Windows

- A graphical environment
- Earlier versions of Windows ran on MS-DOS or PC-DOS version 3.1 or later
- Windows 95/98 contains its own underlying support and will not run on any other version of DOS.
- Window NT, 2000, XP and later variants are complete operating systems without any underlying DOS

32-bit support

- Versions of Windows since Windows 95 and Window NT represent a new and significant improvement over the earlier versions (Windows 3.1 and Windows for Workgroups 3.11)
- They support programs that use 32-bit instructions and memory addressing
- This makes programming simpler and programs run faster.
- Also support preemptive multi-tasking

The windowing environment

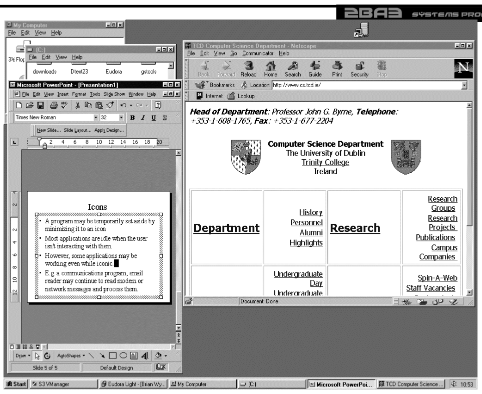
- Windows is a graphical, multi-tasking environment
- I.e. mutiple applications can run simultaneously.
- Each application displays all output in a rectangular area of the screen called a Window

Desktop

- The entire computer screen is referred to as the desktop.
- A user can arrange windows on the desktop in a manner similar to placing pieces of paper on an actual desk.
- You can run two programs in adjacent windows while other programs are temporarily set aside

Icons

- A program may be temporarily set aside by minimizing it to an icon
- Most applications are idle when the user isn't interacting with them.
- However, some applications may be working even while iconic.
- E.g. a communications program, email reader may continue to read modem or network messages and process them.



My Computer
File Edit View Help
3½ Flop
(C:)
File Edit View Help
downloads
Dtext23
Eudora
gstools
Microsoft PowerPoint - [Presentation1]
File Edit View Insert Format Tools Slide Show Window Help
Times New Roman
32
New Slide... Slide Layout... Apply Design...
Draw
AutoShapes
Slide 5 of 5
Default Design
TCD Computer Science Department - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Guide Print Security Stop
Bookmarks
Location: http://www.cs.tcd.ie/
Internet Lookup
Document: Done
Start
S3 VManager
Eudora Light - [Brian Wy...
My Computer
(C:)
Microsoft PowerPoi...
TCD Computer Science ...
10:53

Icons

- A program may be temporarily set aside by minimizing it to an icon
- Most applications are idle when the user isn't interacting with them.
- However, some applications may be working even while iconic.
- E.g. a communications program, email reader may continue to read modem or network messages and process them.

Head of Department: *Professor John G. Byrne,* ***Telephone***: *+353-1-608-1765,* ***Fax***: *+353-1-677-2204*

Computer Science Department
The University of Dublin
Trinity College
Ireland

Department	History Personnel Alumni Highlights	**Research**	Research Groups Research Projects Publications Campus Companies
	Undergraduate Day Undergraduate		Spin-A-Web Staff Vacancies

Windows programming

- Rich programming environment
- supplies extensive support for developing easy-to-use and consistent user interfaces.
- Menus, dialog boxes, list boxes, scroll bars, push buttons, and other components of a user interface are supplied to the developer by Windows.

Device independence

- Windows supplies device-independence
- This allows you to write programs without having detailed knowledge of the hardware platform on which they will eventually run.
- This device independence applies to all the hardware devices on a PC. (e.g. keyboard, mouse, printer, communication ports, graphical devices).

The operating system

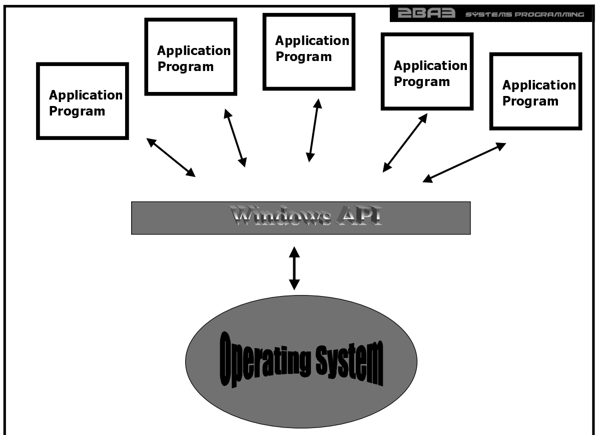
- The Operating System is a system of programs that perform a variety of tasks. E.g:
  - Communicating with peripherals
  - Coordinating concurrent processing of jobs
  - Memory Management
  - Program and Data management.

