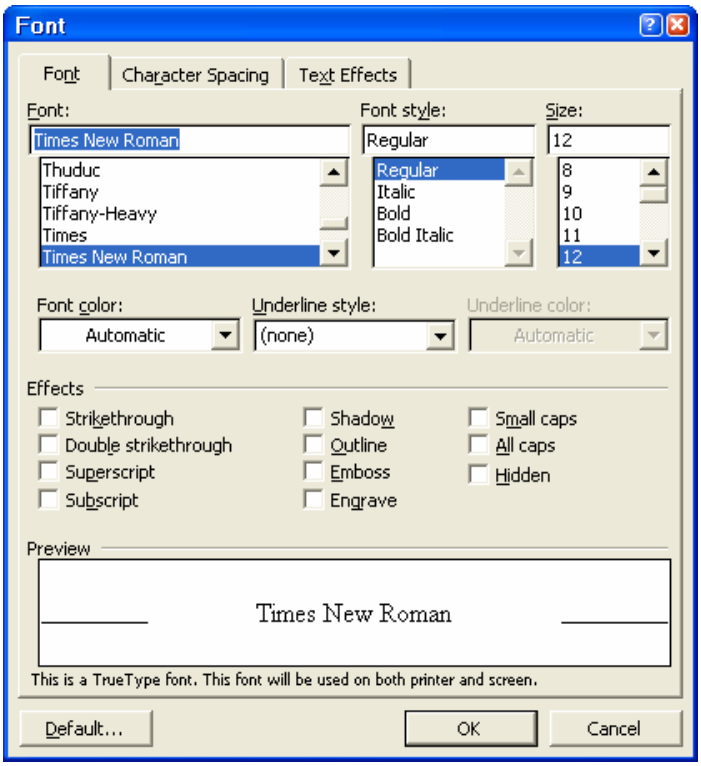
- Dialog:
  - ✓ Một cửa sổ làm việc.
  - ✓ Thường có kích thước cố định.
  - ✓ Thường không có menu bar.
  - ✓ Thường có các button, edit box, list-box,…
  - ✓ Ví dụ: Calculator, CD Player,…

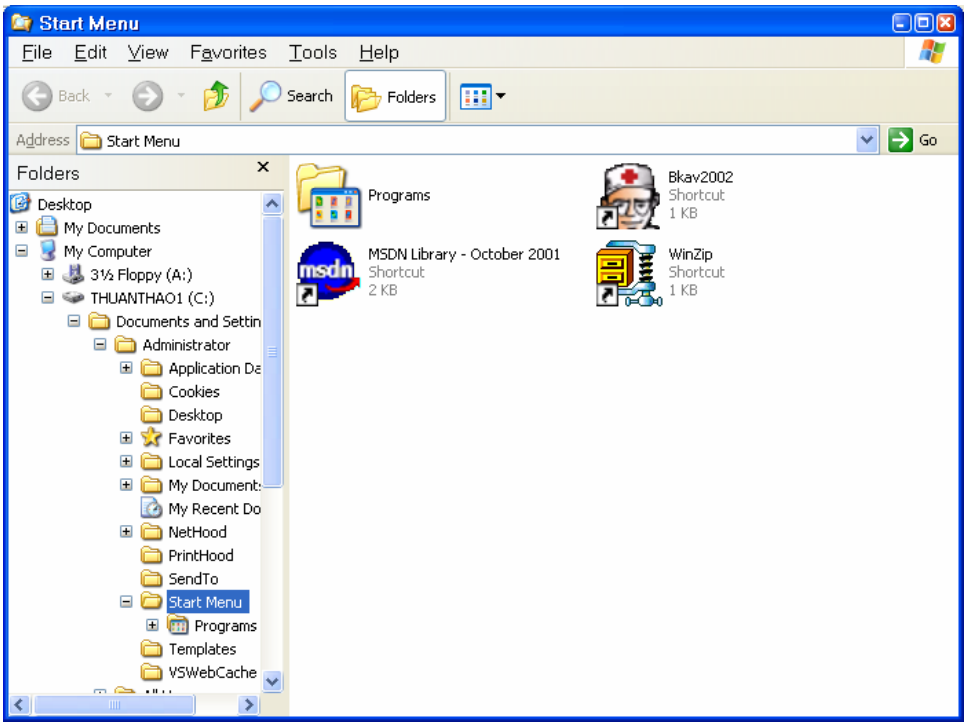


- **Cửa sổ:**
  - ✓ **Định nghĩa:**
    - Là 1 vùng chữ nhật trên màn hình.
    - Dùng để hiển thị kết quả output.
    - Và nhận các input từ người dùng
  - ✓ **Công việc đầu tiên của 1 ứng dụng GUI là tạo 1 cửa sổ làm việc.**
  - ✓ **Nguyên tắc quản lý:**
    - Mô hình phân cấp: mỗi cửa sổ đều có 1 cửa sổ cha (parent window), ngoại trừ cửa sổ nền Desktop.
    - Tại mỗi thời điểm, chỉ có 1 cửa sổ nhận input từ user (Active/Focused window).
  - ✓ **Phân loại:**
    - Cửa sổ Desktop.
    - Cửa sổ tiêu chuẩn.
    - Cửa sổ hộp thoại (Dialog box).
    - Các control.

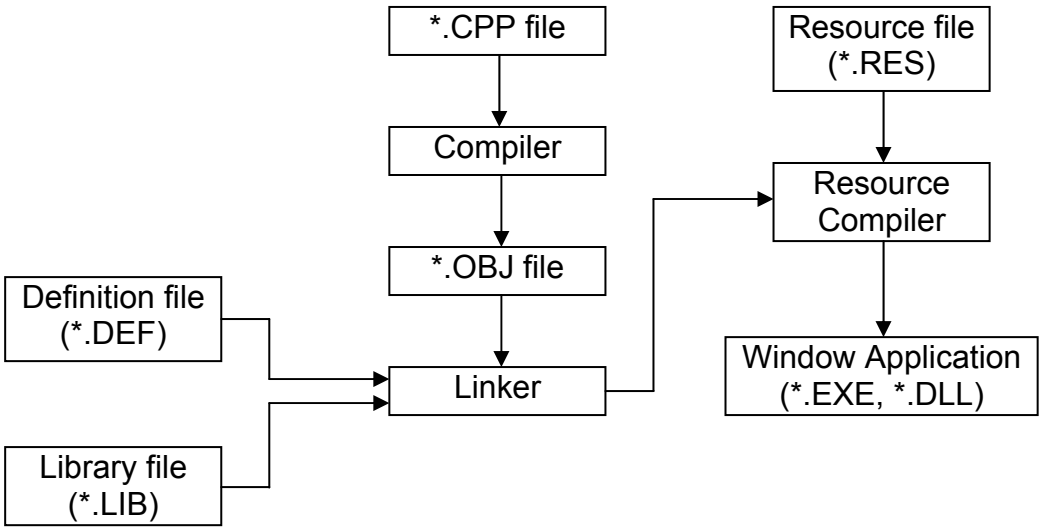



Window icon
Window caption
Title bar
Minimized, Restored, Maximized button
Document - WordPad
File Edit View Insert Format Help
menu bar
Arial
10
Western
standard bar
tool bar
scroll bar
Client area
status bar
For Help, press F1
NUM
Resized border


Font
Font
Character Spacing
Text Effects
Font:
Times New Roman
Thuduc
Tiffany
Tiffany-Heavy
Times
Times New Roman
Font style:
Regular
Regular
Italic
Bold
Bold Italic
Size:
12
8
9
10
11
12
Font color:
Automatic
Underline style:
(none)
Underline color:
Automatic
Effects
Strikethrough
Double strikethrough
Superscript
Subscript
Shadow
Outline
Emboss
Engrave
Small caps
All caps
Hidden
Preview
Times New Roman
This is a TrueType font. This font will be used on both printer and screen.
Default...
OK
Cancel



6. Cấu trúc chương trình C for Win



7. Qui trình hoạt động của chương trình ứng dụng

- Cửa sổ được hiển thị lên màn hình.
- Windows chờ cửa sổ gửi thông điệp.
- Các thông điệp được Windows gửi trả lại chương trình ứng dụng thông qua lời gọi hàm của chúng trong chương trình ứng dụng.
- Khi nhận được thông điệp, chương trình ứng dụng gọi các hàm API và hàm của riêng chúng để thực hiện công việc mong muốn.

Lập trình trên Windows là lập trình trên cơ sở thông điệp, quá trình trao đổi thông tin và điều khiển dựa trên thông điệp. Có rất nhiều thông điệp được phát sinh ngẫu nhiên như nhấn phím hay chuột, chọn menu, ...

Tương tác của ứng dụng với người sử dụng thông qua một hay nhiều cửa sổ, tạo lập các cửa sổ khi cần thiết và quản lý thông tin trong đó.

8. Một số quy ước đặt tên

a. Tên hằng

Chữ cái viết hoa, nên phân loại các hằng theo nhóm. Thông thường gồm có 2 phần: Phần đầu là loại nhóm và phần sau là tên hằng. Loại nhóm và tên hằng cách nhau bằng dấu gạch nối.

Ví dụ: WM_DESTROY (Hằng này được định nghĩa trong windows.h, WM cho ta biết hằng DESTROY thuộc nhóm thông điệp cửa sổ Windows Message)

b. Tên biến

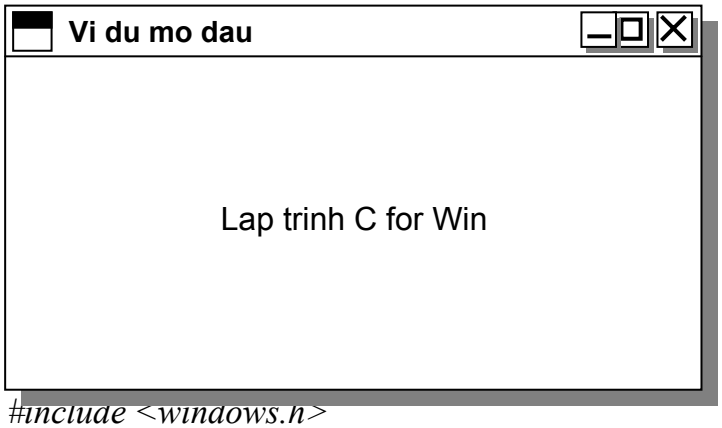
Tên biến bắt đầu bằng ký tự thường cho biết kiểu dữ liệu.

Ví dụ: iTong cho biết biến Tong có kiểu int.

Các tiền tố thường dùng khác: c(char), l (long), p (poiter), d (WORD), dw (DWORD), h (chỉ số).

9. Ví dụ

Xây dựng chương trình hiển thị một cửa sổ như sau:




```
1   #include <windows.h>
2   LRESULT CALLBACK XulyMessage (HWND,UNIT,WPARAM,LPARAM);
3   char szAppName [ ] = "Vidu";
```

```
4   int WINAPI WinMain (HANDLE hInst, HANDLE hPrevInst,
5                       LPSTR lpszCmdLine, int nCmdShow)
6   {
7       HWND hwnd;      MSG msg;
8       WNDCLASSEX wndclass;
9       wndclass.cbSize = sizeof(wndclass);
10      wndclass.style = CS_HREDRAW | CS_VREDRAW;
11      wndclass.lpfnWndProc = XulyMessage;
12      wndclass.cbClsExtra = 0;
13      wndclass.cbWndExtra = 0;
14      wndclass.hInstance = hInst;
15      wndclass.hIcon = LoadIcon (NULL, IDI_APPLICATION);
16      wndclass.hCursor = LoadCursor (NULL, IDC_ARROW);
17      wndclass.hbrBackground = GetStockObject (WHITE_BRUSH);
18      wndclass.lpszMenuName = NULL;
19      wndclass.lpszClassName = szAppName;
20      wndclass.hIconSm = LoadIcon (NULL, IDI_APPLICATION);
21      RegisterClassEx(&wndclass);
22      hwnd = CreateWindow(szAppName,
23                          "Vi du mo dau",
24                          WS_OVERLAPPEDWINDOW,
25                          CW_USEDEFAULT, CW_USEDEFAULT,
26                          CW_USEDEFAULT, CW_USEDEFAULT,
27                          HWND_DESKTOP,
28                          NULL,
29                          hInst,
30                          NULL);
31      ShowWindow (hwnd, nCmdShow);
32      UpdateWindow (hwnd);
33      while (GetMessage (&msg, NULL, 0, 0))
34      {
35          TranslateMessage (&msg);
36          DispatchMessage (&msg);
37      }
38      return msg.wParam;
39  }
40  LRESULT CALLBACK XulyMessage (HWND hwnd, UINT iMsg,
41                                WPARAM wParam, LPARAM lParam)
42  {
43      HDC hdc;
44      PAINTSTRUCT ps;
45      RECT rect;
46      switch (iMsg)
47      {
48          case WM_PAINT:
49              hdc = BeginPaint (hwnd, &ps);
```

```
50                      GetClientRect (hwnd, &rect);
51                      DrawText (hdc, "Lap trinh C for Win", -1, &rect,
52                          DT_SINGLELINE | DT_CENTER | DT_VCENTER);
53                      EndPaint (hwnd, &ps);
54                      break;
55              case WM_DESTROY:
56                      PostQuitMessage(0);
57                      break;
58              default:
59                      return DefWindowProc (hwnd, iMsg, wParam, lParam);
60      }
61      return 0;
62  }
```

Ta sẽ khảo sát ví dụ trên để nắm được nguyên lý hoạt động của chúng. Trên đây là đoạn chương trình đơn giản trên Windows, chương trình chỉ hiển thị 1 khung cửa sổ và 1 dòng chữ nhưng có rất nhiều lệnh mà cú pháp rất khó nhớ. Do vậy, nguyên tắc lập trình trên Windows chủ yếu là sao chép và chỉnh sửa những nơi cần thiết dựa vào một chương trình mẫu có sẵn.

a. Hàm WinMain() được thực hiện đầu tiên hay còn gọi là điểm vào của chương trình.

   ❖ Ta thấy hàm này có 4 tham số:

      o hInst, hPrevinst: Chỉ số chương trình khi chúng đang chạy. Vì Windows là hệ điều hành đa nhiệm, có thể có nhiều bản của cùng một chương trình cùng chạy vào cùng một thời điểm nên phải quản lý chặt chẽ chúng. hInst là chỉ số bản chương trình vừa khởi động, hPrevinst là chỉ số của bản đã được khởi động trước đó và chúng luôn có giá trị NULL.

      o lpszCmdLine: chứa địa chỉ đầu của xâu ký tự các đối số dòng lệnh.

      o nCmdShow: Cho biết cách thức hiển thị cửa sổ khi chương trình khởi động. Windows có thể gán giá trị SW_SHOWNORMAL hay SW_SHOWMINNOACTIVE.

