HELLOWIN.C

The HELLOWIN.C program, a representative example of Windows programming, is predominantly composed of overhead that is common to virtually every Windows program.

In practice, Windows programmers seldom commit the entirety of this syntax to memory.

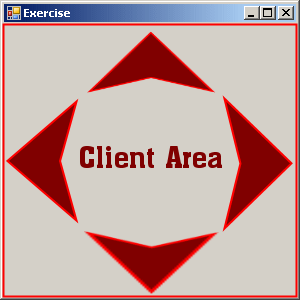
Instead, a common approach is to initiate a new program by duplicating an existing one and subsequently making the necessary modifications.



This flexible method allows for an efficient utilization of existing code structures, a practice explicitly encouraged by the author.

In the earlier mention of HELLOWIN, the assertion that it displays the text string in the center of its window is clarified.

The actual placement is in the center of the program's "client area," delineated in Figure 3-2 as the expansive white space within the title bar and the sizing border.



This distinction is underscored as crucial since the client area represents the canvas within the window where a program can freely draw and present visual output to the user.

Remarkably, despite its relatively concise 80-odd lines of code, HELLOWIN incorporates a myriad of functionalities.

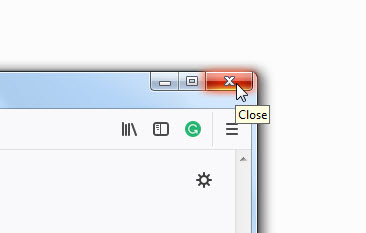
These include the ability to manipulate the window by dragging the title bar or resizing it by interacting with the sizing borders.



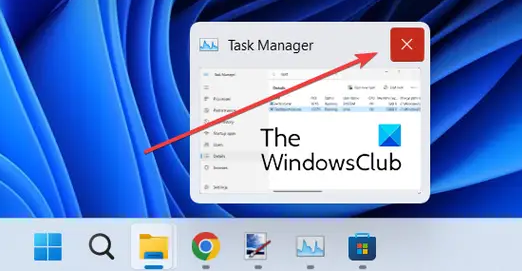
Notably, the program dynamically adjusts the position of the text string to the center of its client area when the window size changes.

Additionally, users can maximize the window to occupy the entire screen, minimize it to remove it from view, and access these options not only through the window buttons but also via the system menu situated at the far left of the title bar.

The program's versatility extends to various interaction modes, such as closing the window to terminate the program.



This termination can be accomplished through multiple avenues, including selecting the Close option from the system menu, clicking the close button at the far right of the title bar, or employing a double-click on the system menu icon.



Looking ahead, the text anticipates an in-depth examination of HELLOWIN.C in the subsequent chapters.

Despite having a WinMain function akin to the sample programs in the initial chapters, it introduces a second function, WndProc, denoted as the window procedure or colloquially referred to as the "win prock" among Windows programmers.

Notably, there is no explicit code within HELLOWIN.C that calls WndProc.

However, its reference in WinMain necessitates its declaration near the program's outset, setting the stage for a comprehensive exploration of its role and significance in the ensuing discussions.

**18 WINDOWS FUNCTIONS THAT HELLOWIN.C CALLS**

LoadIcon

Loads an icon for use by a program. An icon is a small image that is used to identify a program or file. The LoadIcon function loads an icon file into memory and returns a handle to the icon.

LoadCursor

Loads a mouse cursor for use by a program. A mouse cursor is a small image that is displayed on the screen when the mouse is moved. The LoadCursor function loads a cursor file into memory and returns a handle to the cursor.

GetStockObject

Obtains a graphic object, in this case a brush used for painting the window's background. A graphic object is a resource that is used to draw graphics on the screen. The GetStockObject function retrieves a predefined graphic object from the system.

RegisterClass

Registers a window class for the program's window. A window class is a template that defines the characteristics of a window, such as its size, style, and background color. The RegisterClass function registers a window class with the system.

MessageBox

Displays a message box. A message box is a pop-up window that is used to display a message to the user. The MessageBox function creates and displays a message box.

CreateWindow

Creates a window based on a window class. The CreateWindow function creates a window based on a window class that was previously registered with the system.

ShowWindow

Shows the window on the screen. The ShowWindow function makes a window visible on the screen.

UpdateWindow

Directs the window to paint itself. The UpdateWindow function sends a message to a window telling it to repaint itself.

GetMessage

Obtains a message from the message queue. A message is a notification that is sent to a window by the operating system. The GetMessage function retrieves a message from the message queue.

TranslateMessage

Translates some keyboard messages. The TranslateMessage function converts certain keyboard messages into Windows messages.

DispatchMessage

Sends a message to a window procedure. The DispatchMessage function sends a message to the window procedure for the window that received the message.

PlaySound

Plays a sound file. The PlaySound function plays a sound file.

BeginPaint

Initiates the beginning of window painting. The BeginPaint function prepares a window for painting.

GetClientRect

Obtains the dimensions of the window's client area. The GetClientRect function retrieves the dimensions of the client area of a window.

DrawText

Displays a text string. NThe DrawText function displays a text string on the screen.