Communicating with the OS

- Usually, the operating system works behind the scenes, taking care of business without the knowledge or intervention of the user.
- However, it is often necessary for a user to communicate directly with the OS. E.g.
  - Opening applications
  - Printing, copying, saving, deleting.
- This can be done manually, via:
  - Command-line Interface (e.g. MS-DOS)
  - Graphical User Interface (e.g. MS-Windows)

Applications Program Interface - API

- Sometimes, you may wish to access the Operating System directly from a program.
- This is called systems programming.
- With each Operating system is associated a Applications Program Interface (API).
- The API consists of functions that allow the programmer to communicate with the Operating System
- E.g. You can delete a file by
  - Typing a command at a prompt, e.g. >>del x.txt
  - Clicking on the icon of the file, and choosing delete from a menu
  - Calling a delete function from an applications program, sending the name of the file to delete as a parameter.



What is Win32?

- Win32 is the Application Programs Interface (API) for Windows 95 and later variations
- It consists of a family of Windows programming interfaces.
- It is specified in terms of 32-bit values, rather than the 16-bit values used to program Windows 3.x

Windows vs MSDOS

- A Windows program must share system resources, MSDOS applications generally don't need to share.
- Windows applications produce graphical output, MSDOS applications usually produce text.
- Windows applications accept input and manage memory quite differently from MSDOS.
- Windows apps do not require detailed hardware knowledge, whereas MSDOS applications often work on only one type of device.

Resource Sharing

- Windows applications share the resources of the system on which they run.
- Resources such as main memory, the processor, the display, keyboard, hard disks, and drives are shared by all applications running under Windwos.
- To accomplish this sharing, an application must interact with the resources of the computer only through the appropriate Windows API.
- Accessing the resources only through the API enables Windows to control those resources.
- This restriction enables multiple programs to run concurrently in the Windows environment.

Resources and MSDOS

- MSDOS applications typically expect all the resources of the system to be available to the application.
- It can allocate all available memory without concern for other applications.
- It can directly manipulate the serial and parallel ports.
- MSDOS applications do not need to be designed to share system resources.

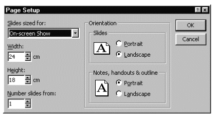
Graphical User Interface

- Windows applications typically run with a GUI
- Each application is given access to the graphical display through a window.
- It is through this window that an application interacts with the user.
- Windows is very focused on running interactive programs with a single user.
- A windows application may have several windows open simultaneously on the desktop.

Input Controls

- Rich set of input controls provided as part of the GUI

E.g. dialog boxes, combo boxes, buttons, menus, scroll bars, tab controls, up-down ("spin") controls.



Slides sized for: On-screen Show

Width: 24 cm

Height: 18 cm

Number slides from: 1

Slides: ○ Portrait ● Landscape

Notes, handouts & outline: ● Portrait ○ Landscape

- Dialog boxes appear on the screen to display information to the user and to request input
- Push buttons, check boxes, and radio buttons allow the user to select true/false, on/off types of program options.
- Menus can be displayed as necessary.
- Scroll bars enable a user to indicate which portion of output should be displayed in the current window.

Input Facilities

- How User input is handled is one of the biggest differences between Windows and MSDOS.
- Windows controls the input devices and distributes input from the devices among each of the multiple concurrent applications as required.
- A Windows application cannot read an input device such as the mouse of keyboard on demand.
- A Windows application is structured to accept input whenever it is produced by the user, rather than demand it when it is required by the application.
- I.e. you cannot use commands such as cin.get()
- In practice, you can still have a fair amount of control what and when the user gets to input, presuming you take the effort.

Event-driven Programming

- A Windows program is ‘told’ whenever a user presses or releases a key.
- As the mouse cursor moves across an application's window, a stream of messages pours into the application, keeping it informed as to the mouse cursor's current location.
- When a mouse button is pressed or released, Window notifies the application of the event and tells it where in the window the mouse cursor was pointing when the event occurred.
- While the user is selecting from a menu, checking a check box, pushing a push button, or scrolling a scroll bar, the application receives a constant flow of messages updating is as to exactly what the user is doing.
- This type of programming is called event-driven programming, because the program responds to events , such as a keyhit, as they occur.

Memory Management

- Memory is a resource that applications share
- A standard MSDOS program assumes that the entire computer's memory is available for use
- Multiple Windows applications must share available memory.
- A Windows application should not initially reserve all memory required by and application, and hold it until the application terminates.
- Instead, it should allocate only the storage required at any given time, and free it as soon as it is no longer needed.