      Các tham số trên do hệ điều hành truyền vào.

   ❖ Định nghĩa lớp cửa sổ và đăng ký với Windows

      o Lớp cửa sổ (window class):

- Là một tập các thuộc tính mà HĐH Windows sử dụng làm khuôn mẫu (template) khi tạo lập cửa sổ.
- Mỗi lớp cửa sổ được đặc trưng bằng 1 tên (class-name) dạng chuỗi.
- Phân loại class:
  - Lớp cửa sổ của hệ thống (System class):
    Được định nghĩa trước bởi HĐH Windows.
    Các ứng dụng không thể hủy bỏ.

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

  - Lớp cửa sổ do ứng dụng định nghĩa:
    Được đăng ký bởi ứng dụng.
    Có thể hủy bỏ khi không còn sử dụng nữa.
    Lớp toàn cục của ứng dụng (Application global class).
    Lớp cục bộ của ứng dụng (Application local class).
  - Mỗi cửa sổ đều thuộc một lớp xác định.

o Khi lần đầu chạy, ứng dụng phải định nghĩa và đăng ký lớp với cửa sổ (Window Class). Đây là cấu trúc dữ liệu mô tả tính chất của cửa sổ, lần lượt ta gán các giá trị ban đầu cho các thành phần của cấu trúc lớp cửa sổ, bao gồm: Kích thước, kiểu, địa chỉ hàm xử lý thông điệp cửa sổ, định nghĩa hình dạng cho con trỏ chuột (cursor) và biểu tượng (Icon), màu nền, tên lớp cửa sổ.

Macro	Màu nền cửa sổ

BLACK_BRUSH	Đen
DKGRAY_BRUSH	Xám đen
HOLLOW_BRUSH	Không tô
LTGRAY_BRUSH	Xám nhạt
WHITE_BRUSH	Trắng

```
struct WNDCLASSEX {
        UINT cbSize;
        UINT style;
        WNDPROC lpfnWndProc;
        int cbClsExtra;
        int cbWndExtra;
        HINSTANCE hInstance;
        HICON hIcon;
        HCURSOR hCursor;
        HBRUSH hbrBackground;
        LPCTSTR lpszMenuName;
        LPCTSTR lpszClassName;
        HICON hIconSm;
};
```

- Sau khi đã định nghĩa các thành phần lớp cửa sổ ta phải đăng ký lớp cửa sổ với hệ điều hành (RegisterClassEX).

  *ATOM RegisterClassEx (CONST WNDCLASSEX *lpWClass);*

  với: Kiểu giá trị của ATOM được định nghĩa trong window.h là WORD; lpWClass là con trỏ đến cấu trúc lớp cửa sổ; hàm này trả về chỉ số của lớp cửa sổ.
- Có hai nguyên nhân dẫn đến việc đăng ký cửa sổ thất bại:
  - Trùng tên giữa các ứng dụng trong hệ điều hành.
  - Không đủ bộ nhớ.

❖ Tạo lập cửa sổ làm việc (Frame Window)

- Sau khi đăng ký thành công ta có thể tạo lập cửa sổ thông qua hàm CreateWindow().

HWND CreateWindow (

LPCSTR lpClassName,

LPCSTR lpWinName,

DWORD dwStyle,

int X, int Y,

int Width, int Height,

HWND hParent,

HMENU hMenu,

HINSTANCE hInst,

LPVOID lpszAdditional);

Kiểu	Mô tả
WS_MAXIMIZEBOX	Cửa sổ có phím dãn to trên thanh tiêu đề
WS_MINIMIZEBOX	Cửa sổ có phím co nhỏ trên thanh tiêu đề
WS_OVERLAPPED	Cửa sổ maximize và không có cửa sổ cha
WS_SYSMENU	Cửa sổ có hộp thực đơn hệ thống
WS_VSCROLL	Cửa sổ có thanh trượt dọc
WS_HSCROLL	Cửa sổ có thanh trượt ngang

- Gọi hàm ShowWindow()để hiển thị cửa sổ

BOOL ShowWindow (HWND hwnd, int nShow);

với: hwnd chỉ số cửa sổ cần hiển thị.

nShow cách thức hiển thị của cửa sổ, tham số này được nhận giá trị lần đầu tiên của hàm WinMain(), chúng có thể nhận các giá trị sau:

Macro	Cách thức hiển thị
SW_HIDE	Dấu cửa sổ
SW_MINIMIZE	Thu nhỏ cửa sổ
SW_MAXIMIZE	Phóng to cửa sổ toàn màn hình
SW_RESTORE	Trở lại kích thước thông thường

- Để thông báo cho ứng dụng biết là phải vẽ lại vùng làm việc của cửa sổ, ta phải gọi hàm UpdateWindow() yêu cầu Windows gửi thông điệp đến hàm xử lý thông điệp cửa sổ.

❖ Vòng lặp thông điệp

- Khi nhấn phím hay chuột, Windows chuyển đổi sự kiện này thành các thông điệp và đặt vào hàng đợi thông điệp. Vòng lặp thông điệp có nhiệm vụ nhận và xử lý các thông điệp trong hàng đợi.
- TranslateMessage: Dịch thông điệp sang dạng tiêu chuẩn.
- DispatchMessage: Phân phối thông điệp đến hàm xử lý thông điệp tương ứng.

b. Thủ tục xử lý thông điệp

❖ Nhận và xử lý thông điệp của chương trình.

❖ Một chương trình có thể có nhiều thủ tục window.

❖ Một lớp cửa sổ sẽ khai báo 1 thủ tục window.

❖ Các thông điệp sau khi xử lý nên trả về giá trị 0.

❖ Dạng tổng quát:

```
LRESULT CALLBACK WndProc(
    HWND hWnd,        //handle của window nhận message
    UINT message,     //ID của thông điệp (tên thông điệp)
    WPARAM wParam,//thamsố thứ nhất của message (WORD)
    LPARAM lParam)   //thamsố thứ hai của message (LONG)
{
    switch (message)
    {
            case WM_COMMAND:
                return 0;
            case WM_PAINT:
                return 0;
            case WM_DESTROY:
                PostQuitMessage(0);
                return 0;
            default:
                return DefWindowProc(hWnd, message, wParam,
                lParam);
    }
}
```

Thông điệp WM_PAINT:

❖ Cập nhật lại thông tin vẽ trên màn hình.

- Các trạng thái xuất hiện thông điệp WM_PAINT:
  i. Tạo cửa sổ ⟶ Hiển thị ⟶ Cập nhật
  CreateWindow *ShowWindow* *UpdateWindow*
  ii. Xuất hiện hộp thoại (Dialog box), thông báo (Message box) làm che một phần hoặc toàn bộ cửa sổ, khi các hộp thoại này đóng đi thì phải gọi WM_PAINT để vẽ lại cửa sổ.
  iii. Khi thay đổi kích thước cửa sổ
  WS_HREDRAW | WS_VREDRAW
  iv. Cửa sổ đang ở minimize → maximize
- HDC: (Handle to a device context) chỉ đến 1 ngữ cảnh thiết bị gồm thiết bị phần cứng và trình điều khiển thiết bị.
- BeginPaint: Lấy ngữ cảnh thiết bị.
- EndPaint: Giải phóng ngữ cảnh thiết bị.

Thông điệp WM_DESTROY:

- Xuất hiện khi người dùng chọn nút close trên cửa sổ hoặc nhấn Alt+F4.
- Nhiệm vụ PostQuitMessage đặt thông điệp WM_QUIT vào hàng đợi.

10. Tài nguyên của ứng dụng (Resources)

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

Accelerator	Bảng mô tả phím tắt (hot-key).
Bitmap	Ảnh bitmap.
Caret	Con trỏ văn bản.
Cusor	Con trỏ chuột.
Dialog box	Khung hộp thoại.
Enhance metafile	Tập hợp các cấu trúc để lưu ảnh (picture) theo định dạng "độc lập thiết bị" (Device-Independent format).
Font	Font chữ.

Icon	Biểu tượng.
Menu	Menu.
String-table entry	Bảng mô tả các chuỗi ký tự.
Version information	Bảng mô tả thông tin phiên bản.

11. Một số kiểu dữ liệu mới

Stt	Kiểu dữ liệu	Chú thích
1	HANDLE	(*nguyên không dấu 16 bit*) dùng để định danh đối tượng • HWND : window • HMENU : menu • HCURSOR : cursor
2	HBRUSH	(*brush*) mẫu tô: solid, dash, dot, cross, …
3	HPALLETE	(*pallete*) bảng màu
4	HFONT	(*font*) Facename, size, style
5	HBITMAP	*bitmap*
6	HICON	*icon*
7	HPEN	Nét vẽ: solid, dot, dash, size, color
8	HINSTANCE	Instance
9	HDC	Device context
10	LTSTR	(*long pointer string*) con trỏ đến chuỗi ký tự
11	WPARAM LPARAM	(*word*) các tham số đi kèm message.
12	LRESULT	(*long*) kiểu trả về của hàm xử lý Message.
13	LPVOID	Con trỏ đến kiểu dữ liệu bất kỳ.

12. Phân tích, tìm hiểu source code của project

```
1  // bt1.cpp : Defines the entry point for the application.
2  #include "stdafx.h"
3  #include "resource.h"
4  #define MAX_LOADSTRING 100
5  // Global Variables:
6  HINSTANCE hInst;                    // current instance
7  TCHAR szTitle[MAX_LOADSTRING];   // The title bar text
```

```
TCHAR szWindowClass[MAX_LOADSTRING];// The title bar text
// Foward declarations of functions included in this code module:
ATOM                MyRegisterClass(HINSTANCE hInstance);
BOOL                InitInstance(HINSTANCE, int);
LRESULT CALLBACK    WndProc(HWND, UINT, WPARAM, LPARAM);
LRESULT CALLBACK    About(HWND, UINT, WPARAM, LPARAM);
int APIENTRY WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                     LPSTR     lpCmdLine, int       nCmdShow)
{
        // TODO: Place code here.
        MSG msg;
        HACCEL hAccelTable;
        // Initialize global strings
        LoadString(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
        LoadString(hInstance, IDC_BT1, szWindowClass, MAX_LOADSTRING);
        MyRegisterClass(hInstance);
        // Perform application initialization:
        if (!InitInstance (hInstance, nCmdShow))
        {
                return FALSE;
        }
        hAccelTable = LoadAccelerators(hInstance, (LPCTSTR)IDC_BT1);
        // Main message loop:
        while (GetMessage(&msg, NULL, 0, 0))
        {
                if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
                {
                        TranslateMessage(&msg);
                        DispatchMessage(&msg);
                }
        }
        return msg.wParam;
}
//  FUNCTION: MyRegisterClass()
//  PURPOSE: Registers the window class.
//  COMMENTS:
//    This function and its usage is only necessary if you want this code
//    to be compatible with Win32 systems prior to the 'RegisterClassEx'
//    function that was added to Windows 95. It is important to call this function
//    so that the application will get 'well formed' small icons associated
//    with it.
ATOM MyRegisterClass(HINSTANCE hInstance)
{
        WNDCLASSEX wcex;
        wcex.cbSize = sizeof(WNDCLASSEX);
        wcex.style          = CS_HREDRAW | CS_VREDRAW;
```

```
54          wcex.lpfnWndProc   = (WNDPROC)WndProc;
55          wcex.cbClsExtra    = 0;
56          wcex.cbWndExtra    = 0;
57          wcex.hInstance     = hInstance;
58          wcex.hIcon         = LoadIcon(hInstance, (LPCTSTR)IDI_BT1);
59          wcex.hCursor       = LoadCursor(NULL, IDC_ARROW);
60          wcex.hbrBackground      = (HBRUSH)(COLOR_WINDOW+1);
61          wcex.lpszMenuName       = (LPCSTR)IDC_BT1;
62          wcex.lpszClassName      = szWindowClass;
63          wcex.hIconSm       = LoadIcon(wcex.hInstance,(LPCTSTR)IDI_SMALL);
64          return RegisterClassEx(&wcex);
65  }
66  //  FUNCTION: InitInstance(HANDLE, int)
67  //  PURPOSE: Saves instance handle and creates main window
68  //  COMMENTS:
69  //       In this function, we save the instance handle in a global variable and
70  //       create and display the main program window.
71  BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
72  {
73          HWND hWnd;
74          hInst = hInstance; // Store instance handle in our global variable
75          hWnd = CreateWindow(szWindowClass,
76                                  szTitle,
77                                  WS_OVERLAPPEDWINDOW,
78                                  CW_USEDEFAULT,
79                                  0,
80                                  CW_USEDEFAULT,
81                                  0,
82                                  NULL,
83                                  NULL,
84                                  hInstance,
85                                  NULL);
86          if (!hWnd)
87          {
88                  return FALSE;
89           }
90          ShowWindow(hWnd, nCmdShow);
91          UpdateWindow(hWnd);
92           return TRUE;
93  }
94  // FUNCTION: WndProc(HWND, unsigned, WORD, LONG)
95  // PURPOSE:  Processes messages for the main window.
96  // WM_COMMAND- process the application menu
97  // WM_PAINT        - Paint the main window
98  // WM_DESTROY   - post a quit message and return
```

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM
wParam, LPARAM lParam)
{
        int wmId, wmEvent,x,y;
        PAINTSTRUCT ps;
        HDC hdc;
        TCHAR szHello[MAX_LOADSTRING];
        LoadString(hInst, IDS_HELLO, szHello, MAX_LOADSTRING);
        switch (message)
        {
                case WM_COMMAND:
                        wmId    = LOWORD(wParam);
                        wmEvent = HIWORD(wParam);
                        // Parse the menu selections:
                        switch (wmId)
                        {
                                case IDM_ABOUT:
                                        DialogBox(hInst,(LPCTSTR)IDD_ABOUTBOX,
                                        hWnd, (DLGPROC)About);
                                        break;
                                case IDM_EXIT:
                                        DestroyWindow(hWnd);
                                        break;
                                default:
                                        return DefWindowProc(hWnd, message,
                                        wParam, lParam);
                        }
                        break;
                case WM_LBUTTONDOWN:
                        hdc = GetDC(hWnd);
                        // TODO: Add any drawing code here...
                        x=LOWORD(lParam);
                        y=HIWORD(lParam);
                        TextOut(hdc,x,y,(LPCTSTR)szHello, strlen(szHello));
                        break;
                case WM_PAINT:
                        hdc = BeginPaint(hWnd, &ps);
                        // TODO: Add any drawing code here...
                        RECT rt;
                        GetClientRect(hWnd, &rt);
                        DrawText(hdc, szHello, strlen(szHello), &rt, DT_CENTER);
                        EndPaint(hWnd, &ps);
                        break;
                case WM_DESTROY:
                        PostQuitMessage(0);                     break;
                default:
```