EndPaint

Ends window painting. The EndPaint function completes the painting of a window.

PostQuitMessage

Inserts a "quit" message into the message queue. The PostQuitMessage function inserts a "quit" message into the message queue. This message tells the program to terminate.

DefWindowProc

Performs default processing of messages. The DefWindowProc function performs default processing of messages that are not handled by the window procedure.

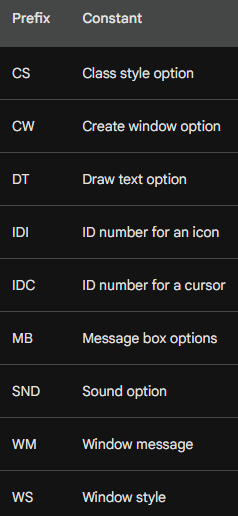
UPPERCASE IDENTIFIERS

Uppercase identifiers are used in HELLOWIN.C because they are defined in the Windows header files.

Several of these identifiers have a two-letter or three-letter prefix followed by an underscore.

The prefix indicates a general category to which the constant belongs, as indicated in the table below.

**Table of Prefixes**

****

**Examples**

**CS\_HREDRAW -** Specifies that the window should be redrawn when its client area is resized.

**DT\_VCENTER -** Specifies that text should be displayed in the center of the rectangle specified by the DrawText function.

**SND\_FILENAME -** Specifies that the PlaySound function should play the sound file specified by the filename parameter.

**CS\_VREDRAW -** Specifies that the window should be redrawn when its vertical scroll bar is moved.

**IDC\_ARROW -** Specifies the standard arrow cursor.

**WM\_CREATE -** Sent when a window is created.

**CW\_USEDEFAULT -** Specifies that the default size and position should be used for the window.

**IDI\_APPLICATION -** Specifies the application's default icon.

**WM\_DESTROY -** Sent when a window is destroyed.

**DT\_CENTER -** Specifies that text should be centered horizontally.

**MB\_ICONERROR -** Specifies that the message box should have an error icon.

**WM\_PAINT -** Sent when a window's client area needs to be painted.

**DT\_SINGLELINE -** Specifies that text should be drawn on a single line.

**SND\_ASYNC -** Specifies that the PlaySound function should play the sound file asynchronously.

**WS\_OVERLAPPEDWINDOW -** Specifies that the window should have a title bar, a minimize button, a maximize button, a system menu, and a sizing border.

**Numeric Constants**

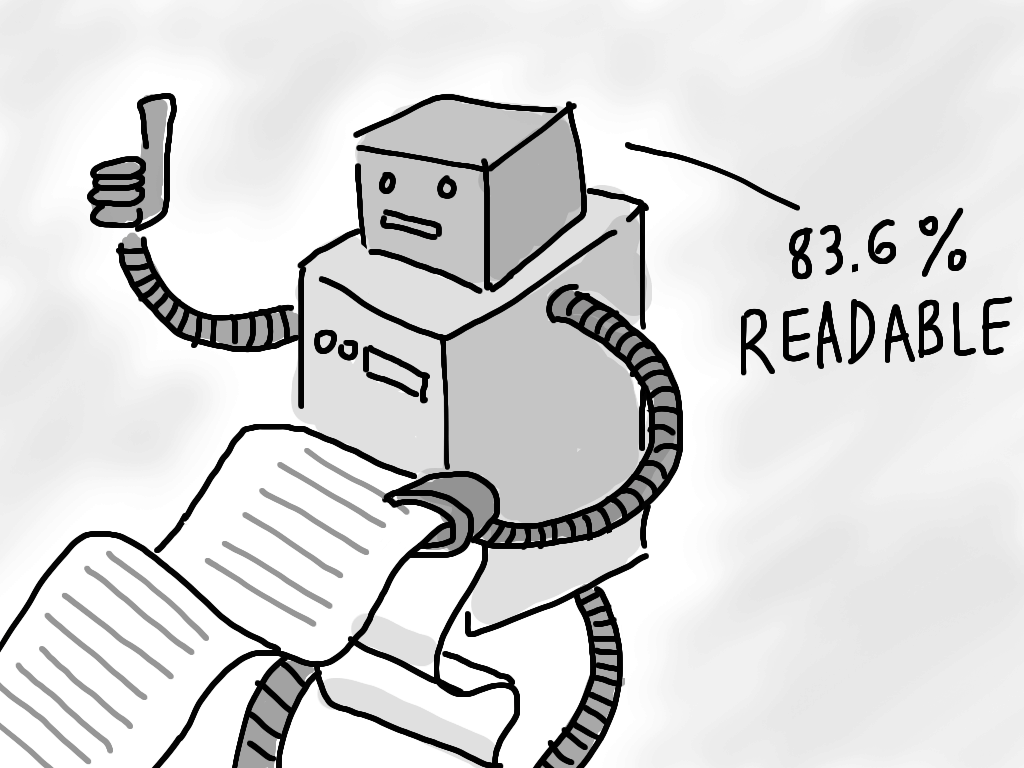
Uppercase identifiers are simply numeric constants.

You almost never need to remember numeric constants when programming for Windows because virtually every numeric constant has an identifier defined in the header files.