Device-Independent Graphics

- Windows provides device-independent graphical operations.
- A Windows program which can draw lines and circles on a specific graphics display will also draw the figures correctly on all graphical displays supported by Windows.
- It will also display correctly on a dot-matrix printer, a laser printer, a plotter, or other output devices supported by Windows (honest).

The Windows Programming Model

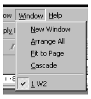
- Windows applications are programs that react to different forms of user input and provide graphical output
  - Any window display by an application must react to user actions.
  - Menus must pop down and enable selections
  - Check boxes must check and uncheck themselves
- Objects in general in Windows should react when manipulated.
- This allows a conceptual view of a Windows applications as a collection of objects.

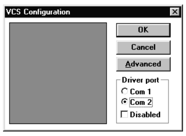
Objects in Windows

- There are many objects in windows, Including:
  - **Pens** - object that have a width, a colour, and a dash style, and are used to draw lines

  - **Brushes** - objects that have a colour, and leave a certain pattern when used to paint areas

  - **Menus** and **dialog boxes**.





Driver port
○ Com 1
● Com 2
□ Disabled

The window object

- However, the first and most important object is the **window** object.
- A window displayed on the screen has data associated with it
  - a background colour
  - a title
  - a menu
  - ... and many other attributes
- When a user depresses a mouse button while the cursor is pointing somewhere within the window, the window is sent a notification of the event, and the window decides how to react.

Polymorphism

- OO GUI programming encourages polymorphism.
- This is the ability to take on many different forms or behaviours.
- E.g. one window may react in one manner to a depressed mouse button, whereas another window might react quite differently to the same action.
- The window might even decide to ignore the event completely.

Inheritance

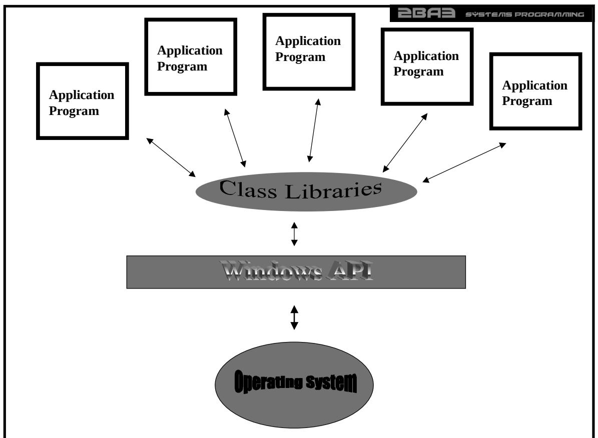
- OO GUI programming also uses inheritance.
- It is often convenient in Windows programming to create a new type of window just like an existing one, except with additional, reduced, or slightly different functionality.
- You can use existing windows as templates for designing new ones, rather than create a new window each time.
- Therefore, if you are modifying the behaviour of built-in controls on an inherited window, you program only the changes in functionality you deem important, and accept all the original functionality you require as given.

Object Libraries

- Microsoft, Borland and others provide libraries that define C++ objects representing windows and all their attributes such as scroll bars and dialog boxes.
- Microsoft Foundation Classes (MFC) and Borlands Object Windows Library (OWL), are collections of C++ classes and methods that give you a high-level programming interface to Windows. (I.e. they add another layer on top of the API).

Object Libraries - cont.

- The facilities provided by these libraries are not part of the Windows API. They are implemented in application code, which in turn makes calls to the API.
- These libraries allow some slick programming effects (tiled windows, tool palettes), and hide some of the more difficult API detail from the programmer,
- However, unless you understand the underlying Windows mechanisms and the capabilities of the API, using the libraries can be quite difficult.



Characteristics of a window

- When a program creates a window, that window has certain characteristics:
  - it location on the screen, its size, a title, a menu etc..
- These are its **physical** characteristics
- A window has other characteristics which determine how it reacts to notifications of various events
- These are its **behavioural** characteristics

Window functions and messages

- All windows have an associated function, called the **window function**.
- This function determines how the window reacts to the notification of an event.
- The notification itself is called a **message**.
- Things like buttons and scroll-bar controls, themselves contained within a window, are also viewed as windows.
- They too have an associated window function that controls how they react.
  - E.g: A scroll bar "window" reacts to a mouse click by sending a message to its parent window, notifying it of the request to scroll.

Messages

Where do they come from?

- Messages to a window originate from many sources
  - from the windows operating environment
  - from another window within the application
  - from another application
  - from itself !
    - a windows application may send itself a message to do something in the future. Later, when the message arrives, the application performs the desired action.