```
145                     return DefWindowProc(hWnd, message, wParam, lParam);
146         }
147         return 0;
148 }
149 // Mesage handler for about box.
150 LRESULT CALLBACK About(HWND hDlg, UINT message, WPARAM wParam,
151 LPARAM lParam)
152 {
153         switch (message)
154         {
155                 case WM_INITDIALOG:             return TRUE;
156                 case WM_COMMAND:
157                         if (LOWORD(wParam)== IDOK||LOWORD(wParam)==
158                         IDCANCEL)
159                         {
160                                 EndDialog(hDlg, LOWORD(wParam));return TRUE;
161                         }
162                         break;
163         }
164         return FALSE;
165 }
```

Bài 2: PAINT VÀ REPAINT

Phân bố thời lượng:
- *Số tiết giảng ở lớp: 6 tiết*
- *Số tiết tự học ở nhà: 6 tiết*
- *Số tiết cài đặt chương trình ở nhà: 12 tiết*

1. Giới thiệu

Windows không giữ lại những gì chúng hiển thị trên vùng làm việc của cửa sổ, cho nên chương trình ứng dụng phải hiển thị nội dung cửa sổ khi cần thiết. Vẽ lại nội dung cửa sổ khi:

- ❖ Dùng hàm ScrollWindow: Dữ liệu hiển thị thay đổi → cập nhật lại.
- ❖ Hàm InvalidateRect: Làm bất hợp lệ 1 phần hay toàn bộ vùng làm việc.
- ❖ Menu chương trình bật xuống làm che khuất một phần cửa sổ.
- ❖ Di chuyển chuột, di chuyển icon.

➢ Vùng hình chữ nhật hợp lệ và bất hợp lệ thông qua lời gọi hàm

BOOL InvalidateRect(HWND hwnd, CONST RECT *lpRect, BOOL bErase);

với: bErase = TRUE thì tô lại nền, FALSE thì giữ nguyên.

➔ Hàm BeginPaint() sẽ làm hợp lệ lại vùng bất hợp lệ.

➢ Trong cửa sổ chứa PAINTSTRUCT, mục đích là sẽ tổ hợp lại 2 hay nhiều vùng bất hợp lệ chồng lên nhau.

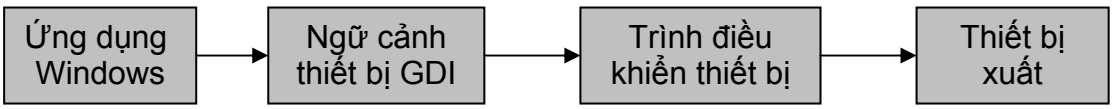
```
typedef struct tagPAINTSTRUCT
        {
                HDC hdc;
                BOOL fErase;
                RECT rcPaint;
                BOOL fRestore;
                BOOL flncUpdate;
                BYTE rgReserved[32];
        }PAINTSTRUCT;
```

Với: *fErase* = { TRUE: xoá vùng hình chữ nhật bất hợp lệ.
FALSE: Không xoá mà ghi chồng lên.

rcPaint chứa toạ độ vùng bất hợp lệ.

```
typedef tagRECT
        {
                LONG left, top;
                LONG right, bottom;
        }RECT;
```

2. Tổng quan về GDI (Graphics Device Interface)



a) Làm việc với ngữ cảnh thiết bị

- hdc chứa các thông tin nền cần thiết cho việc vẽ lên màn hình, tự động giao tiếp với phần cứng.
- Có nhiều cách để nhận và giải phóng hdc.
  - BeginPaint() và EndPaint() : Cặp hàm này chủ yếu được dùng trong phần WM_PAINT.

    HDC BeginPaint(HWND hwnd, LPPAINTSTRUCT lpPS);

    *BOOL EndPaint(HWND hWnd, CONST PAINTSTRUCT *lpPaint);*
  - GetDC() và ReleaseDC() : Không làm hợp lệ bất cứ vùng bất hợp lệ nào.

    HDC GetDC(HWND hwnd);

    int ReleaseDC(HWND hwnd, HDC hdc);

    → *trả về TRUE nếu giải phóng được hdc.*
- Việc lấy và giải phóng hdc chỉ nên được tiến hành bên trong phần xử lý 1 message.
- Ngoài ra, còn có thể nhận về device context của toàn màn hình bằng hàm: hDC = **CreateDC**("DISPLAY", NULL, NULL, NULL);

  Để lấy toạ độ và kích thước của cửa sổ làm việc ta dùng hàm
  BOOL GetClientRect(HWND hWnd, LPRECT lpRect);
  trả về giá trị khác không nếu thành công, ngược lại trả về 0.
- Hiển thị số lên màn hình

  wsprintf(s, "%d + %d= %d", a, b, a+b);

  TextOut(hdc, x, y, s, wsprintf());

b) Chế độ ánh xạ

- ❖ Vị trí hiển thị ký tự TextOut() là tọa độ tương đối trong cửa sổ (tọa độ logic).
- ❖ Windows sẽ ánh xạ đơn vị này thành pixel khi hiển thị ký tự.
- ❖ Ở chế độ mặc định tọa độ logic ≈ pixel.

c) Mô hình màu RGB (Red – Green – Blue)

Byte 3	Byte 2	Byte 1	Byte 0
0	Blue	Green	Red

- ❖ Có giá trị từ 0 – 255

  $(0, 0, 0)_{\text{đen}} \rightarrow (255, 255, 255)_{\text{trắng}}$
- ❖ Các hàm API liên quan đến màu đều sử dụng mô hình RGB.
- ❖ Định nghĩa màu COLORREF RGB (int red, int green, int blue).

Ví dụ 1 : Vẽ hình chữ nhật

```
HDC hDC;
HPEN hPen, oldHPen;
hDC=GetDC(hWnd);
hPen=CreatePen(PS_SOLID, 5, RGB(0, 0, 255));
oldHPen=(HPEN)SelectObject(hDC, hPen);
Rectangle(hDC, 20, 20, 100, 100);
SelectObject(hDC, oldHPen);
DeleteObject(hPen);
ReleaseDC(hWnd, hDC);
```

d) Tạo lập và giải phóng memory device context

- ❖ Memory device context (MDC) là một device context ảo không gắn với một thiết bị xuất cụ thể nào. Muốn kết quả kết xuất ra thiết bị vật lý ta phải chép MDC lên một device context thật sự(device context có liên kết với thiết bị vật lý). MDC thường được dùng như một device context trung gian để vẽ trước khi thực sự xuất ra thiết bị, nhằm giảm sự chớp giật nếu thiết bị xuất là window hay màn hình.
- ❖ Để tạo MDC tương thích với một hDC cụ thể, sử dụng hàm CreateCompatibleDC:

  ```
  HDC hMemDC;
  hMemDC = CreateCompatibleDC(hDC);
  ```
- ❖ Đơn giản hơn, có thể đặt NULL vào vị trí hDC, Windows sẽ tạo một device context tương thích với màn hình.

- Hủy MDC bằng hàm DeleteDC.
- MDC có bề mặt hiển thị như một thiết bị thật. Tuy nhiên, bề mặt hiển thị này lúc đầu rất nhỏ, chỉ là một pixel đơn sắc. Không thể làm gì với một bề mặt hiển thị chỉ gồm 1 bit như vậy. Do đó cần làm cho bề mặt hiển thị này rộng hơn bằng cách chọn một đối tượng bitmap GDI vào MDC:

  SelectObject(hMemDC, hBitmap);
- Chỉ có thể chọn đối tượng bitmap vào MDC, không thể chọn vào một device context cụ thể được.
- Sau khi chọn một đối tượng bitmap cho MDC, có thể dùng MDC như một device context thật sự.
- Sau khi được hoàn tất trong MDC, ảnh được đưa ra device context thật sự bằng hàm BitBlt:

  BitBlt(hDC, xDest, yDest, nWidth, nHeight, hMemDC, xSource, ySource);
- Ví dụ : Chuẩn bị ảnh trước khi đưa ra màn hình, tránh gây chớp màn hình trong thông điệp WM_PAINT.

```
case WM_PAINT:
hdc = BeginPaint(hWnd, &ps);
// Lấy về kích thước vùng client của cửa sổ hiện hành
RECT rect;
GetClientRect(hWnd, &rect);
// Tạo MDC tương thích với DC của cửa sổ
HDC hMemDC;
hMemDC = CreateCompatibleDC(hdc);
// Chọn một đối tượng bitmap để mở rộng vùng hiển thị cho MDC
HBITMAP bitmap,oBitmap;
bitmap = CreateCompatibleBitmap(hdc, rect.right, rect.bottom);
oBitmap = (HBITMAP)SelectObject(hMemDC, bitmap);
// Vẽ lại nền MDC
FillRect(hMemDC, &rect, HBRUSH (GetBkColor(hMemDC)));
// Xuất hình ảnh, text ra MDC
SetPixel(hMemDC, 0, 0, RGB(255,0,0));
MoveToEx(hMemDC, 50, 50, NULL);
LineTo(hMemDC, 100, 100);
Rectangle(hMemDC, 10, 10, 100, 100);
TextOut(hMemDC, 15 ,15, "Testing MDC", 11);
```

```
If (!BitBlt(hdc, 0, 0, rect.right, rect.bottom, hMemDC, 0, 0,
SRCCOPY))
MessageBox(hWnd, "Failed to transfer bit block", "Error",MB_OK);
// Phục hồi lại bitmap cũ cho MDC
SelectObject(hMemDC, oBitmap);
// Giải phóng MDC, bitmap đã tạo
DeleteDC(hMemDC);
DeleteObject(bitmap);
EndPaint(hWnd, &ps);
break;
```

3. Một số hàm đồ họa cơ sở

a) Nhóm hàm vẽ

- **COLORREF GetPixel(HDC** hDC, **int** nXPos, **int** nYPos);

  Lấy về giá trị màu tại vị trí (nXPos, nYPos) của hDC, trả về -1 nếu điểm này nằm ngoài vùng hiển thị.

- **COLORREF SetPixel(HDC** hDC, **int** nXPos, **int** nYPos, **COLORREF** clrRef);

  Vẽ một điểm màu clrRef tại vị trí (nXPos, nYPos) lên hDC. Giá trị trả về là màu của điểm (nXPos, nYPos) hoặc -1 nếu điểm này nằm ngoài vùng hiển thị.

- **DWORD MoveToEx(HDC** hDC, **int** x, **int** y);

  Di chuyển bút vẽ đến tọa độ (x, y) trên hDC. Giá trị trả về là tọa độ cũ của bút vẽ, x = LOWORD, y = HIWORD.

- **BOOL LineTo(HDC** hDC, **int** xEnd, **int** yEnd);

  Vẽ đoạn thẳng từ vị trí hiện hành đến vị trí (xEnd, yEnd) trên hDC. Hàm trả về TRUE nếu thành công, FALSE nếu thất bại.

- **BOOL Polyline(HDC** hDC, **const POINT FAR** *lpPoints, **int** nPoints);

  Vẽ đường gấp khúc lên hDC bằng các đoạn thẳng liên tiếp, số đỉnh là nPoints với tọa độ các đỉnh được xác định trong lpPoints. Hàm trả về TRUE nếu thành công, FALSE nếu thất bại.

- **BOOL Polygon(HDC** hDC, **const POINT FAR** *lpPoints, **int** nPoints);

  Vẽ đa giác có nPoints đỉnh, tọa độ các đỉnh được xác định bởi lpPoints. Hàm trả về TRUE nếu thành công, FALSE nếu thất bại.

- **BOOL Rectangle(HDC** hDC, **int** left, **int** top, **int** right, **int** bottom);
  Vẽ hình chữ nhật có tọa độ là left, top, right, bottom lên hDC.
- **HPEN CreatePen(int** penStyle, **int** penWidth, **COLORREF** penColor);
  Tạo bút vẽ có kiểu penStyle, độ dày nét vẽ là penWidth, màu penColor. Hàm trả về handle của bút vẽ nếu thành công và trả về NULL nếu thất bại. Các giá trị của penStyle như sau :

Giá trị	Giải thích
PS_SOLID	
PS_DASH	
PS_DOT	
PS_DASHDOT	
PS_DASHDOTDOT	
PS_NULL	Không hiển thị
PS_INSIDEFRAME	

Các kiểu bút vẽ penStyle

Ví dụ : Tạo bút vẽ mới và dùng bút vẽ này vẽ một số đường cơ sở.

```
HDC hDC;
POINT PointArr[3];
HPEN hPen, hOldPen;
hDC = GetDC(hWnd);
PointArr[0].x = 50;
PointArr[0].y = 10;
PointArr[1].x = 250;
PointArr[1].y = 50;
PointArr[2].x = 125;
PointArr[2].y = 130;
Polyline(hDC, PointArr, 3);
hPen = (HPEN)CreatePen(PS_SOLID, 1, RGB(0, 0, 255));
hOldPen = SelectObject(hDC, hPen);
MoveToEx(hDC, 100, 100, NULL);
LineTo(hDC, 200, 150);
SelectObject(hDC, hOldPen);
DeleteObject(hPen);
ReleaseDC(hWnd, hDC);
```

b) Nhóm hàm miền

- **HBRUSH CreateSolidBrush(COLORREF cRef);**

  Tạo mẫu tô đặc với màu cRef.

- **HBRUSH CreateHatchBrush**(**int** bStyle, **COLORREF** cRef);

  Tạo mẫu tô dạng lưới kiểu bStyle với màu cRef.
  Các kiểu bStyle :

HS_HORIZONTAL	HS_BDIAGONAL
HS_VERTICAL	HS_CROSS
HS_FDIAGONAL	HS_DIAGCROSS

- **BOOL FloodFill**(**HDC** hDC, **int** xStart, **int** yStart, **COLORREF** cRef);

  Tô màu một vùng kín, màu đường biên là cRef.
- **BOOL ExtFloodFill**(**HDC** hDC, **int** xStart, **int** yStart, **COLORREF** cRef, **UINT** fillStyle);
  Tô màu một vùng kín, fillStyle quyết định cách tô :
  o FLOODFILLBORDER : Tô màu vùng có màu đường biên là cRef.
  o FLOODFILLSURFACE : Tô vùng có màu cRef.

Ví dụ : Sử dụng các mẫu có sẵn và tạo các mẫu tô mới để tô.

