C / C++ NOTATION STORY

Hungarian Notation

Hungarian Notation is mainly confined to Microsoft Windows programming environments, such as Microsoft C, C++ and Visual Basic. It was originally devised by **Charles Simonyi**, a Hungarian, who was a senior programmer at Microsoft for many years. Hungarian notation involves storing information about the variable in the very name of the variable. Information such as the data type and variable scope can usually be inferred by looking at the name of the variable. With Hungarian notation, variable names are separated into two parts:

1. The lowercase prefix, which contains information about the variable type, and
2. The qualifier, which tells you what the variable contains. The qualifier usually begins with a capital letter.

For example: `pstrError`

This would be a pointer to a string, and which will contain an error message.

Most of the Windows API, code samples and documentation are written in this notation. Please note that the Hungarian notation is something developed for the C language and not the C++. In C, it is quite simple to come up meaningful prefixes for the "few" types, which make sense in a particular domain (like Windows programming). With C++, it is really more difficult situation.
Hungarian Notation is not really necessary when using a modern strongly-typed language as the compiler warns the programmer if a variable of one type is used as if it were another type. It is less useful in object-oriented programming languages such as C++, where many variables are going to be instances of classes.

CamelCase notation

CamelCase notation is a common name for the practice of writing compound words or phrases where the words are joined without spaces, and each word is capitalized within the compound. This practice is known by a large variety of names, including **BiCapitalization**, **InterCaps**, **MixedCase**, etc. CamelCase is a standard identifier naming convention for several programming languages and also has become fashionable in marketing for names of products and companies. The CamelCase name comes from the uppercase "bumps" in the middle of the compound word, suggesting the humps of a camel.
For example:

```
thisIsCamelCase
thisIsLowerCamelCase
ThisIsUpperCamelCase
```

Variation and Synonym

There are two common varieties of CamelCase, distinguished by their handling of the initial letter. The variety in which the first letter is capitalized is commonly called **UpperCamelCase**, **PascalCase**, or **BiCapitalized**. The variety in which the first letter is left as lowercase is commonly called **lowerCamelCase** or sometimes simply **camelCase**. The term **StudlyCaps** is similar but not necessarily identical to CamelCase. It is sometimes used in reference to CamelCase but can also refer to **random mixed capitalization** (as in "MiXeD CaPitALiZaTioN") as popularly used in online culture. Other synonym examples for CamelCase notation include:

BumpyCaps
BumpyCase
CamelCaps
CamelHumpedWord
CapWordsPython (reference)
mixedCase (for lowerCamelCase) in Python (reference)
ClCl (Capital-lower Capital-lower) and sometimes ClC
HumpBackNotation
InterCaps
InternalCapitalization
NerdCaps
WordsStrungTogether or WordsRunTogether

In programs of any significant size, there is a need for descriptive (hence multi-word) identifiers, like "`previous balance`" or "`end of file`". Writing the words together as in "`endoffile`" is not satisfactory because the names often become unreadable. ASCII character set standard that had been established by the late 1960s, allowing the designers of the C language to adopt the underscore character "_" as a word joiner. Underscore-separated compounds like "`end_of_file`" are still prevalent in C programs and libraries.

Notation Used In Windows programming

Excerpt from Windows documentation...

Variable Names and Hungarian Notation

As programs have become more complex both in terms of size and of the proliferation of data types, many programmers have adopted a variable-naming convention, which is commonly referred to as **Hungarian notation** (apocryphally named in honor of Microsoft programmer, Charles Simonyi). Another notation commonly used in programming language is CamelCase.
Over the past several years, several standard versions of Hungarian notation have been proposed and/or published. The version given here is dictated in part by personal preferences and in part by conventions established by Windows in naming constants, variable, and data structure definitions. Because all of these standards are intended to be mnemonic for your convenience, you may follow or alter them as desired. Using Hungarian notation, variable names begin with **one or more lowercase letters** that denote the **variable type**, thus providing an inherent identification. For example, the prefix `h` is used to identify a handle, as in **`hWnd`** or **`hDlg`**, referring to window and dialog box handles, respectively. In similar fashion, the prefix `lpsz` identifies a `long` pointer to a null-terminated (`ASCIIZ`) string. Table 1 summarizes some of the Hungarian notation conventions.

Prefix	Data type
`b`	boolean.
`by`	byte or unsigned char.
`c`	char.
`cx / cy`	short used as size.
`dw`	`DWORD`, double word or unsigned long.
`fn`	function.
`h`	handle.
`i`	int (integer).
`l`	Long.
`n`	short int.
`p`	a pointer variable containing the address of a variable.
`s`	string.
`sz`	ASCIIZ null-terminated string.
`w`	`WORD` unsigned int.
`x, y`	short used as coordinates.

Table 1: Hungarian Notation Convention examples.

Predefined Constants

Windows also uses an extensive list of predefined constants that are used as messages, flag values, and other operational parameters. These constant values are always full uppercase and most include a two- or three-letter prefix set off by an underscore. Here are some examples:

`CS_HREDRAW`	`CS_VREDRAW`	`CW_USERDEFAULT`
`DT_CENTER`	`DT_SINGLELINE`	`DT_VCENTER`
`IDC_ARROW`	`IDI_APPLICATION`	`WM_DESTROY`
`WM_PAINT`	`WS_OVERLAPPEDWINDOW`	

In the case of constant identifiers, the prefixes indicate the general category of the constant. Table 2 shows the meanings of the prefixes in the examples shown here.

PrefixCategory	Mean
`CS`	Class style
`CW`	Create window

DT	Draw text
IDC	Cursor ID
IDI	Icon ID
WM	Window message
WS	Window style

Table 2: A Few Constant Prefixes.

Data Types

Windows also uses a wide variety of new data types and type identifiers, most of which are defined in either the `WinDef.H` or `WinUser.H` header files. Table 3 lists a few of the more common data type examples.

Data type	Meaning
FAR	Same as `far`. Identifies an address that originally used the `segment:offset` addressing schema. Now `FAR` simply identifies a (default) 32-bit address but may be omitted entirely in many cases.
PASCAL	Same as `pascal`. The `pascal` convention demanded by Windows defines the order in which arguments are found in the stack when passed as calling parameters.
WORD	Unsigned integer (16 bits)
UINT	Unsigned integer, same as `WORD`
DWORD	Double word, unsigned long int (32 bits)
LONG	Signed long integer (32 bits)
LPSTR	Long (far) pointer to character string
NEAR	Obsolete, previously identified an address value within a 16KB memory block.

Table 3: A Few Windows Data Types.

Data Structures

Similarly, Windows adds a variety of new data structures. Again, most are defined in either `WinDef.H` or `WinUser.H`. The examples shown in Table 4.

Structure	Example	Meaning
MSG	`msg`	Message structure.
PAINTSTRUCT	`ps`	Paint structure.
PT	`pt`	Point structure (mouse position).
RECT	`rect`	Rectangle structure, two coordinate pairs.
WNDCLASS	`wc`	Window class structure.

Table 4: Examples of notation used for Windows structures.

Handle Identifiers

In like fashion, a variety of handles are defined for use with different Windows elements. Like constants, the handle **types** use all uppercase identifiers. Table 5 shows a few examples.

Handle type	Examples	Meaning
HANDLE	`hnd or hdl`	Generic handle
HWND	`hwnd or hWnd`	Window handle
HDC	`hdc or hDC`	Device-context handle (CRT)
HBRUSH	`hbr or hBrush`	Paint brush handle
HPEN	`hpen or hPen`	Drawing pen handle

Table 5: Five Handle Identifiers.