Messages

What kinds of messages are there?

- Most messages are notifications of some external event
  - Something has happened: e.g. a mouse click, or a key press
  - Such notifications are sent to a selected window.
  - This window may then send notifications to other windows
- Windows are also sent messages that are requests for information from the window
  - a message may ask a window what its status is, or request other information
- Some messages tell a window to do something,
  - e.g. change its colour, or some other feature of its appearance

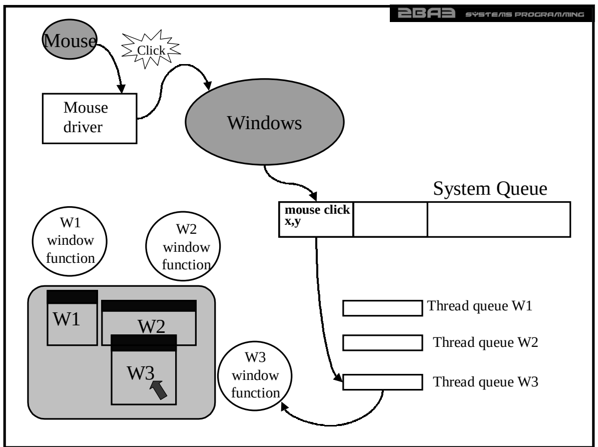
Messages

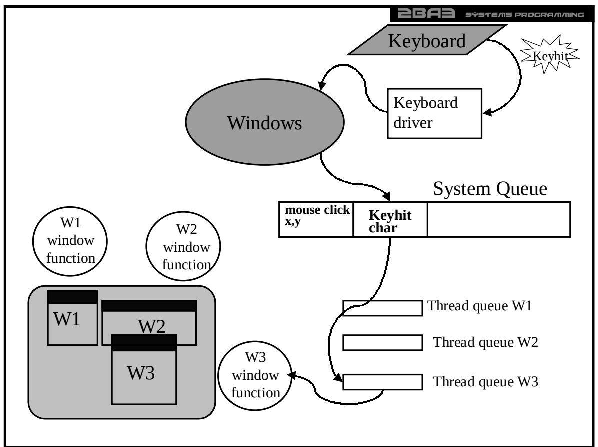
How does the window get them?

- Delivering messages in the proper sequence to the proper destination is the key to Windows operation.
- Many messages in Windows originate from devices, and are handled by device drivers.
  - Device drivers are programs written to handle communication with specific hardware devices. You usually get this program supplied by the manufacturer of the device.
  - E.g. depressing and releasing a key on the keyboard generates interrupts that are handled by the keyboard device driver, a mouse click interrupt is handled by the mouse device driver.
- These device drivers call Windows to translate the hardware event into a message
- The resulting message is then places into the Windows **system queue**.

Windows Queues

- There are two types of queues in Windows: **System** and **Thread**
- There is only one **system queue**.
- Hardware events that are converted into messages are placed into the **system queue**.
- Messages reside in the **system queue** only briefly.
- Each running Windows application has its own unique **thread queue**.
- Windows transfers the messages in the system queue to the appropriate **thread queue**.
- The messages are processed by the application in first-in-first-out (FIFO) order.





Sharing Resources

- Windows implements the sharing of the shared resources (e.g. mouse, keyboard) by using the system queue.
- When an event occurs, a message is placed into the system queue.
- Windows must then decide which thread queue should receive the message.

Input Focus

- Windows uses the concept of the **input focus** to decide which thread should receive the message.
- The input focus is an attribute possessed by only one window in the system at a time.
- The window with the input focus is the focal point for all keyboard input.
- Mouse messages are usually sent to the window that is underneath the mouse pointer.

Keyboard Messages

- Keyboard messages are moved from the system queue into the thread queue for the thread with the window that presently has the input focus.
- As the input focus moves from window to window, Windows moves keyboard messages from the system queue to the proper thread queue
- This is performed on a message-by-message basis because certain keystrokes are requests to change the input focus from one window to another window
- Subsequent messages in the system queue would then go to a different thread queue

Mouse messages

- Mouse messages are handled a little differently from keyboard messages
- They usually are sent to the window that is underneath the mouse pointer.
- When multiple windows are overlapped, the one on the top receives the mouse message.
  - It is possible, however, for a Windows application program to "**capture**" the mouse.
  - This forces all messages from the mouse to be sent to that application's thread queue, no matter where the mouse is pointing.
  - The application must be careful to release the captured mouse eventually, to allow other applications to use it.

Message Loop

- A program's thread queue will fill up with messages.
- The program needs to get the message from the thread queue, and deliver it to the proper window function.
- In order to do this, the programmer writes a small piece of code called the **message loop**.
- The message loop retrieves input messages from the application's thread queue and dispatches them to the appropriate window functions.