```
HDC hDC;
HPEN hPen;
HBRUSH hBrush, hOldBrush;
hDC = GetDC(hWnd);
//Vẽ hai hình chữ nhật với bút vẽ Black
hPen = (HPEN)GetStockObject(BLACK_PEN);
SelectObject(hDC, hPen);
Rectangle(hDC, 10, 10, 50, 50);
Rectangle(hDC, 100, 100, 200, 200);
// Dùng một trong các mẫu tô có sẵn để tô hình
hBrush = (HBRUSH)GetStockObject(GRAY_BRUSH);
SelectObject(hDC, hBrush);
FloodFill(hDC, 30, 30, RGB(0,0,255));
// Tạo mẫu tô mới để tô hình thứ hai
hBrush = (HBRUSH)CreateHatchBrush(HS_DIAGCROSS, RGB(0, 255, 255));
hOldBrush = (HBRUSH)SelectObject(hDC, hBrush);
FloodFill(hDC, 150, 150, RGB(0, 0, 0));
SelectObject(hDC, hOldBrush);
//Xóa mẫu tô và giải phóng hDC
DeleteObject(hBrush);
ReleaseDC(hWnd, hDC);
```

4. Kết luận

WM_PAINT là message có độ ưu tiên thấp. Khi WM_PAINT trong hàng chờ và có một số Window Message khác thì Windows xử lý WM khác rồi mới xử lý WM_PAINT.

Bài 3: CÁC THIẾT BỊ NHẬP LIỆU

Phân bố thời lượng:
- *Số tiết giảng ở lớp: 15 tiết*
- *Số tiết tự học ở nhà: 15 tiết*
- *Số tiết cài đặt chương trình ở nhà: 30 tiết*

1. Bàn phím

a. Chương trình điều khiển bàn phím (Keyboard.drv)

Windows được nạp Keyboard.drv khi khởi động và xử lý phím. Sau đó keyboard.drv chuyển cho USER biến phím nhấn thành message và đưa vào hàng đợi (Hàng đợi hệ thống và hàng đợi chương trình).

b. Cửa sổ có focus

- ❖ Khi cửa sổ có focus thì phát sinh thông điệp WM_SETFOCUS.
- ❖ Ngược lại phát sinh WM_KILLFOCUS.

c. Thông điệp phím

```
MSG msg;
while(GetMessage(&msg,NULL,0,0))
{
        TranslateMessage(&msg);
        DispatchMessage(&msg);
}
```

Thông điệp	Nguyên nhân phát sinh
WM_ACTIVATE	Thông điệp này cùng được gởi đến các cửa sổ bị kích hoạt và cửa sổ không bị kích hoạt. Nếu các cửa sổ này cùng một hàng đợi nhập liệu, các thông điệp này sẽ được truyền một cách đồng bộ, đầu tiên thủ tục Windows của cửa sổ trên cùng bị mất kích hoạt, sau đó đến thủ tục của cửa sổ trên cùng được kích hoạt. Nếu các cửa sổ này không nằm trong cùng một hàng đợi thì thông điệp sẽ được gởi một cách không đồng bộ, do đó cửa sổ sẽ được kích hoạt ngay lập tức.
WM_APPCOMMAND	Thông báo đến cửa sổ rằng người dùng đã tạo một sự kiện lệnh ứng dụng, ví dụ khi người dùng kích vào button sử dụng chuột hay đánh vào một kí tự kích hoạt một lệnh của ứng dụng.

WM_CHAR	Thông điệp này được gởi tới cửa sổ có sự quan tâm khi thông điệp WM_KEYDOWN đã được dịch từ hàm TranslateMessage. Thông điệp WM_CHAR có chứa mã kí tự của phím được nhấn.
WM_DEADCHAR	Thông điệp này được gởi tới cửa sổ có sự quan tâm khi thông điệp WM_KEYUP đã được xử lý từ hàm TranslateMessage. Thông điệp này xác nhận mã kí tự khi một phím dead key được nhấn. Phím dead key là phím kết hợp để tạo ra kí tự ngôn ngữ không có trong tiếng anh (xuất hiện trong bàn phím hỗ trợ ngôn ngữ khác tiếng Anh).
WM_GETHOTKEY	Ứng dụng gởi thông điệp này để xác định một phím nóng liên quan đến một cửa sổ. Để gởi thông điệp này thì dùng hàm SendMessage.
WM_HOTKEY	Thông điệp này được gởi khi người dùng nhấn một phím nóng được đăng kí trong RegisterHotKey.
WM_KEYDOWN	Thông điệp này được gởi cho cửa sổ nhận được sự quan tâm khi người dùng nhấn một phím trên bàn phím. Phím này không phải phím hệ thống (Phím không có nhấn phím Alt).
WM_KEYUP	Thông điệp này được gởi cho cửa sổ nhận được sự quan tâm khi người dùng nhả một phím đã được nhấn trước đó.Phím này không phải phím hệ thống (Phím không có nhấn phím Alt).
WM_KILLFOCUS	Thông điệp này được gởi tới cửa sổ đang nhận được sự quan tâm trước khi nó mất quyền này.
WM_SETFOCUS	Thông điệp này được gởi tới cửa sổ sau khi cửa sổ nhận được sự quan tâm của Windows
WM_SETHOTKEY	Ứng dụng sẽ gởi thông điệp này đến cửa sổ liên quan đến phím nóng, khi người dùng nhấn một phím nóng thì cửa sổ tương ứng liên quan tới phím nóng này sẽ được kích hoạt.
WM_SYSCHAR	Thông điệp này sẽ được gởi tới cửa sổ nhận được sự quan tâm khi hàm TranslateMesage xử lý xong thông điệp WM_SYSKEYDOWN.

	Thông điệp WM_SYSCHAR chứa mã cửa phím hệ thống. Phím hệ thống là phím có chứa phím Alt và tổ hợp phím khác.
WM_SYSDEADCHAR	Thông điệp này được gởi tới cửa sổ nhận được sự quan tâm khi một thông điệp WM_SYSKEYDOWN được biên dịch trong hàm TranslateMessage. Thông điệp này xác nhận mã kí tự của phím hệ thống deadkey được nhấn.
WM_SYSKEYDOWN	Thông điệp này được gởi tới cửa sổ nhận được sự quan tâm khi người dùng nhấn phím hệ thống.

d. Ví dụ

```
1   #define BUFSIZE 65535
2   #define SHIFTED 0x8000
3
4   LONG APIENTRY MainWndProc(HWND hwndMain, UINT uMsg,
5   WPARAM wParam, LPARAM lParam)
6   {
7      HDC hdc;                // handle to device context
8      TEXTMETRIC tm;          // structure for text metrics
9      static DWORD dwCharX;   // average width of characters
10     static DWORD dwCharY;   // height of characters
11     static DWORD dwClientX; // width of client area
12     static DWORD dwClientY; // height of client area
13     static DWORD dwLineLen; // line length
14     static DWORD dwLines;   // text lines in client area
15     static int nCaretPosX = 0; // horizontal position of caret
16     static int nCaretPosY = 0; // vertical position of caret
17     static int nCharWidth = 0; // width of a character
18     static int cch = 0;     // characters in buffer
19     static int nCurChar = 0; // index of current character
20     static PTCHAR pchInputBuf; // input buffer
21     int i, j;               // loop counters
22     int cCR = 0;            // count of carriage returns
23     int nCRIndex = 0;       // index of last carriage return
24     int nVirtKey;           // virtual-key code
25     TCHAR szBuf[128];       // temporary buffer
26     TCHAR ch;               // current character
27     PAINTSTRUCT ps;         // required by BeginPaint
28     RECT rc;                // output rectangle for DrawText
29     SIZE sz;                // string dimensions
30     COLORREF crPrevText;    // previous text color
```

```
31          COLORREF crPrevBk;        // previous background color
32           switch (uMsg)
33           {
34                case WM_CREATE:
35                 // Get the metrics of the current font.
36                 hdc = GetDC(hwndMain);
37                GetTextMetrics(hdc, &tm);
38                ReleaseDC(hwndMain, hdc);
39                 // Save the average character width and height.
40                 dwCharX = tm.tmAveCharWidth;
41                dwCharY = tm.tmHeight;
42                 // Allocate a buffer to store keyboard input.
43                 pchInputBuf = (LPTSTR) GlobalAlloc(GPTR,
44                   BUFSIZE * sizeof(TCHAR));
45                return 0;
46              case WM_SIZE:
47                 // Save the new width and height of the client area.
48                 dwClientX = LOWORD(lParam);
49                dwClientY = HIWORD(lParam);
50                 // Calculate the maximum width of a line and the
51                // maximum number of lines in the client area.
52                dwLineLen = dwClientX - dwCharX;
53                dwLines = dwClientY / dwCharY;
54                break;
55             case WM_SETFOCUS:
56                 // Create, position, and display the caret when the
57                // window receives the keyboard focus.
58                CreateCaret(hwndMain, (HBITMAP) 1, 0, dwCharY);
59                SetCaretPos(nCaretPosX, nCaretPosY * dwCharY);
60                ShowCaret(hwndMain);
61                break;
62              case WM_KILLFOCUS:
63                 // Hide and destroy the caret when the window loses the
64                // keyboard focus.
65                HideCaret(hwndMain);
66                DestroyCaret();
67                break;
68              case WM_CHAR:
69                switch (wParam)
70                {
71                   case 0x08:  // backspace
72                   case 0x0A:  // linefeed
73                   case 0x1B:  // escape
74                      MessageBeep((UINT) -1);
75                      return 0;
76                    case 0x09:  // tab
```

```
77                      // Convert tabs to four consecutive spaces.
78                      for (i = 0; i < 4; i++)
79                         SendMessage(hwndMain, WM_CHAR, 0x20, 0);
80                      return 0;
81                    case 0x0D:  // carriage return
82                      // Record the carriage return and position the
83                      // caret at the beginning of the new line.
84                      pchInputBuf[cch++] = 0x0D;
85                      nCaretPosX = 0;
86                      nCaretPosY += 1;
87                      break;
88                    default:    // displayable character
89                      ch = (TCHAR) wParam;
90                      HideCaret(hwndMain);
91                      // Retrieve the character's width and output
92                      // the character.
93                      hdc = GetDC(hwndMain);
94                      GetCharWidth32(hdc, (UINT) wParam, (UINT) wParam,
95                         &nCharWidth);
96                      TextOut(hdc, nCaretPosX, nCaretPosY * dwCharY,
97                         &ch, 1);
98                      ReleaseDC(hwndMain, hdc);
99                      // Store the character in the buffer.
100                     pchInputBuf[cch++] = ch;
101                     // Calculate the new horizontal position of the
102                     // caret. If the position exceeds the maximum,
103                     // insert a carriage return and move the caret
104                     // to the beginning of the next line.
105                     nCaretPosX += nCharWidth;
106                     if ((DWORD) nCaretPosX > dwLineLen)
107                     {
108                        nCaretPosX = 0;
109                        pchInputBuf[cch++] = 0x0D;
110                        ++nCaretPosY;
111                     }
112                     nCurChar = cch;
113                     ShowCaret(hwndMain);
114                     break;
115                 }
116                 SetCaretPos(nCaretPosX, nCaretPosY * dwCharY);
117                 break;
118               case WM_KEYDOWN:
119                 switch (wParam)
120                 {
121                    case VK_LEFT:   // LEFT ARROW
122                       // The caret can move only to the beginning of
```

```
123                     // the current line.
124                      if (nCaretPosX > 0)
125                      {
126                         HideCaret(hwndMain);
127                          // Retrieve the character to the left of
128                         // the caret, calculate the character's
129                         // width, then subtract the width from the
130                         // current horizontal position of the caret
131                         // to obtain the new position.
132                          ch = pchInputBuf[--nCurChar];
133                         hdc = GetDC(hwndMain);
134                         GetCharWidth32(hdc, ch, ch, &nCharWidth);
135                         ReleaseDC(hwndMain, hdc);
136                         nCaretPosX = max(nCaretPosX - nCharWidth, 0);
137                         ShowCaret(hwndMain);
138                      }
139                      break;
140                   case VK_RIGHT:  // RIGHT ARROW
141                      // Caret moves to the right or, when a carriage
142                      // return is encountered, to the beginning of
143                      // the next line.
144                      if (nCurChar < cch)
145                      {
146                         HideCaret(hwndMain);
147                          // Retrieve the character to the right of
148                         // the caret. If it's a carriage return,
149                         // position the caret at the beginning of
150                         // the next line.
151                          ch = pchInputBuf[nCurChar];
152                         if (ch == 0x0D)
153                         {
154                            nCaretPosX = 0;
155                            nCaretPosY++;
156                         }
157                          // If the character isn't a carriage
158                         // return, check to see whether the SHIFT
159                         // key is down. If it is, invert the text
160                         // colors and output the character.
161                          else
162                         {
163                            hdc = GetDC(hwndMain);
164                            nVirtKey = GetKeyState(VK_SHIFT);
165                            if (nVirtKey & SHIFTED)
166                            {
167                               crPrevText = SetTextColor(hdc,
168                                  RGB(255, 255, 255));
```

```
169                         crPrevBk = SetBkColor(hdc,
170                            RGB(0,0,0));
171                         TextOut(hdc, nCaretPosX,
172                            nCaretPosY * dwCharY,
173                            &ch, 1);
174                         SetTextColor(hdc, crPrevText);
175                         SetBkColor(hdc, crPrevBk);
176                      }
177                       // Get the width of the character and
178                      // calculate the new horizontal position of the caret.
179                       GetCharWidth32(hdc, ch, ch, &nCharWidth);
180                      ReleaseDC(hwndMain, hdc);
181                      nCaretPosX = nCaretPosX + nCharWidth;
182                   }
183                   nCurChar++;
184                   ShowCaret(hwndMain);
185                   break;
186                }
187                break;
188             case VK_UP:     // UP ARROW
189             case VK_DOWN:   // DOWN ARROW
190                MessageBeep((UINT) -1);
191                return 0;
192              case VK_HOME:   // HOME
193                 // Set the caret's position to the upper left
194                // corner of the client area.
195                nCaretPosX = nCaretPosY = 0;
196                nCurChar = 0;
197                break;
198              case VK_END:    // END
199                 // Move the caret to the end of the text.
200                 for (i=0; i < cch; i++)
201                 {
202                    // Count the carriage returns and save the
203                    // index of the last one.
204                     if (pchInputBuf[i] == 0x0D)
205                     {
206                        cCR++;
207                        nCRIndex = i + 1;
208                     }
209                }
210                nCaretPosY = cCR;
211
212                // Copy all text between the last carriage
213                // return and the end of the keyboard input
214                // buffer to a temporary buffer.
```