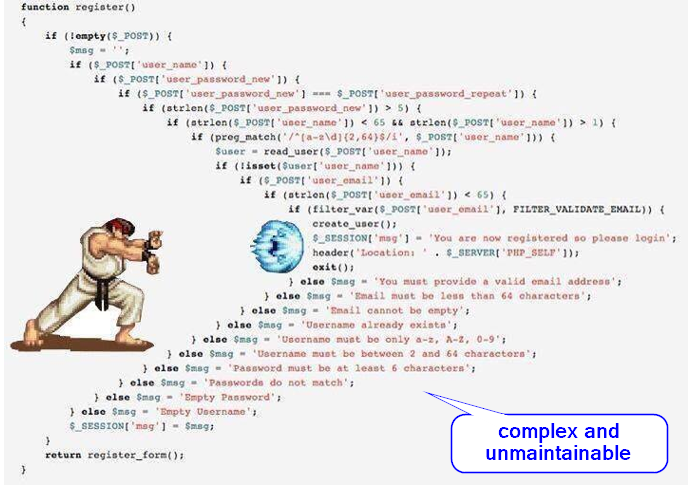


**Benefits of Using Uppercase Identifiers**

**Code Readability:** Uppercase identifiers make code more readable and understandable. This is because they provide meaningful names for numeric constants, which can be difficult to remember and understand.



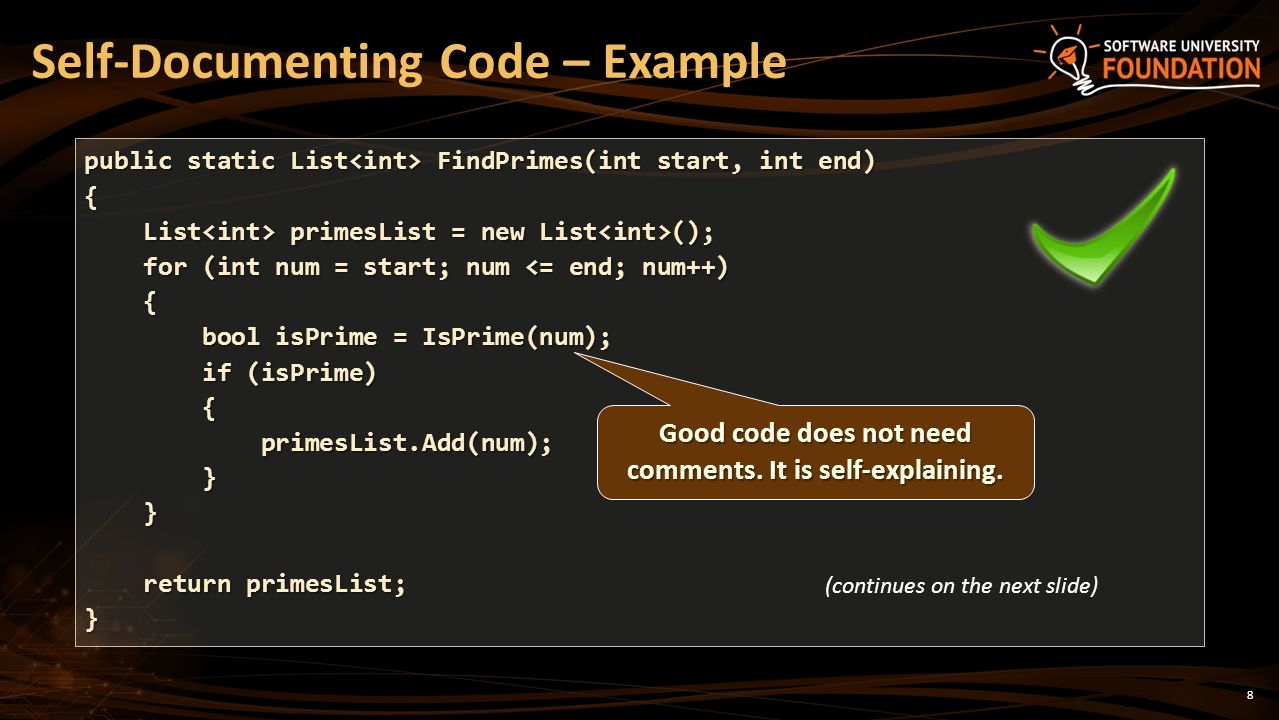
**Code Maintainability:** Uppercase identifiers make code more maintainable. This is because they make it easier to understand and change code, especially for programmers who are not familiar with the code.



**Error Reduction:** Uppercase identifiers can help to reduce errors in code. This is because they make it more likely that programmers will use the correct constant for a particular situation.



**Self-Documenting Code:** Uppercase identifiers serve as a form of self-documenting code. By providing descriptive names for constants, developers are essentially embedding documentation within the code itself, making it easier for others to understand the purpose of different variables and values.



**Consistency with Programming Conventions:** The use of uppercase identifiers is a well-established convention in many programming languages, particularly in Windows programming. Following these conventions improves code consistency and makes it easier for developers to collaborate and understand each other's work.