Message Loop - cont.

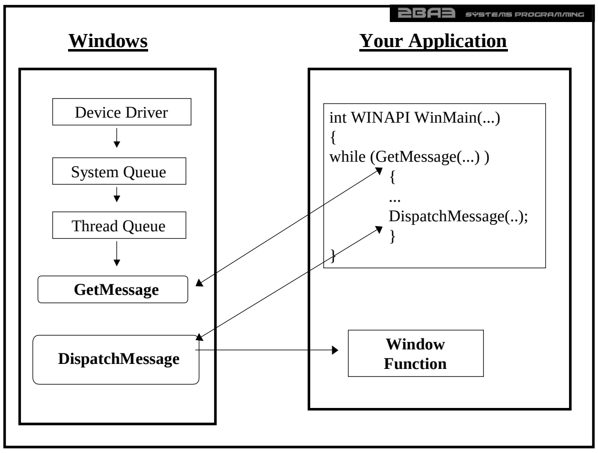
- The message loop continuously retrieves and dispatches messages until it retrieves a special messages that signals that the loop should terminate.
- One message loop is the main body of a Windows application.
- A Windows application initializes, and then repeatedly executes the message loop logic until instructed to stop, and then terminates.

Getting Messages

- The program uses the Win32 API function GetMessage to retrieve a message from its thread queue
- Windows moves the message from the queue into a data area within the program.
- Now the program has the message, but it still needs to be sent to the proper window function.
- I.e. the message must be **dispatched** to the appropriate function.

Dispatching Messages

- A program must call the Win32 API function DispatchMessage in order to dispatch the message received to the right window function.
- Why do you call Windows to send a message to a window function within your own program?
  - A program may create more than one window
  - Each window may have its own unique window function, or multiple windows may use the same window function
  - Many window functions may not be in your program at all, but inside Windows itself
  - Calling DispatchMessage hides all this complexity by determining for you which of the program's window functions, or Window's built-in window functions gets the message.
  - It then calls the proper window function directly.

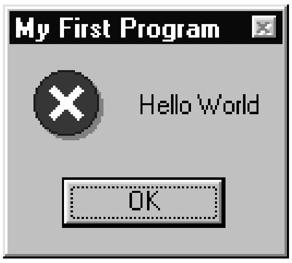


```
int WINAPI WinMain(...)
{
while (GetMessage(...) )
        {
        ...
        DispatchMessage(..);
        }
}
```

A Simple Windows Program

```
#include <windows.h>

int WINAPI WinMain(HANDLE ghInstance, HANDLE prevInstance,
                    LPSTR lpCmdLine, int nCmdShow)
{
 Beep(100,100);
 MessageBox(NULL,"Hello World",
           "My First Program",MB_OK|MB_ICONHAND);
 return 0;
}
```



My First Program

Hello World

Windows Programs

- This program is a fully working Windows program.
- However, all it can do is display simple messages.
  - All the work for creating and displaying the window is done by Windows.
  - There is no functionality you can add to a message box apart from the options provided in the arguments of the command
  - The owner of the window is Windows, not your program.
- What you really want is to have your own window which you can
  - fill with more complex contents such as text, graphics, menus, and buttons, and which the user can resize minimise or maximise.

The windows.h Header File

- One of the first things you notice about a Windows program is that many of the data types used are not standard C++ data types
- These types are defined in an include file that must be included in every Windows program.
- All Windows programs must start with the pre-processor directive:

```
#include <windows.h>
```

- This include file in its turn includes nearly 30 other include files, thereby defining the Win32 API.

- The files included via windows.h contain mainly three types of statements:
  - #define
  - typedef
  - function prototypes
- NOTE::
  - NO CLASSES!
  - The Windows API is not object oriented, although the Windows concept is.
  - We will have to make it object oriented, either by using class libraries, or by making our own classes.

Conditional Compilation

- windows.h and its components are very large for an include file.
- Many applications do not use much of the information available in windows.h.
- Therefore it is possible to conditionally exclude parts of windows.h from processing.
- This is achieved via pre-processor directives, e.g.:

```
#define NOCOLOR
#define NOMENUS
#define NOSCROLL
```

- The preprocessor checks if various options have been selected, and only includes the files that are needed.
- This then reduces the size of your program.