```
                    for (i = nCRIndex, j = 0; i < cch; i++, j++)
                      szBuf[j] = pchInputBuf[i];
                    szBuf[j] = TEXT('\0');
                    // Retrieve the text extent and use it
                    // to set the horizontal position of the
                    // caret.
                    hdc = GetDC(hwndMain);
                    GetTextExtentPoint32(hdc, szBuf, lstrlen(szBuf), &sz);
                    nCaretPosX = sz.cx;
                    ReleaseDC(hwndMain, hdc);
                    nCurChar = cch;
                    break;
                  default:
                    break;
                }
                SetCaretPos(nCaretPosX, nCaretPosY * dwCharY);
                break;
            case WM_PAINT:
                if (cch == 0)       // nothing in input buffer
                   break;
                hdc = BeginPaint(hwndMain, &ps);
                HideCaret(hwndMain);
                // Set the clipping rectangle, and then draw the text
                // into it.
                SetRect(&rc, 0, 0, dwLineLen, dwClientY);
                DrawText(hdc, pchInputBuf, -1, &rc, DT_LEFT);
                ShowCaret(hwndMain);
                EndPaint(hwndMain, &ps);
                break;
                // Process other messages.
            case WM_DESTROY:
                PostQuitMessage(0);
                // Free the input buffer.
                GlobalFree((HGLOBAL) pchInputBuf);
                UnregisterHotKey(hwndMain, 0xAAAA);
                break;
             default:
                return DefWindowProc(hwndMain, uMsg, wParam, lParam);
        }
        return NULL;
    }
```

2. Thiết bị chuột

a. Kiểm tra thiết bị chuột

```
int GetSystemMetrics(
```

```
  int nIndex   // system metric or configuration setting
);
```

fMouse = GetSystemMetrics(SM_MOUSEPRESENT);

Giá trị trả về fMouse là TRUE (1) nếu có thiết bị chuột được cài đặt, và ngược lại bằng FALSE (0) nếu thiết bị chuột không được cài đặt vào máy.

b. Trong lớp cửa sổ ta định nghĩa con trỏ chuột cho ứng dụng

wndclass.hCursor = **LoadCursor** (NULL, IDC_ARROR);

wndclass.style = CS_HREDRAW|CS_VREDRAW|CS_DBLCLKS;

Với thiết bị chuột ta có thể có các hành động như sau:

- *Kích chuột* : nhấn và thả một nút chuột.
- *Kích đúp chuột* : nhấn và thả chuột nhanh (nhấn 2 lần nhanh).
- *Kéo* : di chuyển chuột trong khi vẫn nắm giữ một nút.

c. Thông điệp chuột trong vùng làm việc

Nút	Nhấn	Thả	Nhấn đúp
Trái	WM_LBUTTONDOWN	WM_LBUTTONUP	WM_LBUTTONDBLCLK
Giữa	WM_MBUTTONDOWN	WM_MBUTTONUP	WM_MBUTTONDBLCLK
Phải	WM_RBUTTONDOWN	WM_MBUTTONUP	WM_RBUTTONDBLCLK

d. Giá trị wParam sẽ cho biết trạng thái của nút nhấn, phím Shift, và phím Ctrl.

MK_LBUTTON	Nút chuột trái nhấn
MK_MBUTTON	Nút chuột giữa nhấn
MK_RBUTTON	Nút chuột phải nhấn
MK_SHIFT	Phím Shift được nhấn
MK_CONTROL	Phím Ctrl được nhấn

e. Giá trị lParam sẽ cho biết vị trí chuột tại thời điểm phát sinh message.

- *2 bytes thấp: tọa độ x*
- *2 bytes cao: tọa độ y*

f. Ví dụ

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT message,
WPARAM wParam, LPARAM lParam)
{
    HDC hdc;
    static POINT oldPoint;
    static int iC;
    int WIDTH_PEN = 2;
    HPEN oPen,pen;
    COLORREF Col [ ] ={ RGB (0, 0, 0) , RGB (255 ,0 ,0),
    RGB (0, 255, 0), RGB (0, 0, 255), RGB (255, 255, 0)};
    POINT point;
    TCHAR str [255];
    switch ( message ) // Xử lý thông điệp
    {
        case WM_LBUTTONDOWN:
        /* Vẽ đường thẳng từ vị trí trước đó đến vị trí chuột hiện tại*/
            hdc = GetDC ( hWnd );
            pen = CreatePen ( PS_SOLID,WIDTH_PEN,Col [
            iC] );
            oPen = ( HPEN ) SelectObject ( hdc,pen );
            point.x = LOWORD ( lParam );
            point.y = HIWORD ( lParam );
            MoveToEx ( hdc, oldPoint.x, oldPoint.y, NULL );
            LineTo ( hdc, point.x, point.y );
            oldPoint = point;
            /* Chọn lại bút vẽ trước đó và hủy bút vẽ vừa tạo*/
            SelectObject ( hdc, oPen );
            DeleteObject ( pen );
            ReleaseDC ( hWnd, hdc );
            break;
        case WM_RBUTTONDOWN:
        /* Chuyển index của bảng màu sang vị trí tiếp theo, nếu
        cuối bảng màu thì quay lại màu đầu tiên*/
            iC = ( iC+1 ) % ( sizeof ( Col ) / sizeof (
            COLORREF ) );
            break;
        case WM_MOUSEMOVE:
        /* Xuất toạ độ chuột hiện thời lên thanh tiêu đề*/
            sprintf ( str,"Toa do chuot x = %d, To do y = %d",
            LOWORD(lParam), HIWORD(lParam));
            SetWindowText ( hWnd, str );
            /* Kiểm tra xem có giữ phím chuột trái hay không*/
            if ( wParam & MK_LBUTTON )
            {
                hdc = GetDC ( hWnd );
```

```
46                                  pen = CreatePen (
47                                  PS_SOLID,WIDTH_PEN,Col [ iC ] );
48                                  oPen = ( HPEN ) SelectObject ( hdc, pen );
49                                  point.x = LOWORD ( lParam );
50                                  point.y = HIWORD ( lParam );
51                                  MoveToEx ( hdc, oldPoint.x, oldPoint.y,
52                                  NULL );
53                                  LineTo ( hdc, point.x, point.y );
54                                  oldPoint = point;
55                                  SelectObject ( hdc, oPen );
56                                  DeleteObject ( pen );
57                                  ReleaseDC ( hWnd, hdc );
58                          }
59                          break;
60                  case WM_DESTROY:
61                          PostQuitMessage ( 0 );
62                          break;
63                  default:
64                          return DefWindowProc ( hWnd, message, wParam,
65                          lParam );
66          }
67          return 0;
68  }
```

3. Timer

a. Khởi tạo

```
UINT_PTR SetTimer( HWND hWnd, UINT_PTR nIDEvent, UINT uElapse, TIMERPROC lpTimerFunc );
```

- hWnd : Định danh của cửa sổ khai báo dùng bộ định thời gian.
- nIDEvent : Định danh của bộ định thời gian.
- nElapse : Là khoảng thời gian nghỉ giữa hai lần gởi thông điệp
- lpTimerFunc : Hàm sẽ xử lý khi thông điệp WM_TIMER phát sinh, nếu chúng ta khai báo là NULL thì Windows sẽ gởi thông điệp WM_TIMER vào hàng đợi thông điệp của cửa sổ tương ứng.

b. Hủy

```
BOOL KillTimer( HWND hWnd, UINT_PTR uIDEvent );
```

- hWnd : Định danh của cửa sổ dùng bộ định thời gian
- uIDEvent : Định danh của bộ định thời gian.

c. Ví dụ 1

```
1   #include <time.h>
```

```
#include "stdio.h"
#define MAX_POINT 10000
#define IDT_TIMER1 1
LRESULT CALLBACK WndProc(HWND hWnd, UINT message,
WPARAM wParam, LPARAM lParam)
{
    PAINTSTRUCT ps;
    HDC hdc;
    static int NumCir = 0;
    static POINT point [ MAX_POINT ];
    int r = 5, i;
    HPEN pen, oldPen;
    RECT rc;
    TCHAR str [255];
    /* Xử lý thông điệp*/
    switch ( message )
    {
        case WM_CREATE:
            SetTimer(hWnd, IDT_TIMER1, 500,
            (TIMERPROC) NULL);
            srand ( (unsigned) time( NULL ) );
            break;
        case WM_PAINT:
            hdc = BeginPaint ( hWnd, &ps );
            pen = CreatePen ( PS_SOLID, 2, RGB (255,0,0) );
            oldPen = (HPEN) SelectObject ( hdc, pen );
            for( i=0; i < NumCir; i++ )
                Arc ( hdc, point[i].x-r, point[i].y-r,
                point[i].x+r, point[i].y+r, point[i].x+r,
                point[i].y,point[i].x+r,point[i].y);
            SelectObject ( hdc, oldPen );
            DeleteObject ( pen );
            EndPaint ( hWnd, &ps );
            break;
        case WM_TIMER:
            GetClientRect ( hWnd, &rc );
            point [NumCir].x = rand( ) % (rc.right - rc.left);
            point [NumCir].y = rand( ) % (rc.bottom - rc.top);
            NumCir++;
            sprintf ( str,"So vong tron : %d", NumCir);
            SetWindowText ( hWnd, str );
            InvalidateRect ( hWnd, &rc, FALSE);
            break;
        case WM_DESTROY:
            KillTimer ( hWnd, IDT_TIMER1 );
            PostQuitMessage ( 0 );
```

```
                    break;
                default:
                    return DefWindowProc ( hWnd, message, wParam,
                    lParam );
            }
            return 0;
        }
```

d. Ví dụ 2

```
#include <time.h>
#include "stdio.h"
#define IDT_TIMER1 1
LRESULT CALLBACK WndProc(HWND hWnd, UINT message,
WPARAM wParam, LPARAM lParam)
{
    PAINTSTRUCT ps;
    HDC hdc;
    /* Khai báo biến lưu các giá trị không gian*/
    struct tm *newtime;
    time_t CurTime;
    TCHAR str [255];
    RECT rc;
    /* Biến LOGFONT để tạo font mới*/
    LOGFONT lf;
    HFONT oldFont, font;
    COLORREF color = RGB (255, 0, 0), oldColor;
    switch ( message )
    {
    case WM_CREATE:
    /* khởi tạo bộ định thời gian, và khai báo hàm xử lý Timer*/
    SetTimer ( hWnd, IDT_TIMER1, 1000, ( TIMERPROC )
    TimerProc );
    break;
    case WM_PAINT:
        hdc = BeginPaint ( hWnd, &ps );
        time( &CurTime );
        newtime = localtime ( &CurTime );
        GetClientRect ( hWnd, &rc );
        sprintf(str,"Gio hien tai : %d gio: %d phut: %d giay",
        newtime->tm_hour,newtime->tm_min, newtime-
        >tm_sec);
        oldColor = SetTextColor ( hdc, color );
        memset ( &lf, 0, sizeof ( LOGFONT ) );
        lf.lfHeight = 50;
        strcpy ( lf.lfFaceName, "Tahoma" );
```

```
37                      font = CreateFontIndirect ( &lf );
38                      oldFont = ( HFONT ) SelectObject ( hdc,font );
39                      DrawText ( hdc, str, strlen(str), &rc, DT_CENTER |
40                      DT_VCENTER | DT_SINGLELINE );
41                      SetTextColor ( hdc,oldColor );
42                      SelectObject ( hdc,oldFont );
43                      DeleteObject ( font );
44                      EndPaint ( hWnd, &ps );
45                      break;
46              case WM_DESTROY:
47                      PostQuitMessage ( 0 );
48                      break;
49              default:
50                      return DefWindowProc ( hWnd, message, wParam,
51                      lParam );
52              }
53              return 0;
54      }
55      VOID CALLBACK TimerProc( HWND hwnd, UINT uMsg,
56      UINT_PTR idEvent, DWORD dwTime)
57      {
58              RECT rc;
59              GetClientRect ( hwnd, &rc );
60              InvalidateRect ( hwnd, &rc, TRUE );
61      }
```

Bài 4: HỘP THOẠI VÀ ĐIỀU KHIỂN

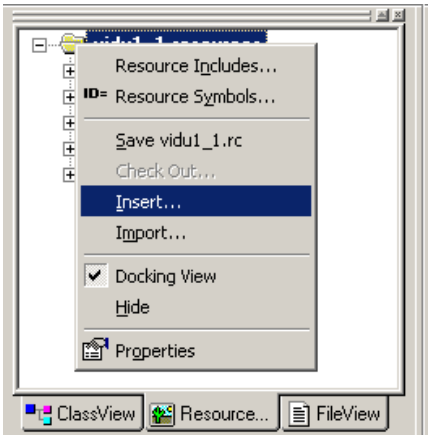
Phân bố thời lượng:
- *Số tiết giảng ở lớp: 12 tiết*
- *Số tiết tự học ở nhà: 12 tiết*
- *Số tiết cài đặt chương trình ở nhà: 24 tiết*

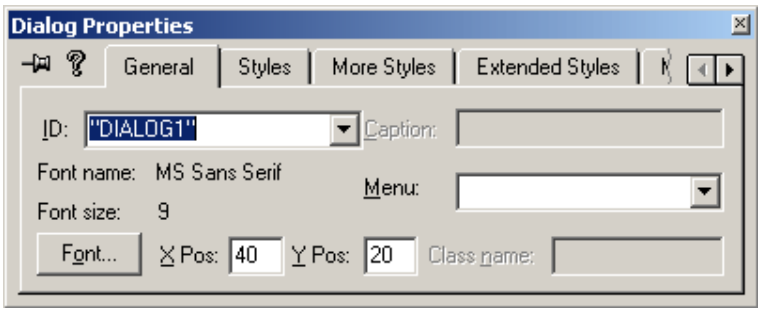
1. Hộp thoại

Hộp thoại phối hợp giữa người sử dụng với chương trình bằng một số phần tử điều khiển mà các phần tử này nhận nhiệm vụ thu nhận thông tin từ người dùng và cung cấp thông tin đến người dùng khi người dùng tác động đến các phần tử điều khiển. Các phần tử điều khiển này nhận cửa sổ cha là một hộp thoại. Các phần tử điều khiển thường là các **Button**, **List Box**, **Combo Box**, **Check Box**, **Radio Button**, **Edit Box**, **Scroll Bar**, **Static**.

