LSN 4
GUI Programming Using The
WIN32 API

ECT362 Operating Systems
LSN 4 – Why program GUIs?

• This application will help introduce you to using the Win32 API
• Gain familiarity with using advanced libraries in C/C++
• Demonstrate the formation and format of typical Windows applications
• GUI’s are cool
LSN 4 – GUI Terminology

• **Window**
  – An area of the screen controlled by an application

• **Control**
  – GUI object used for controlling the application
  – Usually generate events

• **Child**
  – A sub-window or control of a parent window

• **Handle**
  – A reference to a position in computer memory
LSN 4 – Win32

- Uses many new data types defined in `windows.h`
  - `HINSTANCE` is a new type, its a handle to an instance
  - `LPSTR` is a long pointer to a string which normally resolves to `char*`
  - `LPCSTR` indicates a pointer to a const string
  - `UINT` in an `unsigned int`
LSN 4 – WinMain()

- Programs using the Win32 API no longer use main() but instead use WinMain() as the application entry point
- WinMain() responsibilities
  - Creating a new window class, a template for our application's main window
  - Registering that class with Windows, so we can create a new window based on it
  - Creating our main window based on our new window class
  - Displaying that window on the screen
  - Retrieving messages from our application's message queue, and passing them to our window procedure
LSN 4 – WinMain()

- **WinMain()**
  
  ```c
  int WINAPI WinMain( HINSTANCE hInstance,
  HINSTANCE hPrevInstance,
  LPSTR lpCmdLine,
  int nCmdShow )
  ```

  - **HINSTANCE hInstance**
    - Handle to the programs executable module (the .exe file in memory)

  - **HINSTANCE hPrevInstance**
    - Always NULL for Win32 programs

  - **LPSTR lpCmdLine**
    - The command line arguments as a single string. NOT including the program name

  - **int nCmdShow**
    - An integer value which may be passed to ShowWindow()
To create a window as you want it, you need to define a class of window and register it with Windows. This allows us to combine some flags to describe the low level style for our window. Tell Windows which of our functions is going to handle messages sent to this window. Handle to application instance that we got in the first parameter of WinMain(). Icon associated with window. Cursor that will be displayed over our window. Name to identify the class with.
Once window class has been defined and registered, new windows can be created using it.

```c
/* Create a window based on our new class */
hwnd = CreateWindow(szAppName, // class name the window is to be based on
    "My 1st Window", // the title of the window that will
    // appear in the bar at the top
    WS_OVERLAPPEDWINDOW, // window style (a window that
    // has a caption, a system menu,
    // a thick frame and a minimise
    // and maximise box)
    /* Use default starting location for window */
    CW_USEDEFAULT, // initial x position (top left corner)
    CW_USEDEFAULT, // initial y position (top left corner)
    440, // initial width
    220, // initial height
    NULL, // window parent (NULL for not a child window)
    NULL, // menu (NULL to use class menu)
    hInstance, // the program instance passed to us
    NULL); // pointer to any parameters wished to be
    // passed to the window producer when the window
    // is created

if(hwnd == NULL) // check to see if an error occurred in creating the window
{
    MessageBox(NULL, "Window Creation Failed!", "Error!",
        MB_ICONEXCLAMATION | MB_OK);
    return 0;
}

/* Show and update our window */
ShowWindow(hwnd, SW_SHOW);
UpdateWindow(hwnd);
```

Create a standard MS Windows message window with no owner and an exclamation-point icon in the window with an ‘ok’ button.
LSN 4 – The Message Loop

• All user operations with the new window are sent via messages

```c
/* Retrieve and process any queued messages until we get WM_QUIT */
/* Recall that Windows uses a messaging system to notify window of */
/* user actions */
while ( GetMessage(&msg, NULL, 0, 0) )
{
    TranslateMessage(&msg); // for certain keyboard messages
    DispatchMessage(&msg); // send message to WndProc
}

/* Exit with status specified in WM_QUIT message */
return msg.wParam;
} // end WinMain()
```

msg is a pointer to a message structure that Windows fills out automatically.
LSN 4 – WndProc()

- **WndProc()**
  ```c
  LRESULT CALLBACK WndProc( HWND hwnd,
                           UINT iMsg,
                           WPARAM wParam,
                           LPARAM lParam )
  ```

  - **HWND hWnd**
    - The unique handle of the window
  
  - **UINT iMsg**
    - This is the message your window has received
  
  - **WPARAM wParam / LPARAM lParam**
    - Each message sent to your window can have a number of extra values
LSN 4 – The Window Function

/* Function | WndProc() */
/* Description | Whenever anything happens to your window, Windows will call this function telling you what has happened. */
/* | The message parameter contains the message sent */
/* Inputs | None */
/* Output | Integer value 0 */

RESULT CALLBACK WndProc( HWND hwnd,
UINT iMsg,
WPARAM wParam,
LPARAM lParam )

FAINTSTRUCT ps;
MDC hdc;

/* Switch according to what type of message we have received */
switch ( iMsg ) {
    case WM_PAINT:
        /* We receive WM_PAINT every time window is updated */
        hdc = BeginPaint(hwnd, &ps);
        TextOut(hdc, 10, 160, "My First Window", 15);
        EndPaint(hwnd, &ps);
        break;
    case WM_CREATE:
        /* Operations to be performed when this window is created */
        break;
    case WM_DESTROY:
        /* Window has been destroyed, so exit cleanly */
        PostQuitMessage(0);
        break;
    case WM_COMMAND:
        /* User selected a command from a menu or a control sent a message */
        break;
    default:
        /* We do not want to handle this message so pass back to Windows */
        /* to handle it in a default way */
        return DefWindowProc(hwnd, iMsg, wParam, lParam);
    }

    return 0;

