First Program

HELLOMSG.C

```c
/*------------------------------------------------------------
HelloMsg.c -- Displays "Hello, Windows 98!" in a message box
(c) Charles Petzold, 1998
------------------------------------------------------------*/
#include <windows.h>

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance, PSTR szCmdLine, int iCmdShow)
{
    MessageBox (NULL, TEXT ("Hello Windows98!"), TEXT ("MyFirstGUI"), 0) ;
    return 0 ;
}
```

HEADER FILES

The header files you mentioned are all important for writing Windows programs in C. They define all the Windows data types, function calls, data structures, and constant identifiers.

WINDEF.H: Basic type definitions.
WINNT.H: Type definitions for Unicode support.
WINBASE.H: Kernel functions.
WINUSER.H: User interface functions.
WINGDI.H: Graphics device interface functions.

You can search through these header files using the Find In Files option from the Edit menu in Visual Studio 2022 Community. You can also open the header files in the Developer Studio and examine them directly.

PROGRAM ENTRY POINT:

The program entry point for a Windows program is the WinMain() function. It is declared in the WINBASE.H header file.

The third parameter was changed from **LPSTR** to **PSTR**. These two data types are both defined in WINNT.H as pointers to character strings. The LP prefix stands for "long pointer" and is an artifact of 16-bit Windows.

Two of the parameter names from the WinMain declaration were changed

to follow Hungarian notation. This system involves prefacing the variable name with a short prefix that indicates the variable's data type. The prefix **i** stands for **int**, and **sz** stands for **"string terminated with a zero."**

The **WinMain function is declared as returning an int.** The WINAPI identifier is defined in WINDEF.H and specifies a calling convention that involves how machine code is generated to place function call arguments on the stack. **Most Windows function calls are declared as WINAPI.**

```
int WINAPI WinMain(
    HINSTANCE hInstance,
    HINSTANCE hPrevInstance,
    LPSTR lpCmdLine,
    int nShowCmd
);
```

hInstance: A **handle** to the current instance of the program. The first parameter to WinMain is an instance handle, which is a number that an application uses to identify itself. It is required as an argument to some other Windows function calls.

hPrevInstance: A **handle** to the previous instance of the program, if there was one. The second parameter to WinMain is hPrevInstance, which is a handle to the previous instance of the program, if there was one. In 32-bit versions of Windows, this parameter is always NULL.

lpCmdLine: A **pointer** to the command line string that was passed to the program when it was started. The third parameter to WinMain is the command line used to run the program. Some Windows applications use this to load a file into memory when the program is started.

nShowCmd: A **flag** that specifies how the program window should be initially displayed. The fourth parameter to WinMain indicates how the program should be initially displayed. It can be displayed normally, maximized, or minimized.

You can also write the code this way:

```
int WINAPI WinMain(
    HINSTANCE hInstance,
    HINSTANCE hPrevInstance,
    LPSTR lpCmdLine,
    int nShowCmd
) {
    MessageBox(NULL, TEXT("Hello, Windows!"), TEXT("HelloMsg"), 0);
    return 0;
}
```

MESSAGEBOX FUNCTION

In the Windows equivalent of the "hello, world" program, the WinMain() function simply calls the MessageBox() function to display a message box with the text "Hello, Windows!".

The MessageBox() function is used to display a message box to the user. It takes four arguments:

hParent/Window Handle (HWND): hParent is a handle to the window with which the message box is associated. Typically, the first argument to MessageBox is an HWND, a reference to a window. This handle is used to specify the parent window of the dialog box.

lpText/Message Text: lpText is a pointer to a text string that will be displayed in the body of the message box. The second argument of MessageBox is where you provide the text string that appears in the main content of the message box. This is the actual message you want to convey to the user.

lpCaption/Caption Text: lpCaption is a pointer to a text string displayed in the caption bar of the message box. The third argument in MessageBox is where you specify the text string for the caption, which typically serves as the title or heading for the dialog box.

uType/Dialog Box Style: uType is a set of flags that determine the appearance and behavior of the message box. The fourth argument in MessageBox is a combination of constants defined in WINUSER.H. These constants define the style and behavior of the message box, including aspects like which buttons to display, the default button, and whether to include icons.

It includes options like specifying which buttons should appear

(e.g., MB_OK for an OK button), which button should be the default, **and** whether to display an icon with the message.

The fourth argument, uType, can be a combination of constants beginning with the prefix MB_ that are defined in WINUSER.H.

These **constants** specify the buttons that will be displayed in the message box, the icon that will be displayed in the message box, and the default button.

The uType argument can be a combination of the following constants:

MB_OK: Displays an OK button.
MB_OKCANCEL: Displays an OK and Cancel button.
MB_ABORTRETRYIGNORE: Displays an Abort, Retry, and Ignore button.
MB_YESNOCANCEL: Displays a Yes, No, and Cancel button.
MB_YESNO: Displays a Yes and No button.
MB_RETRYCANCEL: Displays a Retry and Cancel button.
MB_ICONHAND: Displays a stop sign icon.
MB_ICONQUESTION: Displays a question mark icon.
MB_ICONEXCLAMATION: Displays an exclamation point icon.
MB_ICONASTERISK: Displays an asterisk icon.
MB_DEFBUTTON1: Makes the first button the default button.
MB_DEFBUTTON2: Makes the second button the default button.
MB_DEFBUTTON3: Makes the third button the default button.
MB_DEFBUTTON4: Makes the fourth button the default button.

The MessageBox() function returns the ID of the button that was clicked. If the user clicks the Cancel button or closes the message box, the MessageBox() function returns IDCANCEL.

Example:

The following code shows how to use the MessageBox() function to display a message box with the text "Hello, world!".

```c
#include <windows.h>

int main()
{
    MessageBox(NULL, TEXT("Hello, world!"), TEXT("HelloMsg"), MB_OK);
    return 0;
}
```

This code will display a message box with the text "Hello, world!" and an OK button. When the user clicks the OK button, the program will exit.

The MessageBox() function is a simple but powerful function for displaying messages to the user. It can be used to display a variety of messages, from simple informational messages to complex messages with multiple buttons.

Here are some common options for specifying buttons and default buttons:

```c
#define MB_OK 0x00000000L
#define MB_OKCANCEL 0x00000001L
#define MB_ABORTRETRYIGNORE 0x00000002L
#define MB_YESNOCANCEL 0x00000003L
#define MB_YESNO 0x00000004L
#define MB_RETRYCANCEL 0x00000005L

#define MB_DEFBUTTON1 0x00000000L
#define MB_DEFBUTTON2 0x00000100L
#define MB_DEFBUTTON3 0x00000200L
#define MB_DEFBUTTON4 0x00000300L
```

Additionally, you can specify icons to be displayed alongside the message using options like MB_ICONHAND, MB_ICONQUESTION, MB_ICONEXCLAMATION, and MB_ICONASTERISK.

```
#define MB_ICONHAND 0x00000010L
#define MB_ICONQUESTION 0x00000020L
#define MB_ICONEXCLAMATION 0x00000030L
#define MB_ICONASTERISK 0x00000040L
```

These icons are often used to convey the nature or importance of the message. It's important to note that the MessageBox function returns a value that corresponds to the user's choice.

For example, if the user clicks the OK button, it returns IDOK, which is defined as 1.

Depending on the buttons available in the message box, it can also return IDYES, IDNO, IDCANCEL, IDABORT, IDRETRY, or IDIGNORE.

While the MessageBox function may not offer the same formatting power as printf, it's an essential tool for displaying information and gathering simple user input in Windows applications.

In the next chapter, you will learn how to create customized message boxes that can offer more formatting options.