Hungarian Reference Sheet

This is a summary of some common Hungarian usage. It is intended to be a supplement to documents describing Hungarian in more detail. It leaves out some useful features in the interest of simplicity.

Identifiers and Defined Constants

Scope + Prefix + Base Type + Qualifier (not all may appear; Base Type should always be used)
Prefix and Base Type in lower case, Qualifier capitalized; e.g. `c` + `ch` + `LastName` = `cchLastName`

Scope

`m_`	member of a class
`g_`	global
`s_`	static

Prefixes

`p`	pointer
`a`	array
`i`	index into an array
`c`	count
`d`	difference
`h`	handle

Base Types

`f`	flag (Boolean)
`ch`	character
`sz`	zero-terminated (C-style) string
`st`	Pascal-type string

New Base Types

2-4 letters, somewhat mnemonic, acronyms common
structs/classes are typically new Base Types

Qualifiers

Used to distinguish multiple instances of a type and convey purpose.
No qualifier is used if the purpose is unambiguous; typically no qualifier on the primary one.
First letter capitalized, rest are lowercase.
Occasionally, more than one qualifier used, e.g. `pchMinSav`.

`Min`	the first element in the set
`Max`	the upper limit of elements in a set (one past the last valid element)
`Mic`	the *current* first element in a set (rarely used)
`Most`	the *current* last element in a set
`Mac`	the *current* upper limit of elements in a set (one past the last current element)
`First`	the first element *to be dealt with*
`Last`	the last element *to be dealt with*
`Lim`	the upper limit of elements *to be dealt with* (one past the last element to be dealt with)

For dealing with members of a set (array):

`Min` <= `Mic` <= `First` <= `Last` <= `Most` < ***`Lim`*** <= ***`Mac`*** <= ***`Max`***

`Lim`, `Mac`, and `Max` refer to invalid elements (one past the last valid element)

`Sav`	temporary saved value
`Nil`	special illegal value
`T`	temporary value (use sparingly; if multiple in the same context, be more descriptive)
`Cur`	the current item
`Src`	source
`Dst`	destination
`Next`	the next item
`Prev`	the previous item

Functions

Begin with the value returned (if any).
Follow return type with statement of what function does (verb + object), e.g. `FInitAch`.
Capitalize the first letter of each word.

Examples

When working with screen coordinates, it's common to use `x` and `y`. The distance between coordinates is expressed using the "`d`" prefix.

```
dxScreen = xMax - xMin;          yTop += dy;
```

When you define a struct or class, its name should be all uppercase, and any variables of that type should use the same letters all lowercase. For example, if you create a struct to hold game state, you might use `GS` (an acronym for Game State).

```
struct  GS {...};                GS gs;
```

Suppose you want to use an array to store people's heights, and decide to use integers to express height in inches. Even though you are using a built-in type (int), you are essentially creating a new Base Type for height since that's what you're manipulating. You could choose to call it something like "`ht`" or "`hgt`".

```
int aht[ihtMax];       for (iht=0; iht<ihtMax; ++iht)          ht=aht[iht];
```

(Note that "`ht`" doesn't become plural when you make an array -- it's *not* `ahts`.)

Consider an array that can hold 20 characters, and currently contains "The quick fox"
Suppose that we want to capitalize all letters in the word "quick"

```
#define ichMax 20          // or const int ichMax = 20;
char ach[ichMax];
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
T	h	e		q	u	i	c	k		f	o	x							

`ichMin`	`0`	index of first *possible* char in ach
`ichMax`	`20`	index just after last *possible* char in ach
`ichMic`	`0`	index of lowest *valid* char in ach
`ichMost`	`12`	index of last *valid* char in ach
`ichMac`	`13`	index just after last *valid* char in ach
`ichFirst`	`4`	index of first char to *modify*
`ichLast`	`8`	index of last char to *modify*
`ichLim`	`9`	index of char just after last char to *modify*

To capitalize letters, we could call a function `ChUpper` (which returns a `ch` that is uppercase) as follows:

```
for (ich=ichFirst; ich<ichLim; ++ich)
       ach[ich]=ChUpper(ach[ich]);
```

(or)

```
for (ich=ichFirst; ich<=ichLast; ++ich)
       ach[ich]=ChUpper(ach[ich]);
```

Consider a struct `FOO` that is a node for a linked list and contains a pointer to the next node, "pfooNext".

```
void OutputFooList(FOO *pfooHead)
{
   FOO *pfoo;
   for (pfoo = pfooHead; pfoo != NULL; pfoo = pfoo->pfooNext)
         // output contents of pfoo...
}
```