Some components of windows.h

- windef.h
  - defines the base windows types: BOOL, RECT, POINT, NULL, TRUE, etc....
- winbase.h
  - The base API, file symbols, communication port symbols, and most other constants.
- winnetwk.h
  - the network API
- wingdi.h
  - The Graphics Device Interface (GDI) API.
- etc. etc...

windows.h typedef Declarations

- windows.h uses many typedef declarations.
- typedef statements are used to declare types for which C++ does not have a native type.
  - E.g. Non-standard C++, and ANSI C do not have a native Boolean type.(ANSI C++ does).
  - Although an int may be used as a Boolean, confusion may arise.
  - Defining and using an explicit Boolean type, BOOL, eliminates this confusion.

```
#include <windows.h>
BOOL var;
var = TRUE;
if(var){.....}
```

Handles

- An important data type defined by windows.h is a **handle**.
- A handle to some data is used to reference the data a bit like a pointer.
- A Windows handle is like a token, or claim check.
  - E.g. If you go to the theatre or a restaurant, and give your coat to the cloakroom attendant, you receive a token that you later use to retrieve your belongings.
  - Handles are used in the same way.

Window handles

- When you want to create a window in a Windows application, you call the **CreateWindow** function
- This function returns a "window handle" that identifies the created window
- You use this window handle whenever you ask Windows to perform some action on behalf of that window.
- Whenever Windows sends your window function a message about a window, the window handle is included as part of the message to identify the window to which the message applies.

Other Handles

- Handles are used throughout Windows.
- They provide a way to reference items that are managed by the operating system.
- Most Windows objects are identified by a handle:
  - windows: e.g. HWND window;
  - icons: e.g. HICON myicon;
  - menus: e.g. HMENU mymenu;
  - cursors: eg. HCURSOR mycursor;
  - bitmaps: e.g. HBITMAP mybitmap;
  - etc....

Standard prefixes for Windows variables

• b	BYTE, an 8-bit unsigned integer
• ch	CHAR, a character
• d	double
• dw	DWORD, a 32-bit unsigned integer
• f	BOOL, Boolean (nonzero true, 0 false)
• hwnd	HWND, a windows handle
• h	a handle to an object
• l	LONG, 32-bit signed integer
• lp	pointer to a LONG
• n	short, a 16-bit signed integer
• w	WORD, a 16-bit unsigned integer
• sz	CHAR array, null-terminated character string.

Examples

fSuccess, chAnswer, szFilename,
wParam, dwValue, hwndMyWindow

- NOTE. This variable naming convention is just a convention, and even Microsoft doesn't adhere to its own convention everywhere. In fact, it actually leads sometimes to errors.
- Therefore, many programmers don't include the type as a prefix, but instead choose to adopt a different naming convention for variables

Overview of traditional Win32 programming

- Write the **WinMain** function
- **Register** the **window class**
- Write the **window function**
- Create the application window
- Show the window
- Write the **message loop**

The WinMain function

- The entry point for a windows program is the function WinMain

```
int WINAPI WinMain(HINSTANCE hInst,
HINSTANCE hPrevInst,
LPSTR lpCmdLine, int nCmdShow)
```

- The first parameter hInst is a handle for the current instance of the program.
- You can start most windows programs multiple times,and you will get a different window every time.

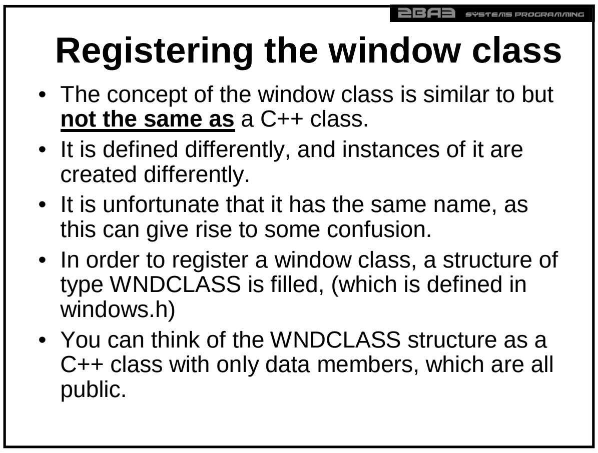
- Internally Windows only loads the program's code once to save memory space.
- However, it gives every instance its own data area and stack, and you don't have to bother about whether there is one or many instances of your program.
- This handle is very important as it is needed as a parameter for various functions.

- The second parameter to WinMain is hPrevInst.
- Under Win16, this is a handle to the previous instance of the program, if any, otherwise it is NULL.
- In Win32 this parameter is obsolete and is therefore always NULL
- lpCmdLine may contain a file name, which your program can check and open it.
- This is because many users start programs by double clicking on a document file in a program such as file manager or explorer.
- If the file extension of the document has been assigned to your application, the above explorer programs will then call your program and specify the documents file name in the lpCmdLine parameter.

