Windows Programming

Prepared for: All Students to study a big syllabus in simplistic way.

Windows Fundamentals

- Windows applications can be developed using a procedure-oriented approach in either C or C++.
- All approaches bring together point-and-shoot control, pop-up menus, and the ability to run applications written especially for the Windows environment.
- Windows gives the ability to develop graphics user interface(GUI)

The Windows Environment

- Windows is a graphics-based multitasking operating system.
- Programs developed for this environment have a consistent look and command structure.
- To the user, this makes learning each successive Windows application easier.
- To help in the development of Windows applications, Windows provides numerous **built-in** functions that allow for easy implementation of pop-up menus, scroll bars, dialog boxes, icons that represent a user-friendly interface.
- Windows permits the application to work in a hardware-independent manner.

Windows Advantages

- **Graphics User Interface (GUI)**
  - All versions of Windows are based on the same standardized interface.
  - This interface uses pictures, or icons, to represent disk drives, files, subdirectories, and many of the operating system commands and actions.

- **Multitasking Environment**
  - The Windows multitasking environment allows the user to have several applications, or several instances of the same application, running at the same time.
  - Each application occupies a rectangular window on the screen.

Windows Advantages(contd.)

- **Advantages of Using a Queued Input**

- Windows receives all input from the keyboard, mouse, and timer in the system queue.
- It is the queue's responsibility to redirect the input to the appropriate program since more than one application can be running.
- It is achieved by copying the message from the system queue into the application's queue.
- When application is ready to process the input, it reads from its queue and dispatches a message to the correct window.
- Input is accessed with the use of a uniform format called an ***input message*.**

Windows Advantages(contd.)

- **OOPs and Windows Messages**

- Windows has always employed a pseudo-OOP environment.
- Message is a notification that some event of interest has occurred that may or may not need a specific action.
- User may initiate these events by clicking or moving the mouse, changing the size of a window, or making a menu selection.
- The events can also be initiated by the application itself e.g. a graphics-based spreadsheet could finish a recalculation that results in the need to update a graphics bar chart.
- It is the message system that allows Windows to achieve its multitasking capabilities.
- The message system makes it possible for Windows to share the processor among different applications.

Windows Advantages(contd.)

- **Managing Memory**

- Most important shared resources under Windows is system memory.
- As new programs are started and old ones are terminated, memory become fragmented.
- Windows is capable of consolidating free memory space by moving blocks of code and data in memory.

Windows Advantages(contd.)

- **Hardware-independence**

- Windows frees developer from having to build programs that take into consideration every possible monitor, printer, and input device available for computers.
- To achieve it, a device driver for each hardware device is written once.
- It can be supplied by Microsoft (as it includes a large variety of hardware drivers with Windows), the application vendor, or the user.
- Now, the application instructs Windows to draw a filled rectangle, for example, and Windows worries about how to accomplish it on the installed hardware.

Windows Advantages(contd.)

- **Dynamic Link Libraries (DLL)**

➢ Supports much of Windows' functionality.

➢ Enhance the base operating system by providing a powerful GUI.

Layout of a Window

- **Border**
- **Title Bar**
- **Control Icon**
- **System Menu**
- **Minimize Icon**
- **Maximize Icon**
- **Close Window Icon**
- **Vertical Scroll Bar,** if desired.
- **Horizontal Scroll Bar,** if desired.
- **Menu Bar(optional)**
- **Client Area**

Windows Graphics Objects

They are collection of data that can be manipulated as a whole and is presented to the user as part of the visual interface.

- Menus
- Title bars
- Control boxes
- Scroll bars
- Icons
- Cursors
- Carets
- Message Boxes
- Windows Dialog Boxes
- Fonts
- Bitmaps
- Pens
- Brushes

How Windows' Applications Handled?

- Windows provides an application program with access to hundreds of function calls, directly or indirectly, through foundation classes.
- These function calls are handled by several main modules-
  - KERNEL- responsible for memory management, , loading and running an application, and scheduling.
  - GDI (graphics device interface)- contains all of the routines to create and display graphics
  - USER modules- takes care of all other application requirements.

The Windows Message Format

- Messages are used to notify a program that an event of interest has occurred.
- Only one message system exists under Windows —the system message queue.
- Each program currently running under Windows also has its own program message queue.
- The USER module must transfer each message in the system message queue to a program's message queue.
- The program's message queue stores all messages for all windows in that program.