Conditionally decide what action to take based upon the received message type
Message sent when window needs redrawing
Message sent when window is created
Message sent if the user has closed the window
Message is sent when a menu item or associated control is selected
Must return this value of a message is not being handled, else window will not appear
LSN 4 – Win32 Projects in MSVS

• Create a new Visual C++ Win32 Project (not console) named Win1 Proj
  – Remove all source files and header files except Resource.h
  – Move Win1 Proj.rc from “Resource Files” to “Source Files”
  – Add ECT361_Win1.cpp to “Source Files”
  – Remove “Using Precompiled Headers” option in project properties
LSN 4 – Getting Fancy

• Changing the window class’s associated icon
  – Must update the icon members of the windows class
    
    wndclass.hIcon = LoadIcon( GetModuleHandle(NULL),
                            MAKEINTRESOURCE(IDI_MYICON));
    wndclass.hIconSm = (HICON)LoadImage( GetModuleHandle(NULL),
                                MAKEINTRESOURCE(IDI_MYICON),
                                IMAGE_ICON, 16, 16, 0);

• Update the project resource script (*.rc)
  – Use notepad to add new icon in icon section of file
    
    IDI_MYICON ICON "myicon.ico"

• Update the project resource (resource.h)
  
    #define IDI_MYICON 129
LSN 4 – Preparations for Next Class

• Prior to next class be sure your window program builds and runs (you will use this program throughout next class)
  – Spend time getting familiar with the MSVS help function
  – Spend some time adjusting your window class and see how the changes affect its visualization
  – Investigate your program and associated window using SPY++
LSN 4 – Adding Window Controls

• A Control is a child window, an application used in conjunction with another windows to perform simple input and output (I/O) tasks
• Many controls available for use in windows
  – Button
  – ComboBox
  – ListBox
  – Static
  – ImageLists
  – EditBox
LSN 4 – Buttons

• A control that the user can click to provide input to an application

• Button creation steps:
  – Create an ID for the button window
    
    ```
    #define ID_BUTTON 1001
    ```

  – Create button control (window) as a child to our window
    • Use the `CreateWindow()` method for the `WM_CREATE` case within `WndProc()`
    • Must declare a handle within `WndProc()` for child window (button)

  – Respond to messages from the button by monitoring the `WM_COMMAND` messages from with `WndProc()`
    • `if( HIWORD(wParam) == BN_CLICKED ),` evaluate `LWORD(wParam)` for `ID_BUTTON`
LSN 4 – Buttons

• To create an “ADD” button to our window

```c
static HWND addbutton;

/* Switch according to what type of message we have received */
switch ( iMsg )
{
    case WM_PAINT:
        /* We receive WM_PAINT every time window is updated */
        hdc = BeginPaint(hwnd, &ps);
        TextOut(hdc, 10, 160, "My First Window", 15);
        EndPaint(hwnd, &ps);
        break;
    case WM_CREATE:
        /* Operations to be performed when this window is created */
        /* Create a child window for a pushbutton */
        addbutton = CreateWindow( "BUTTON", // predefined class
                                "ADD", // button text
                                WS_VISIBLE | WS_CHILD | BS_PUSHBUTTON,
                                // Size and position values are given
                                // explicitly, because the CW_USEDEFAULT
                                // constant gives zero values for buttons.
                                10, // starting x position
                                10, // starting y position
                                100, // button width
                                30, // button height
                                hwnd, // parent window
                                (HMENU)ID_ADDBUTTON, // No menu
                                (HINSTANCE) 0, // ignored for Windows XP
                                NULL); // pointer not needed
        break;
```
LSN 4 – Buttons

The action taken upon receiving a message from a button is up to the programmer . . . what needs to be done
LSN 4 – Edit Boxes

• A rectangular control window typically used in a dialog box to permit the user to enter and edit text by typing on the keyboard

• Edit Box creation steps:
  – Same creation as for buttons
  – Utilize Win32 library functions to retrieve text entered and its properties

```
len = GetWindowTextLength(editbox);
GetWindowText(editbox, string, len + 1);
```

Handle to edit box
String to store text into
Number of characters to read (must add 1 for ‘\0’)

Handle to edit box
String to store text into
Number of characters to read (must add 1 for ‘\0’)

sschneider@udayton.edu
LSN 4 – List Boxes

• A control window that contains a list of items from which the user can choose (or just view)

• List Box creation steps:
  - Same as for buttons except not reading messages from it, only interested in sending messages to it
  - Messages typically sent based upon external event (button press, etc.)

```c
SendDlgItemMessage( hwnd, ID_LIST, LB_ADDSTRING, 0, (LPARAM)"Added" ) ;
```

- Handle to the window that contains the control
- Specifies the identifier of the control that receives the message
- Specifies the message to be sent
- Specifies additional message-specific information
LSN 4 – Homework

• Assignment – HW 2
  – Complete HMWRK2.pdf

• References
  – Help within MSVS for specific function syntax descriptions
  – Programming Windows®, Fifth Edition