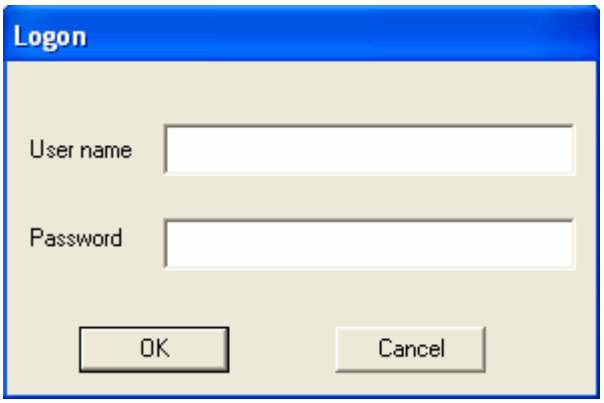
- Hộp thoại trạng thái (*modal*).
- Hộp thoại không trạng thái (*modeless*).
- Hộp thoại thông dụng (*common dialog*)

a) Thiết kế hộp thoại





Ví dụ:




```
IDD_DIALOG1 DIALOG DISCARDABLE  0, 0, 196, 102
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION
CAPTION "Logon"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON   "OK",IDOK,24,81,50,14
    PUSHBUTTON      "Cancel",IDCANCEL,109,81,50,14
    LTEXT           "User name",IDC_STATIC,7,23,40,15
    LTEXT           "Password",IDC_STATIC,7,50,40,16
    EDITTEXT DC_EDT_NAME,52,19,137,16,ES_AUTOHSCROLL
    EDITTEXT IDC_EDT_PASSWORD, 52, 48, 137, 16, ES_AUTOHSCROLL
END
```

Kiểu điều khiển	Lớp cửa sổ	Kiểu
PUSHBUTTON	Button	BS_PUSHBUTTON
DEFPUSHBUTTON	Button	BS_DEFBUSHBUTTON | WS_TABSTOP
CHECKBOX	Button	BS_CHECKBOX | WS_TABSTOP
RADIOBUTTON	Button	BS_RADIOBUTTON | WS_TABSTOP
GROUPBOX	Button	BS_GROUPBOX | WS_TABSTOP
LTEXT	Static	SS_LEFT | WS_GROUP
CTEXT	Static	SS_CENTER | WS_GROUP
RTEXT	Static	SS_RIGHT | WS_GROUP
ICON	Static	SS_ICON
EDITTEXT	Edit	ES_LEFT | WS_BORDER |

		WS_STABSTOP
SCROLLBAR	Scrollbar	SBS_HORZ
LISTBOX	Listbox	LBS_NOTIFY | WS_BORDER | WS_VSCROLL
COMBOBOX	Combobox	CBS_SIMPLE | WS_TABSTOP

Các kiểu điều khiển

*Các kiểu điều khiển được khai báo trong resource script có dạng như sau, ngoại trừ kiểu điều khiển **LISTBOX**, **COMBOBOX**, **SCROLLBAR**, **EDITTEXT**.*

***Control-type** "text", id, xPos, yPos, xWidth, yHeight, iStyle*

*Các kiểu điều khiển **LISTBOX**, **COMBOBOX**, **SCROLLBAR**, **EDITTEXT** được khai báo trong **resource script** với cấu trúc như trên nhưng không có trường **"text"**.*

*Thêm thuộc tính cho các kiểu điều khiển bằng cách thay đổi tham số iStyle. Ví dụ ta muốn tạo **radio button** với chuỗi diễn đạt nằm ở bên trái của nút thì ta gán trường iStyle bằng **BS_LEFTTEXT** cụ thể như sau.*

***RADIOBUTTON** Radio1",IDC_RADIO1,106,10,53,15,BS_LEFTTEXT*

b) Thủ tục xử lý hộp thoại

❖ Đặc điểm

- Mỗi hộp thoại cần có một thủ tục xử lý riêng.
- Các thông điệp không được gửi tới hàm xử lý cửa sổ chính.
- Là một hàm xử lý cửa sổ.

❖ Mẫu hàm

```
BOOL CALLBACK Tên hàm (HWND, UINT, WPARAM, LPARAM) ;
```

- Có nhiều thông điệp khác nhau.
- Không cần xử lý WM_PAINT và WM_DESTROY.
- Xử lý thông điệp nào thì trả về TRUE, nếu không trả về FALSE.

- Thường phải xử lý hai thông điệp chính: WM_INITDIALOG và WM_COMMAND: LOWORD(WPARAM) chứa ID các điều khiển.

Ví dụ:

```
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM,
LPARAM);
BOOL CALLBACK DialogProc (HWND, UINT, WPARAM,
LPARAM) ;
LRESULT CALLBACK WndProc (HWND hwnd, UINT message,
WPARAM wParam, LPARAM lParam)
{
        static HINSTANCE hInstance ;
        switch (message)
        {
                case WM_CREATE :
                hInstance = ((LPCREATESTRUCT) lParam)->hInstance ;
                return 0 ;
                case WM_COMMAND :
                        switch (LOWORD (wParam))
                        {
                        case IDC_SHOW :
                        DialogBox (hInstance, TEXT ("DIALOG1"),
                        hwnd, DialogProc) ;
                        break;
                        }
                        return 0 ;
                case WM_DESTROY :
                PostQuitMessage (0) ;
                return 0 ;
        }
        return DefWindowProc (hwnd, message, wParam, lParam) ;
}
/*----------------------hàm xử lý thông điệp hộp thoại----------------------*/
BOOL CALLBACK DialogProc (HWND hDlg, UINT message,
WPARAM wParam, LPARAM lParam)
{
        switch (message)
        {
                case WM_INITDIALOG:  return TRUE ;
                case WM_COMMAND:
                        switch (LOWORD (wParam))
                        {
                                case IDOK :
                                EndDialog (hDlg, 0) ;
```

```
41                                  return TRUE ;
42                              }
43                              break ;
44                      }
45                      return FALSE ;
46              }
```

c) Hộp thoại trạng thái

❖ Hiển thị hộp thoại

```
INT_PTR DialogBox(
  HINSTANCE hInstance,  // handle to module
  LPCTSTR lpTemplate,   // dialog box template
  HWND hWndParent,      // handle to owner window
  DLGPROC lpDialogFunc  // dialog box procedure
);
```

Ví dụ:

DialogBox (hInstance, TEXT ("DIALOG1"), hwnd, DialogProc) ;

❖ Gởi thông điệp đến hàm **WndProc** yêu cầu xử lý ngay cả khi hộp thoại đang mở nhờ hàm **SendMessage**:

SendMessage(**GetParent**(hDlg), message, wParam, lParam);

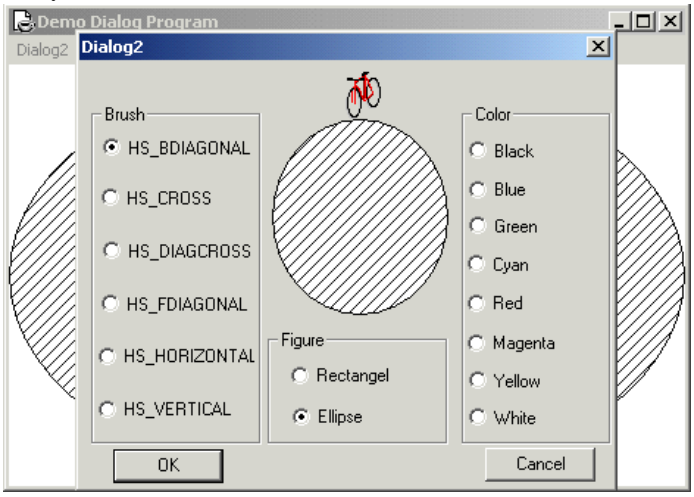
❖ Thêm tiêu đề cho hộp thoại:

SetWindowText(hDlg,**TEXT**("Hello Dialog")); trong xử lý thông điệp **WM_INITDIALOG**

❖ Đóng hộp thoại

```
BOOL EndDialog(
  HWND hDlg,        // handle to dialog box
  INT_PTR nResult   // value to return
);
```

❖ Ví dụ



```
1  LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM);
2  BOOL CALLBACK DialogProc (HWND, UINT, WPARAM, LPARAM);
3  int iCurrentColor = IDC_BLACK, iCurrentFigure = IDC_RECT;
4  int iCurrenBrush = IDC_HS_BDIAGONAL;
5  void PaintWindow(HWND hwnd, int iColor, int iFigure, int iBrush)
6  {
7          static COLORREF crColor[8] = { RGB(0, 0, 0), RGB(0, 0, 255),
8          RGB(0, 255, 0), RGB(0, 255, 255), RGB(255, 0, 0), RGB(255, 0, 255),
9          RGB(255, 255, 0), RGB(255, 255, 255) } ;
10         HBRUSH hBrush,hbrush;
11         HDC hdc ;
12         RECT rect ;
13         hdc = GetDC (hwnd) ;
14         GetClientRect (hwnd, &rect) ;
15         if(iBrush==IDC_HS_BDIAGONAL)
16                 hbrush=CreateHatchBrush(HS_BDIAGONAL,
17                 crColor[iColor-IDC_BLACK]);
18         if(iBrush == IDC_HS_CROSS)
19                 hbrush=CreateHatchBrush(HS_CROSS,
20                 crColor[iColor - IDC_BLACK]);
21         if(iBrush == IDC_HS_DIAGCROSS)
22                 hbrush=CreateHatchBrush(HS_DIAGCROSS,
23                 crColor[iColor - IDC_BLACK]);
24         if(iBrush == IDC_HS_FDIAGONAL)
25                 hbrush=CreateHatchBrush(HS_FDIAGONAL,
26                 crColor[iColor - IDC_BLACK]);
27         if(iBrush == IDC_HS_HORIZONTAL)
28                 hbrush=CreateHatchBrush(HS_HORIZONTAL,
29                 crColor[iColor - IDC_BLACK]);
30         if(iBrush == IDC_HS_VERTICAL)
31                 hbrush=CreateHatchBrush(HS_BDIAGONAL,
32                 crColor[iColor - IDC_BLACK]);
33                 hBrush = (HBRUSH) SelectObject (hdc, hbrush) ;
34         if (iFigure == IDC_RECT)
35                 Rectangle (hdc, rect.left, rect.top, rect.right, rect.bottom) ;
36         else
37                 Ellipse(hdc, rect.left, rect.top, rect.right, rect.bottom) ;
38         DeleteObject (SelectObject (hdc, hBrush)) ;
39         ReleaseDC (hwnd, hdc) ;
40 }
41 void PaintTheBlock(HWND hCtrl, int iColor, int iFigure, int iBrush)
42 {
43         InvalidateRect (hCtrl, NULL, TRUE) ;
44         UpdateWindow (hCtrl) ;
45         PaintWindow (hCtrl, iColor, iFigure,iBrush) ;
46 }
```

```
47  LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM
48  wParam, LPARAM lParam)
49  {
50          static HINSTANCE hInstance ;
51          PAINTSTRUCT ps ;
52          switch (message)
53          {
54          case WM_CREATE:
55                  hInstance = ((LPCREATESTRUCT) lParam)->hInstance ;
56                  return 0 ;
57                  case WM_COMMAND:
58                          switch (LOWORD (wParam))
59                          {
60                                  case IDC_SHOW:
61                                  if (DialogBox (hInstance, TEXT ("DIALOG"),
62                                  hwnd, DialogProc))
63                                          InvalidateRect (hwnd, NULL, TRUE) ;
64                                  return 0 ;
65                          }
66                          break;
67                  case WM_PAINT:
68                          BeginPaint (hwnd, &ps) ;
69                          EndPaint (hwnd, &ps) ;
70                          PaintWindow (hwnd, iCurrentColor, iCurrentFigure,
71                          iCurrenBrush) ;
72                          return 0 ;
73                  case WM_DESTROY:
74                          PostQuitMessage (0) ;
75                          return 0 ;
76          }
77          return DefWindowProc (hwnd, message, wParam, lParam) ;
78  }
79  BOOL CALLBACK DialogProc (HWND hDlg, UINT message, WPARAM
80  wParam, LPARAM lParam)
81  {
82          static HWND hCtrlBlock ;
83          static int iColor, iFigure,iBrush;
84          switch (message)
85          {
86                  case WM_INITDIALOG:
87                          iColor = iCurrentColor ;
88                          iFigure = iCurrentFigure ;
89                          iBrush = iCurrenBrush;
90                          CheckRadioButton(hDlg,IDC_BLACK,IDC_WHITE,
91                          iColor);
```

```
92                  CheckRadioButton(hDlg,IDC_RECT,IDC_ELLIPSE,iFig
93                  ure);CheckRadioButton (hDlg, IDC_HS_BDIAGONAL,
94                  IDC_HS_VERTICAL, iBrush);
95                  hCtrlBlock = GetDlgItem (hDlg, IDC_PAINT) ;
96                  SetFocus (GetDlgItem (hDlg, iColor)) ;
97                  return FALSE ;
98          case WM_COMMAND:
99          switch (LOWORD (wParam))
100         {
101                 case IDOK:
102                         iCurrentColor = iColor ;
103                         iCurrentFigure = iFigure ;
104                         iCurrenBrush = iBrush;
105                         EndDialog (hDlg, TRUE) ;
106                         return TRUE ;
107                 case IDCANCEL:
108                         EndDialog (hDlg, FALSE) ;
109                         return TRUE ;
110                 case IDC_BLACK:
111                 case IDC_RED:
112                 case IDC_GREEN:
113                 case IDC_YELLOW:
114                 case IDC_BLUE:
115                 case IDC_MAGENTA:
116                 case IDC_CYAN:
117                 case IDC_WHITE:
118                         iColor = LOWORD (wParam) ;
119                         CheckRadioButton (hDlg, IDC_BLACK,
120                         IDC_WHITE, LOWORD (wParam)) ;
121                         PaintTheBlock (hCtrlBlock, iColor,
122                         iFigure,iBrush);
123                         return TRUE ;
124                 case IDC_RECT:
125                 case IDC_ELLIPSE:
126                         iFigure = LOWORD (wParam) ;
127                         CheckRadioButton (hDlg, IDC_RECT,
128                         IDC_ELLIPSE, LOWORD (wParam)) ;
129                         PaintTheBlock (hCtrlBlock, iColor,
130                         iFigure,iBrush);
131                         return TRUE ;
132                 case IDC_HS_BDIAGONAL:
133                 case IDC_HS_CROSS:
134                 case IDC_HS_DIAGCROSS:
135                 case IDC_HS_FDIAGONAL:
136                 case IDC_HS_HORIZONTAL:
137                 case IDC_HS_VERTICAL:
```

```
                    iBrush = LOWORD (wParam)
                    CheckRadioButton(hDlg,IDC_HS_BDIAGONAL,
                    IDC_HS_VERTICAL, LOWORD (wParam)) ;
                    PaintTheBlock (hCtrlBlock, iColor,
                    iFigure,iBrush);
                    return TRUE ;
                }
                break;
        case WM_PAINT:
                PaintTheBlock (hCtrlBlock, iColor, iFigure,iBrush) ;
                break ;
        }
        return FALSE ;
}
```

d) Hộp thoại không trạng thái

❖ Hiển thị hộp thoại

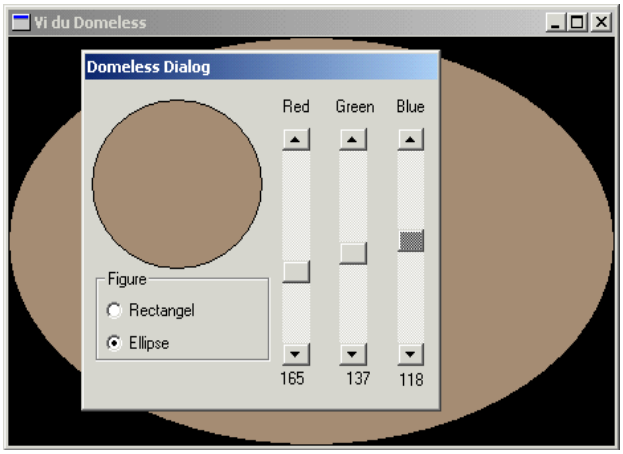
```
HWND hDlgModeless=CreateDialog(hInstance, szTemplate,
hwndParent, DialogProc);
ShowWindow(hDlgModeless,SW_SHOW);

while(GetMessage(&msg, NULL, 0, 0))
{
        if (hDlgModeless==0 || !IsDialogMessage
        (hDlgModeless, &msg);
        {
                TranslateMessage(&msg);
                DispatchMessage(&msg);
        }
}

while(GetMessage(&msg, NULL, 0, 0))
{
        if (hDlgModeless==0 || !IsDialogMessage(hDlgModeless,
        &msg);
        {
                if(TranslateAccelerator (hwnd, hAccel, &msg)
                {
                        TranslateMessage(&msg);
                        DispatchMessage(&msg);
                }
        }
}
```