Frequently Used Win32 Data Types

CALLBACK	Replaces FAR PASCAL in application's call back routine.
HANDLE	32-bit unsigned integer that is used as a handle.
HDC	Handle to a device context.
HWND	32-bit unsigned integer that is used as the handle to a window.
LONG	32-bit signed integer
LPARAM	Type used for declaration of lParam.
LPCSTR	LPCSTR is the same as LPSTR, but is used for read-only string pointers.
LPSTR	32-bit pointer.
LPVOID	A generic pointer type. It is equivalent to (void *).
LRESULT	Used for the return value of a window procedure.
UINT	An unsigned integer type
WCHAR	A 16-bit UNICODE character. WCHAR is used to represent all of the symbols for all of the world's languages.
WINAPI	Replaces FAR PASCAL in API declarations.
WPARAM	Used for the declaration of wParam.
HINSTANCE	Handle to the current instance

Frequently Used Win32 Structures

MSG	Defines the fields of an input message
PAINTSTRUCT	Defines the paint structure used when drawing inside a window
RECT	Defines a rectangle
WNDCLASS	Defines a window class

Handles

- Used when writing procedure-oriented Windows applications.
- Handle is a unique number that identifies objects like-
- Windows
- Controls
- Menus
- Icons
- Pens
- Brushes
- Memory allocation
- Output devices
- Window instances-Each copy of a program loaded in main memory is called an instance.

Instance Handles

- Windows allows to run more than one copy of the same application at the same time, so operating system needs to keep track of each of these instances.
- It does this by attaching a unique instance handle to each running copy of the application.

Calling Convention for Functions

- The parameters for the function are pushed from the rightmost parameter to the leftmost parameter, in a normal C and C++ fashion.
- Function declarations under 16-bit Windows 3.x included the **PASCAL** modifier, which was more efficient under DOS in which parameters are pushed onto the stack from left to right.
- Windows does not use this modifier for 32-bit applications and instead of it uses _stdcall.
- PASCAL's use in 32-bit windows application will not give error, just only warning that _stdcall is not used

Windows Header File: WINDOWS.H

- Provides a path to over a thousand constant declarations, **typedef** declarations, and hundreds of function prototypes
- Main reasons a Windows application takes longer to compile than a non-Windows C or C++ program is the size of this and associated header files.
- Traditionally, WINDOWS.H is a required **include** file in all C and C++ Windows applications.
- When using the MFC, the WINDOWS.H header file is included via the AFXWIN.H header file.

Windows Application Components

Windows applications contain two common and essential elements-

1. WinMain( ) function
2. Window function.

WinMain( ) Function

- WinMain( ) serves as the entry point for the Windows application
- Acts in a way similar to the main( ) function in standard C or C++ programs.
- Responsible for the following:
- Creating and initiating application's message processing loop
- Performing any required initializations
- Registering the application's **window** class
- Terminating the program

WinMain() (contd.)

- Four parameters are passed to the WinMain( ) function from Windows.
- WinMain function is defined as-

  int _stdcall WinMain(HINSTANCE hInstance , HINSTANCE hPrevInstance, LPSTR lpszCmdLine, int nCmdShow)

hInstance contains the instance handle of the application. This number uniquely identifies the program when it is running under Windows.

hPrevInstance will always contain a NULL indicating that there is no previous instance of this application.

MS-DOS versions of Windows (Windows 3.3 and earlier) used hPrevInstance to indicate whether there were any previous copies of the program loaded. Under operating systems, such as Windows 95, 98, and NT, each application runs in its own separate address space. For this reason, under Windows 95, 98, and NT, it returns just NULL.

lpszCmdLine is a long pointer to a null-terminated string that represents the application's command-line arguments. Normally, lpszCmdLine contains a NULL if the application was started using the Windows Run command.

nCmdShow defines the possible ways a window can be displayed, such as SW_SHOWNORMAL, SW_SHOWMAXIMIZED, or SW_MINIMIZED.

WNDCLASS

- **Window** class serves as a template to defines attributes of combination of user-selected styles, fonts, caption bars, icons, size, and so on.
- WinMain( ) function registers the application's main **window** class.
- Same standard C and C++ structure type is used for all Windows class definitions.
- Predefined **window** classes are available, but most programmers define their own **window** class.

An example WNDCLASS

- Following example is taken directly from WINUSER.H, which is an **#include** file referenced in WINDOWS.H.
- The header file contains a **typedef** statement defining the structure type WNDCLASSW (a UNICODE-compatible definition), from which WNDCLASS is derived:

```
typedef struct tagWNDCLASSW {
  UINT        style;              //
  WNDPROC     lpfnWndProc;
  int         cbClsExtra;
  int         cbWndExtra;
  HANDLE      hInstance;
  HICON       hIcon;
  HCURSOR     hCursor;
  HBRUSH      hbrBackground;
  LPCWSTR     lpszMenuName;
  LPCWSTR     lpszClassName;
} WNDCLASSW, *PWNDCLASSW, NEAR *NPWNDCLASSW, FAR *LPWNDCLASSW;
```

Fields of WNDCLASS

- Style

  The style field names the **class** style. The styles can be combined with the bitwise OR operator.