- The nCmdShow parameter is a flag which indicates how windows in your application should be shown.
  - SW_SHOWNORMAL if the windows should be shown in its given size
  - SW_SHOWMINIMIZED if it should be shown as an icon
  - SW_SHOWMAXIMISED if it should fill the screen
  - any other SW_constants in <windows.h>
- Usually nCmdShow is set to SW_SHOWNORMAL, but the user can specify otherwise in the program manager.

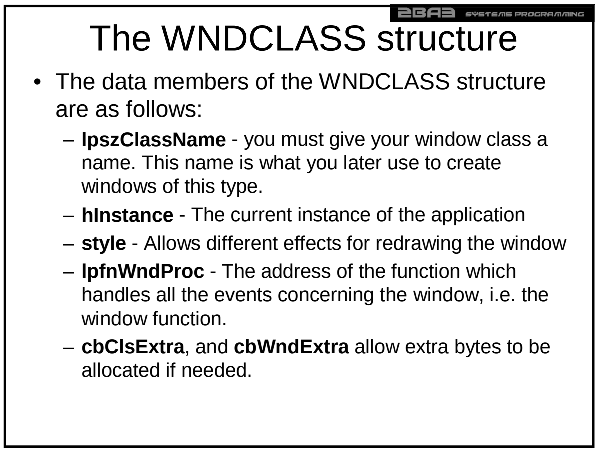
Classifying a Window

- A window class defines general properties for all windows that are derived from this class.
- I.e. once a class has been defined, you can create any number of windows based on this description, and they will all have certain things in common like
  - the background colour,
  - the icon that is shown when the window is minimized
  - Most importantly...the Windows procedure which handles all events concerning the window
- Therefore, before we can create any windows in our application, we must first **register** the window class



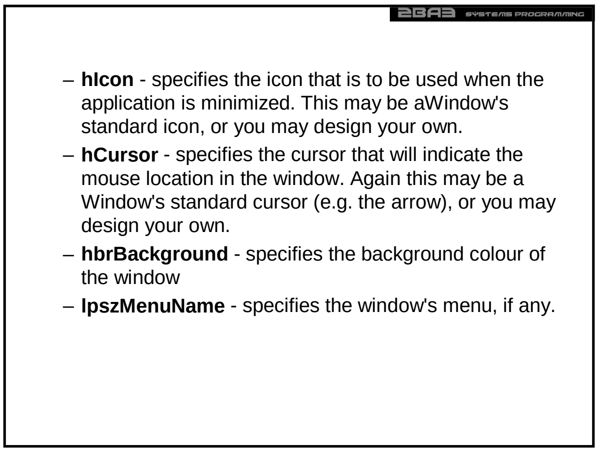
Registering the window class

- The concept of the window class is similar to but **not the same as** a C++ class.
- It is defined differently, and instances of it are created differently.
- It is unfortunate that it has the same name, as this can give rise to some confusion.
- In order to register a window class, a structure of type WNDCLASS is filled, (which is defined in windows.h)
- You can think of the WNDCLASS structure as a C++ class with only data members, which are all public.

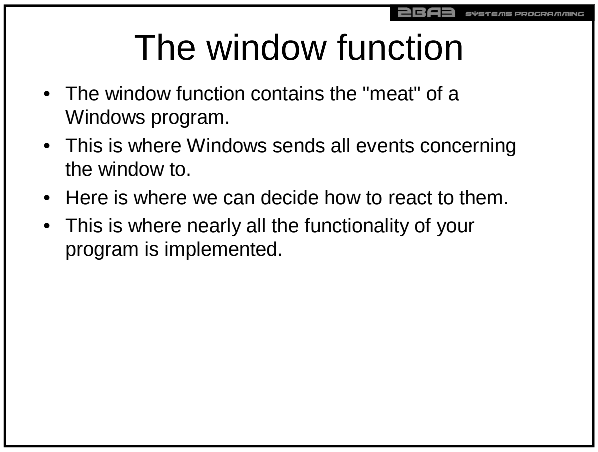


The WNDCLASS structure

- The data members of the WNDCLASS structure are as follows:
  - **lpszClassName** - you must give your window class a name. This name is what you later use to create windows of this type.
  - **hInstance** - The current instance of the application
  - **style** - Allows different effects for redrawing the window
  - **lpfnWndProc** - The address of the function which handles all the events concerning the window, i.e. the window function.
  - **cbClsExtra**, and **cbWndExtra** allow extra bytes to be allocated if needed.