❖ Đóng hộp thoại

Đặt hDlgModeless về giá trị 0.

```
BOOL DestroyWindow(
  HWND hWnd   // handle to window to destroy
);
```

❖ Ví dụ




```
1   void PaintWindow (HWND hwnd, int iColor[], int iFigure)
2   {
3       HBRUSH hBrush ;
4       HDC hdc ;
5       RECT rect ;
6       hdc = GetDC(hwnd) ;
7       GetClientRect (hwnd, &rect) ;
8       hBrush = CreateSolidBrush(RGB(iColor[0], iColor[1],
9       iColor[2]));
10      hBrush = (HBRUSH) SelectObject (hdc, hBrush) ;
11      if (iFigure == IDC_RECT)
12          Rectangle (hdc, rect.left, rect.top, rect.right,
13          rect.bottom) ;
14      else
15          Ellipse(hdc, rect.left, rect.top, rect.right,
16          rect.bottom) ;
17      DeleteObject (SelectObject (hdc, hBrush)) ;
18      ReleaseDC (hwnd, hdc) ;
19  }
20  LRESULT CALLBACK WndProc (HWND hwnd, UINT
21  message, WPARAM wParam, LPARAM lParam)
22  {
23      switch (message)
24      {
25          case WM_PAINT:
26              PaintTheBlock(hwnd, iColor, iFigure) ;
27              return 0 ;
28          case WM_DESTROY :
```

```
29                              DeleteObject((HGDIOBJ)SetClassLong(hw
30                              nd, GCL_HBRBACKGROUND,(LONG)
31                              GetStockObject (WHITE_BRUSH))) ;
32                              PostQuitMessage (0) ;
33                              return 0 ;
34                      }
35                      return DefWindowProc (hwnd, message, wParam,
36                      lParam);
37              }
38              void PaintTheBlock (HWND hCtrl, int iColor[], int iFigure)
39              {
40                      InvalidateRect (hCtrl, NULL, TRUE);
41                      UpdateWindow (hCtrl) ;
42                      PaintWindow (hCtrl, iColor, iFigure) ;
43              }
44              BOOL CALLBACK ColorScrDlg (HWND hDlg, UINT
45              message, WPARAM wParam, LPARAM lParam)
46              {
47                      HWND hwndParent, hCtrl ;
48                      static HWND hCtrlBlock ;
49                      int iCtrlID, iIndex ;
50                      switch (message)
51                      {
52                              case WM_INITDIALOG :
53                                      hCtrlBlock = GetDlgItem (hDlg,
54                                      IDC_PAINT) ;
55                                      for (iCtrlID = 10 ; iCtrlID < 13 ; iCtrlID++)
56                                      {
57                                              hCtrl = GetDlgItem (hDlg, iCtrlID) ;
58                                              PaintTheBlock (hCtrlBlock, iColor,
59                                              iFigure) ;
60                                              PaintTheBlock (hwndParent, iColor,
61                                              iFigure) ;
62                                              SetScrollRange (hCtrl, SB_CTL, 0,
63                                              255, FALSE) ;
64                                              SetScrollPos(hCtrl, SB_CTL, 0,
65                                              FALSE) ;
66                                      }
67                                      return TRUE ;
68                              case WM_COMMAND:
69                                      {
70                                      switch( LOWORD(wParam))
71                                      {
72                                              case IDC_RECT:
73                                              case IDC_ELLIPSE:
74                                                      iFigure = LOWORD(wParam);
```

```
                                        hwndParent =
                                        GetParent(hDlg);
                                        CheckRadioButton(hDlg,
                                        IDC_RECT, IDC_ELLIPSE,
                                        LOWORD (wParam)) ;
                                        PaintTheBlock(hCtrlBlock,
                                        iColor, iFigure) ;
                                        PaintTheBlock (hwndParent,
                                        iColor, iFigure) ;
                                        return TRUE ;
                        }
                        break;
                }
                case WM_VSCROLL :
                        hCtrl = (HWND) lParam ;
                        iCtrlID = GetWindowLong (hCtrl,
                        GWL_ID) ;
                        iIndex = iCtrlID - 10 ;
                        hwndParent = GetParent (hDlg) ;
                        PaintTheBlock (hCtrlBlock, iColor, iFigure);
                        PaintTheBlock (hwndParent, iColor,
                        iFigure) ;
                        switch (LOWORD (wParam))
                        {
                                case SB_PAGEDOWN :
                                        iColor[iIndex] += 15 ;
                                case SB_LINEDOWN :
                                        iColor[iIndex] = min (255,
                                        iColor[iIndex] + 1) ;
                                        break;
                                case SB_PAGEUP :
                                        iColor[iIndex] -= 15 ;
                                        case SB_LINEUP :
                                        iColor[iIndex] = max (0,
                                        iColor[iIndex] - 1);
                                        break;
                                case SB_TOP :
                                        iColor[iIndex] = 0 ;
                                        break;
                                case SB_BOTTOM :
                                        iColor[iIndex] = 255 ;
                                        break;
                                case SB_THUMBPOSITION :
                                case SB_THUMBTRACK :
                                        iColor[iIndex] = HIWORD
                                        (wParam) ;
```

```
121                                         break;
122                                 default :
123                                         return FALSE ;
124                                 }
125                         SetScrollPos(hCtrl, SB_CTL, iColor[iIndex],
126                         TRUE) ;
127                         SetDlgItemInt (hDlg, iCtrlID + 3, iColor[iIndex],
128                         FALSE) ;
129                         InvalidateRect(hwndParent,NULL,TRUE);
130                         DeleteObject ( (HGDIOBJ)SetClassLong(
131                         hwndParent, GCL_HBRBACKGROUND, (LONG)
132                         CreateSolidBrush( RGB(iColor[0], iColor[1],
133                         iColor[2]) ) ) ) ;
134                         return TRUE ;
135                         case WM_PAINT:
136                                 PaintTheBlock(hCtrlBlock, iColor, iFigure) ;
137                                 break;
138                 }
139                 return FALSE ;
```

2. Menu

a) Tạo Menu

```
MENUDEMO MENU DISCARDABLE
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "&New", IDM_FILE_NEW
        MENUITEM "&Open", IDM_FILE_OPEN
        MENUITEM "&Save", IDM_FILE_SAVE
        MENUITEM "Save &As...", IDM_FILE_SAVE_AS
        MENUITEM SEPARATOR
        MENUITEM "E&xit", IDM_APP_EXIT
    END
    POPUP "&Edit"
    BEGIN
        MENUITEM "&Undo", IDM_EDIT_UNDO
        MENUITEM SEPARATOR
        MENUITEM "C&ut", IDM_EDIT_CUT
        MENUITEM "&Copy", IDM_EDIT_COPY
        MENUITEM "&Paste", IDM_EDIT_PASTE
        MENUITEM "De&lete", IDM_EDIT_CLEAR
    END
    POPUP "&Background"
    BEGIN
```

```
            MENUITEM "&White", IDM_BKGND_WHITE,
            CHECKED
            MENUITEM "&Light Gray", IDM_BKGND_LTGRAY
            MENUITEM "&Gray", IDM_BKGND_GRAY
            MENUITEM "&Dark Gray", IDM_BKGND_DKGRAY
            MENUITEM "&Black", IDM_BKGND_BLACK
        END
        POPUP "&Help"
        BEGIN
            MENUITEM "&Help...", IDM_APP_HELP
            MENUITEM "&About ...", IDM_APP_ABOUT
        END
    END
```

b) Thiết lập Menu

wndclass.lpszMenuName = "MENU1";

hoặc:

hMenu = **LoadMenu** (hInstance, **TEXT**("MENU1"));

hwnd = **CreateWindow** (**TEXT**("MyClass"), **TEXT**("Window Caption"), WS_OVERLAPPEDWINDOW, CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, NULL, hMenu, hInstance, NULL);

SetMenu(hWnd, hMenu);

LOWORD(WPARAM) chứa ID các điều khiển.

c) Ví dụ

```
1   LRESULT CALLBACK WndProc (HWND, UINT, WPARAM,
2   LPARAM);
3   /* Khai báo tên dùng chung cho cáctài nguyên trong chương trình.*/
4   TCHAR szAppName[] = TEXT ("MenuDemo") ;
5   int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE
6   hPrevInstance, PSTR szCmdLine, int iCmdShow)
7   {
8       HWND hwnd;
9       MSG msg;
10      WNDCLASS wndclass;
11      wndclass.style = CS_HREDRAW | CS_VREDRAW;
12      wndclass.lpfnWndProc = WndProc ;
13      wndclass.cbClsExtra = 0 ;
14      wndclass.cbWndExtra = 0 ;
15      wndclass.hInstance = hInstance ;
16      wndclass.hIcon = LoadIcon(NULL, IDI_APPLICATION);
```

```
17              wndclass.hCursor = LoadCursor(NULL, IDC_ARROW);
18              wndclass.hbrBackground =
19              (HBRUSH)GetStockObject(WHITE_BRUSH) ;
20              wndclass.lpszMenuName = szAppName ;
21              wndclass.lpszClassName = szAppName ;
22              if (!RegisterClass (&wndclass))
23              {
24                      MessageBox(NULL, TEXT("This program requires
25                      Windows "), szAppName, MB_ICONERROR) ;
26                      return 0 ;
27              }
28              hwnd = CreateWindow (szAppName, TEXT("Menu
29              Demonstration"), WS_OVERLAPPEDWINDOW,
30              CW_USEDEFAULT, CW_USEDEFAULT,
31              CW_USEDEFAULT, CW_USEDEFAULT, NULL, NULL,
32              hInstance, NULL) ;
33              ShowWindow (hwnd, iCmdShow) ;
34              UpdateWindow (hwnd) ;
35              while (GetMessage(&msg, NULL, 0, 0))
36              {
37                      TranslateMessage (&msg) ;
38                      DispatchMessage (&msg) ;
39              }
40              return msg.wParam ;
41      }
42      LRESULT CALLBACK WndProc (HWND hwnd, UINT message,
43      WPARAM wParam, LPARAM lParam)
44      {
45          /* Khao báo danh sách các màu chổi tô, các hằng này được định
46          nghĩa trong file WINGDI.H */
47          static int idColor[5] = { WHITE_BRUSH, LTGRAY_BRUSH,
48          GRAY_BRUSH, DKGRAY_BRUSH, BLACK_BRUSH } ;
49          static int iSelection = IDM_BKGND_WHITE ;
50          HMENU hMenu ;
51          switch (message)
52          {
53                  case WM_COMMAND:
54                  hMenu = GetMenu (hwnd) ; // Lấy định danh của menu
55                  switch (LOWORD (wParam)) //Kiểm tra định danh mục chọn
56                  {
57                          case IDM_FILE_NEW:
58                          case IDM_FILE_OPEN:
59                          case IDM_FILE_SAVE:
60                          case IDM_FILE_SAVE_AS:
61                                  MessageBeep(0) ; //Phát ra tiếng kêu bíp
62                                  return 0 ;
```

```
63                      case IDM_APP_EXIT:
64                      /*Gởi thông điệp để đóng ứng dụng lại*/
65                              SendMessage (hwnd, WM_CLOSE, 0, 0) ;
66                              return 0 ;
67                      case IDM_EDIT_UNDO:
68                      case IDM_EDIT_CUT:
69                      case IDM_EDIT_COPY:
70                      case IDM_EDIT_PASTE:
71                      case IDM_EDIT_CLEAR:
72                              MessageBeep (0) ;
73                              return 0 ;
74                      case IDM_BKGND_WHITE:
75                      case IDM_BKGND_LTGRAY:
76                      case IDM_BKGND_GRAY:
77                      case IDM_BKGND_DKGRAY:
78                      case IDM_BKGND_BLACK:
79                              /* Bỏ check của mục chọn trước đó*/
80                              CheckMenuItem(hMenu,iSelection,
81                              MF_UNCHECKED);
82                              iSelection = LOWORD (wParam) ; /*Lấy ID
83                              mục mới*/
84                              /* Check mục chọn mới*/
85                              CheckMenuItem (hMenu, iSelection,
86                              MF_CHECKED) ;
87                              /* Thiết lập màu tương ứng với mục chọn
88                              mới*/
89                              SetClassLong(hwnd,GCL_HBRBACKGRO
90                              UND, (LONG)
91                              GetStockObject(idColor[iSelection-
92                              IDM_BKGND_WHITE]));
93                              InvalidateRect (hwnd, NULL, TRUE) ;
94                              return 0 ;
95                      case IDM_APP_HELP:
96                              MessageBox(hwnd, TEXT("Help not yet
97                              implemented!"), szAppName,
98                              MB_ICONEXCLAMATION | MB_OK) ;
99                              return 0 ;
100                     case IDM_APP_ABOUT:
101                             MessageBox (hwnd, TEXT ("Menu
102                             Demonstration Program\n (c) Charles
103                             Petzold, 1998"), szAppName,
104                             MB_ICONINFORMATION | MB_OK) ;
105                             return 0 ;
106             }
107                     break;
108             case WM_DESTROY:
```

```
109                         PostQuitMessage(0) ;
110                         return 0 ;
111                 }
112                 return DefWindowProc(hwnd, message, wParam, lParam) ;
113         }
```

Bài 5: XỬ LÝ VĂN BẢN

Phân bố thời lượng:
- *Số tiết giảng ở lớp: 6 tiết*
- *Số tiết tự học ở nhà: 6 tiết*
- *Số tiết cài đặt chương trình ở nhà: 12 tiết*

1. Hiển thị văn bản

Để hiện thị nội dung văn bản trên các thiết bị xuất, dựa vào từng trường hợp thể hiện khác nhau, ta dùng các hàm Win32 API khác nhau. Các hàm này phụ thuộc vào font chữ, thuộc tính của thiết bị ngữ cảnh **DC** (*Device Context*) và khoảng cách ký tự thể hiện.