Frequently Used Windows Styles

CS_HREDRAW	Redraws the window when horizontal size changes.
CS_VREDRAW	Redraws the window when the vertical size changes
CS_GLOBALCLASS	States that the window class is an application
CS_NOCLOSE	Inhibits the close option from the system menu
CS_SAVEBITS	Saves that part of a screen that is covered by another window
CS_CLASSDC	Provides the window class a display context
CS_SAVEBITS	Saves that part of a screen that is covered by another window

Fields of WNDCLASS (contd)

lpfnWndProc

- Receives a pointer to the window function that will carry out all of the tasks for the window.

cbClsExtra

- Gives the number of bytes that must be allocated after the **window** class structure. It can be set to NULL.

cbWndExtra

- Gives the number of bytes that must be allocated after the window instance. It can be set to NULL

hInstance

- Defines the instance handle of the application registering the **window** class. This cannot be set to NULL.

hIcon

- Icon to be used when the window is minimized. This can be set to NULL.

hCursor

- the cursor to be used with the application. This handle can be set to NULL. The cursor is valid only within the application's client area.

Fields of WNDCLASS (contd)

hbrBackground

- Identification for the background brush. This can be a handle to the physical brush or it can be a color value. Color values must be standard colors such as-

  COLOR_ACTIVEBORDER
  COLOR_ACTIVECAPTION

  COLOR_WINDOW
  COLOR_WINDOWFRAME
  COLOR_WINDOWTEXT

  COLOR_MENU
  COLOR_MENUTEXT
  COLOR_SCROLLBAR
- If hbrBackground is set to NULL, the application paints its own background.

Fields of WNDCLASS (contd)

lpszMenuName

- Pointer to a null-terminated character string.
- The string is the resource name of the menu.
- This item can be set to NULL.

lpszClassName

- Pointer to a null-terminated character string.
- The string is the name of the **window** class.

Defining a Window Class

- Applications can define WNDCLASS by filling the structure's fields with the information about the window class.

```
WNDCLASS wndclass;
wndclass.lpszClassName=szProgName;
wndclass.hInstance    =hInstance;
wndclass.lpfnWndProc  =WndProc;
wndclass.hCursor      =LoadCursor(NULL,IDC_ARROW);
wndclass.hIcon        =NULL;
wndclass.lpszMenuName =szApplName;
wndclass.hbrBackground=GetStockObject(WHITE_BRUSH);
wndclass.style        =CS_HREDRAW|CS_VREDRAW;
wndclass.cbClsExtra   =0;
wndclass.cbWndExtra   =0;
if (!RegisterClass (&wndclass))
 return 0;
```

Creating a Window

- Window class defines the general characteristics of a window, allowing the same window class to be used for many different windows.
- While the parameters for CreateWindow( ) specify more detailed information about the window.
- Returns the handle of the newly created window. Otherwise, the function returns a NULL value.
- Parameter information falls under the following categories:

- the class,
- title,
- style,
- screen position,
- window's parent handle,
- menu handle,
- instance handle,
- 32 bits of additional information

Showing and Updating a Window

- actually display a window

ShowWindow(hWnd,nCmdShow);

- Handle of the window created by the call to CreateWindow( ) is held in the hWnd parameter.
- Second parameter determines how the window is initially displayed.

SW_SHOWNORMAL

SW_SHOWMAXIMIZED

SW_SHOWMINIMIZED

SW_SHOWMINNOACTIVE

The Message Loop

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

GetMessage( ) Function

- Copies the message into the message structure pointed to by **long** pointer, msg, and sends the message structure to the main body of the program.

  First NULL parameter instructs the function to retrieve any of the messages for any window that belongs to the application.

  The last two parameters, tell GetMessage( ) not to apply any message filters. Message filters can restrict retrieved messages to specific categories such as keystrokes or mouse moves only.

The TranslateMessage( ) Function

- As long as this function is included in the message loop, the keyboard interface will be in effect.
- TranslateMessage( ) function creates an ASCII character message (WM_CHAR) from a WM_KEYDOWN and WM_KEYUP message.
- In this way, virtual-key messages can be converted into character messages with the TranslateMessage( ) function.
- Required only by applications that need to process character input from the keyboard.
- This ability can be very useful because it allows the user to make menu selections without having to use the mouse.

The DispatchMessage( ) Function

- Windows sends current messages to the correct window procedures with the DispatchMessage( ) function.