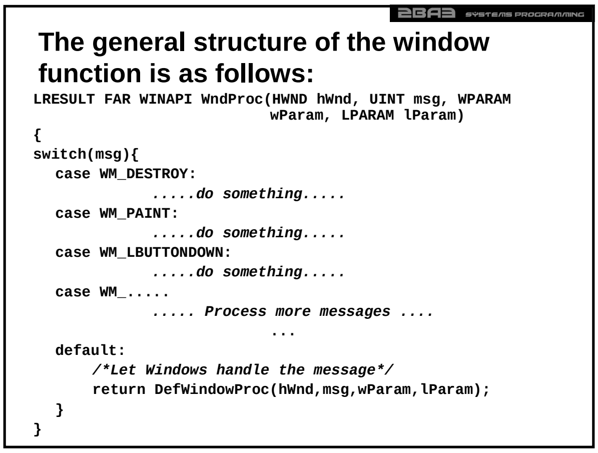


- **hIcon** - specifies the icon that is to be used when the application is minimized. This may be aWindow's standard icon, or you may design your own.
- **hCursor** - specifies the cursor that will indicate the mouse location in the window. Again this may be a Window's standard cursor (e.g. the arrow), or you may design your own.
- **hbrBackground** - specifies the background colour of the window
- **lpszMenuName** - specifies the window's menu, if any.



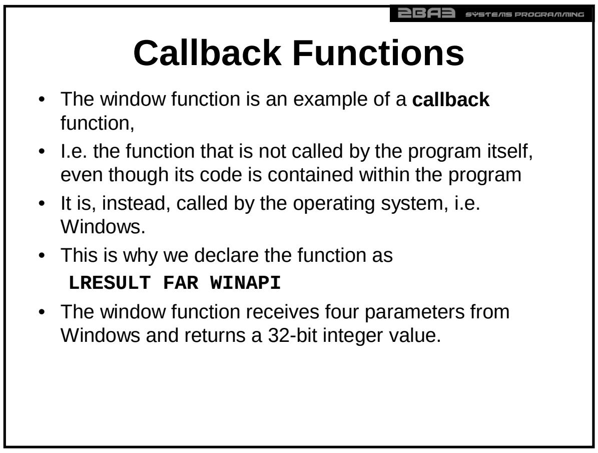
The window function

- The window function contains the "meat" of a Windows program.
- This is where Windows sends all events concerning the window to.
- Here is where we can decide how to react to them.
- This is where nearly all the functionality of your program is implemented.



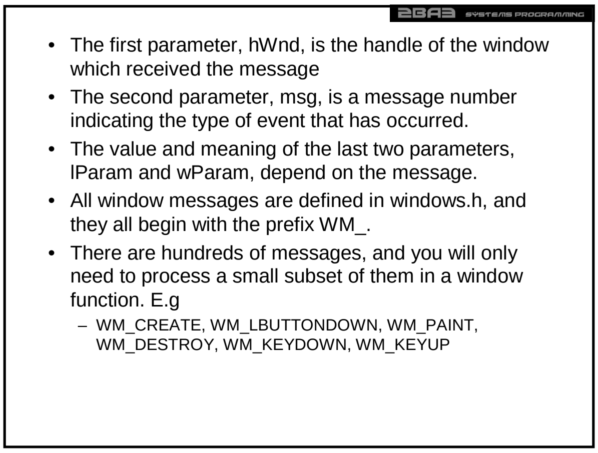
The general structure of the window function is as follows:

```
LRESULT FAR WINAPI WndProc(HWND hWnd, UINT msg, WPARAM
                          wParam, LPARAM lParam)
{
switch(msg){
   case WM_DESTROY:
             .....do something.....
   case WM_PAINT:
             .....do something.....
   case WM_LBUTTONDOWN:
             .....do something.....
   case WM_.....
             ..... Process more messages ....
                          ...
   default:
        /*Let Windows handle the message*/
        return DefWindowProc(hWnd,msg,wParam,lParam);
   }
}
```



Callback Functions

- The window function is an example of a **callback** function,
- I.e. the function that is not called by the program itself, even though its code is contained within the program
- It is, instead, called by the operating system, i.e. Windows.
- This is why we declare the function as `LRESULT FAR WINAPI`
- The window function receives four parameters from Windows and returns a 32-bit integer value.



- The first parameter, hWnd, is the handle of the window which received the message
- The second parameter, msg, is a message number indicating the type of event that has occurred.
- The value and meaning of the last two parameters, lParam and wParam, depend on the message.
- All window messages are defined in windows.h, and they all begin with the prefix WM_.
- There are hundreds of messages, and you will only need to process a small subset of them in a window function. E.g
  – WM_CREATE, WM_LBUTTONDOWN, WM_PAINT, WM_DESTROY, WM_KEYDOWN, WM_KEYUP