Hàm phổ biến nhất thực hiện thao tác xuất một chuỗi ký tự văn bản, sử dụng font chữ, màu chữ và màu nền hiện hành là :

- *BOOL TextOut(HDC hdc, int nXStart, int nYStart, LPCTSTR lpString,int cbString);*

→ *trả về giá trị khác không nếu thành công, ngược lại trả về 0.*

- *LONG TabbedTextOut(HDC hDC, int nX, int nY, LPCTSTR lpString, int nCount, int nNumTabs, LPINT lpnTabStopPositions, int nTabOrigin);*

  Nếu trong chuỗi ký tự có các ký tự tab (‘\t’ hoặc 0x09), hàm TabbedTextOut sẽ chuyển các ký tự tab vào dãy các vị trí "dừng" tương ứng. Số lượng các tab dừng được xác định bởi nNumTabs, và lpnTabStopPositions là dãy vị trí các tab dừng theo đơn vị tính pixels. Ví dụ, nếu độ rộng trung bình của mỗi ký tự là 8 pixels, và mỗi tab dừng cần đặt cách nhau 5 ký tự, dãy các tab dừng sẽ phải lần lượt có giá trị 40, 80, 120, ... Tuy nhiên, các giá trị này không nhất thiết phải là bội số của nhau.

  Nếu biến nNumTabs hoặc lpnTabStopPositions có giá trị là 0 và NULL, các tab dừng được đặt cách nhau từng 8 ký tự. Nếu nNumTabs bằng 1, lpnTabStopPositions trỏ đến giá trị xác định một dãy tăng tuần hoàn là bội số của dãy này. Ví dụ, nếu nNumTabs bằng 1, và

lpnTabStopPositions bằng 30, ta sẽ có dãy tab dừng tại vị trí 30, 60, 90, ... pixels.

Trường nTabOrigin xác định tọa độ theo trục x của điểm bắt đầu tính khoảng cách tới các tab. Giá trị này không nhất thiết phải là vị trí đầu tiên của chuỗi, có thể chọn trùng hoặc không.

Hàm trả về kích thước chuỗi hiển thị, theo đơn vị logic, nếu thành công. Ngược lại, hàm trả về 0. Trong đó, chiều cao chuỗi là WORD cao của biến kiểu LONG, chiều rộng là WORD thấp.

- *int DrawText(HDC hDC, LPCTSTR lpString, int nCount, LPRECT lpRect, UINT uFormat);*

  Cũng như các hàm xuất văn bản khác, hàm DrawText xuất chuỗi xác định bởi con trỏ lpString có độ dài nCount. Tuy nhiên, với chuỗi có ký tự kết thúc là NULL, nếu nCount bằng -1, hàm sẽ tự động tính toán chiều dài của chuỗi.

  Biến lpRect trỏ đến cấu trúc RECT của hình chữ nhật (theo toạ độ logic) mà trong đó văn bản thể hiện theo định dạng được thiết lập trong uFormat.

  Nếu uFormat bằng 0, nội dung văn bản sẽ được hiển thị theo từng dòng từ trên xuống dưới. Mỗi dòng mới được xác định thông qua ký tự về đầu dòng CR (carriage return, bằng '\r' hoặc 0x0D) hoặc ký tự xuống dòng LF (linefeed, bằng '\n' hoặc 0x0A) có trong văn bản. Phần văn bản bên ngoài hình chữ nhật lpRect sẽ bị cắt bỏ.

  Giá trị uFormat bằng 0 cũng chính là giá trị cờ canh lề trái (DT_LEFT). Ngoài ra, ta có thể thiết lập các cờ canh lề phải (DT_RIGHT), và canh lề giữa (DT_CENTER) cho văn bản.

  Để loại bỏ chức năng điều khiển của các ký tự CR và LF, cần thêm vào cờ DT_SINGLELINE. Nếu thiết lập DT_SINGLELINE, ta cũng có thể chỉ định vị trí của dòng hiển thị ở phía trên (DT_TOP), phía dưới (DT_BOTTOM), hoặc ở chính giữa (DT_VCENTER) trong vùng hình chữ nhật.

Trong trường hợp hiển thị nhiều dòng văn bản, Windows chỉ ngắt dòng khi gặp ký tự CR và LF. Để ngắt dòng dài hơn kích thước hình chữ nhật hiển thị, cần thiết lập cờ DT_WORDBREAK. Nếu không muốn Windows cắt bỏ các phần dư ra khi vẽ chữ vượt quá phạm vi khung chữ nhật, ta thêm cờ DT_NOCLIP. Nếu muốn ký tự tab ('\t' hoặc 0x09) được diễn dịch thành ký tự phân cột, cần thêm cờ DT_EXPANDTABS. Giá trị mặc định của tab là 8 khoảng trắng. Cờ DT_TABSTOP được dùng để đặt lại giá trị tab. Trong trường hợp này, byte cao của word thấp (bits 15-8) của uFormat sẽ chứa giá trị tab cần thay thế.

2. Định dạng văn bản

a) Hàm thiết lập màu chữ và màu nền:

- *COLORREF SetTextColor (HDC hdc, COLORREF color);*
- *COLORREF SetBkColor (HDC hdc, COLORREF color);*
  - Trả về giá trị màu trước đó.
  - Nếu có lỗi trả về CLR_INVALID.
- *int SetBkMode (HDC hdc, int mode) ;*
  - Trả về chế độ nền trước đó.
  - Trả về 0 nếu gặp lỗi.

  mode = OPAQUE : Mỗi khi hiển thị văn bản thì màu nền được thay đổi thành màu nền hiện hành. Hoặc TRANSPARENT: Màu nền không bị ảnh hưởng ➔ SetBkColor() bị vô hiệu.

b) Xác định màu chữ và màu nền hiện hành:

- *COLORREF GetTextColor(HDC hDC);*
- *COLORREF GetBkColor(HDC hDC);*

c) Xác định chế độ nền hiện tại:

- *int GetBkMode(HDC hDC);*

  Hàm trả về giá trị TRANSPARENT hoặc OPAQUE, nếu thành công. Ngược lại, giá trị trả về là zero.

d) Để xác lập vị trí chuỗi văn bản hiển thị dựa trên điểm gốc nXStart, nYStart:

❖ *UINT SetTextAlign(HDC hDC, UINT fMode);*

fMode: TA_LEFT, TA_RIGHT, TA_CENTER, TA_TOP, TA_BOTTOM, TA_BASELINE, TA_UPDATE

e) Để biết chế độ canh lề văn bản hiện tại, ta dùng hàm :

❖ *UINT GetTextAlign(HDC hDC);*

Nếu thành công, hàm trả về cờ tương ứng của canh lề văn bản hiện hành. Ngược lại, giá trị trả về là GDI_ERROR.

f) Để thay đổi khoảng cách giữa các ký tự:

❖ *int SetTextCharacterExtra(HDC hDC, int nCharExtra);*

Nếu thành công, hàm trả về khoảng cách trước khi được thiết lập. Ngược lại, giá trị trả về là 0x80000000.

g) Để biết khoảng cách hiện tại, ta dùng hàm :

❖ *int GetTextCharacterExtra(HDC hDC);*

Nếu thành công, giá trị trả về cho biết khoảng cách hiện tại. Ngược lại, giá trị trả về là 0x80000000.

3. Sử dụng font

- Lập chỉ số font chữ.
- Nạp font chữ.
- Gán chỉ số font chữ cho ngữ cảnh thiết bị.

Đối với Font chữ mặc định (hệ thống):Sử dụng các font chữ Windows đang sử dụng.

MACRO	FONT
ANSI_FIXED_FONT	Font với kích thước cố định của ký tự dựa trên Windows. Font Courier là một ví dụ điển hình của dạng font này.
ANSI_VAR_FONT	Font với độ rộng ký tự thay đổi dựa trên các ký tự chuẩn của Windows. Font MS San Serif là một ví dụ điển hình.
DEVICE_DEFAULT_FONT	Font với thiết bị đã cho được chọn mặc nhiên. Dạng font này thường co sẵn trong hệ thống để điều khiển việc trình bày trên thiết

	bị. Tuy nhiên, đối với một số thiết bị, font được cài đặt ngay trên thiết bị. Ví dụ, đối với máy in, các font thiết bị cài sẵn thực hiện thao tác in nhanh hơn so với việc load bitmap ảnh về từ máy tính.
DEFAULT_GUI_FONT	Font của giao diện đồ họa được thiết lập mặc định.
OEM_FIXED_FONT	Font chữ cố định, dựa trên bộ ký tự OEM. Ví dụ, đối với máy IBM®, font OEM dựa trên bộ ký tự IBM PC.
SYSTEM_FONT	Font hệ thống của Windows. Được hệ điều hành dùng để trình bày các thành phần giao diện như thanh tiêu đề, menu, nội dung văn bản trong các hộp thoại thông điệp. Các font hệ thống này luôn có sẵn khi cài hệ điều hành, trong khi các font khác cần phải cài thêm tùy theo ứng dụng sau này.
SYSTEM_FIXED_FONT	Font Windows được sử dụng như font hệ thống trong các phiên bản trước 3.0.

Macro các font định nghĩa sẵn.

- Nạp: HGDIOBJ GetStockObject(int fnObject) → Nếu thành công, trả về handle font chữ. Ngược lại, giá trị trả về là NULL.

  Trong đó, kiểu HGDIOBJ là HFONT, biến fnObject là một trong các macro ở bảng trên.
- Gán chỉ số cho DC: HGDIOBJ SelectObject(HDC hDC, HGDIOBJ hGDIObj) → Trả về handle font chữ vừa sử dụng trước, lỗi trả về GDI_ERROR

Hoặc gọn hơn, ta có thể gọi :

SelectObject(hDC.**GetStockObject**(fnObject));

DeleteObject (Đối tượng): để hủy.

Ví dụ:

```
HFONT hfnt, hOldFont;
hfnt = GetStockObject(ANSI_VAR_FONT);
if (hOldFont = SelectObject(hdc, hfnt))
{
```

```
        TextOut(hdc, 10, 50, "Sample ANSI_VAR_FONT text.", 26);
        SelectObject(hdc, hOldFont);
}
```

❖ Xác định kích thước font

BOOL GetTextMetrics(HDC hdc, LPTEXTMETRIC lptm);

```
typedef struct tagTEXTMETRIC // tm
{
        LONG tmHeight;
        LONG tmAscent;
        LONG tmDescent;
        LONG tmInternalLeading;
        LONG tmExternalLeading;
        LONG tmAveCharWidth;
        LONG tmMaxCharWidth;
        LONG tmWeight;
        LONG tmOverhang;
        LONG tmDigitizedAspectX;
        LONG tmDigitizedAspectY;
        BCHAR tmFirstChar;
        BCHAR tmLastChar;
        BCHAR tmDefaultChar;
        BCHAR tmBreakChar;
        BYTE tmItalic;
        BYTE tmUnderlined;
        BYTE tmStruckOut;
        BYTE tmPitchAndFamily;
        BYTE tmCharSet;
} TEXTMETRIC;
```

Cấu trúc TEXTMETRIC gồm 20 thành phần, một số thành phần quan trọng gồm:

- tmHeight: Chiều cao ký tự tính bằng pixel.
- tmInternalLeading: Vùng chứa dấu trọng âm.
- tmExternalLeading: Không gian giữa 2 dòng.
- tmAveCharWidth: Bề rộng trung bình mỗi ký tự.
- tmPitchAndFamily: Họ c*ủa font (8 bit).*

Ví dụ:

```
static int cxchar, cychar;
```

```
TEXTMETRIC tm;
case WM_CREATE:
    {
            hdc = GetDC(hwnd);
            GetTextMetrics(hdc, &tm);
            cxchar=tm.tmInternalLeading+tm.tmExternal;
            cychar=tm.tmAveCharWidth;
            ReleaseDC(hwnd, hdc);
            return 0;
    }
case WM_PAINT:
{
        for(int i=0; i<10; i++)
            TextOut(hdc, cxchar, cychar*i, "aaa", 3);
    }
```

❖ Tính độ dài của xâu ký tự

- Các ký tự hiển thị có bề rộng khác nhau do vậy không nên dùng hàm strlen() để lấy số ký tự → độ dài.
- Dùng hàm: BOOL GetTextExtentPoint32 (HDC hdc, LPCSTR lpszString, int len, LPSIZE lpSize);

```
typedef struct tagSIZE
{
        long cx;
        long cy;        //Tính theo đơn vị logic
} SIZE;
```

len: Tổng số ký tự.

Tạo lập đặc tính mới cho font chữ

HFONT CreateFont (int Height, int Width, int Escapement, int Orientation, int fnWeight, DWORD Italic, DWORD Underline, DWORD StrikeOut, DWORD CharSet, DWORD outputPrecision, DWORD ClipPrecision, DWORD Quality, DWORD PitchAndFamily, LPCSTR lpszFontName)

Với:

- PitchAndFamily: DEFAULT_PITCH | FF_DONTCARE
- charSet: ANSI_CHARSET
- outputPrecision: OUT_DEFAULT_PRECIS
- clipPrecision: CLIP_DEFAULT_PRECIS
- Quality: DEFAULT_QUALITY
- fnWeight: 0 → 1000 (thông thường là 400)

Tên	Giá trị
FW_DONTCARE	0
FW_THIN	100
FW_EXTRALIGHT	200
FW_ULTRALIGHT	200
FW_LIGHT	300
FW_NORMAL	400
FW_REGULAR	400
FW_MEDIUM	500

Tên	Giá trị
FW_SEMIBOLD	600
FW_DEMIBOLD	600
FW_BOLD	700
FW_EXTRABOLD	800
FW_ULTRABOLD	800
FW_HEAVY	900
FW_BLACK	900

Macro xác định độ đậm nhạt lfWeight

Tài liệu tham khảo

[1] **ĐẶNG VĂN ĐỨC**: "Lập trình C trên Windows". Nhà Xuất Bản Khoa Học Kỹ Thuật – 1998.

[2] **NGUYỄN ĐÌNH QUYỀN – MAI XUÂN HÙNG**: "Giáo trình lập trình C trên Windows". Nhà Xuất Bản Đại Học Quốc Gia Tp. Hồ Chí Minh – 2003.

[3] **MSDN** – 10/2001