

- Jaime Irurzun -

Como muchos conoceréis, la notación húngara es un método propuesto por un antiguo programador de Microsoft, que ofrecía estandarizar la escritura del código fuente para lograr que otros programadores pudieran entender el código fuente escrito por uno mismo, especialmente en proyectos que requieran el trabajo en equipo.

Son muchas las variaciones que la notación original ha sufrido, de forma que cada uno la ha acomodado a su gusto. Yo, como uno más, os propongo mi variación: (Los ejemplos están especialmente dirigidos a la programación en el lenguaje Clipper)

variables

Los nombres de las variables irán precedidas de una letra minúscula que indique el tipo de dato que maneja esa variable. Se seguirán las siguientes reglas:

Variables de tipo carácter (string)	->	cVariable	Ej.: cApellidos
Variables de tipo numérico	->	nVariable	Ej.: nCodigo
Variables de tipo fecha	->	dVariable	Ej.: dAlta
Variables de tipo lógico (booleano)	->	lVariable	Ej.: lSeguro
Variables de tipo objeto	->	oVariable	Ej.: oDlg
Variables de tipo codeblock	->	bVariable	Ej.: bKeyDown
Variables sin tipo determinado	->	xVariable	Ej.: xDato

Esta norma se aplicará también a los nombres de los campos de las bases de datos. Sin embargo, los nombres de los TAGs se diferenciarán por una "t". Ejemplo: tProvincia.

De la misma forma, los arrays irán precedidos de la letra "a" (ej: aDatos). Además, cuando se conozca el tipo de dato que manejarán los elementos y posiciones de un array, a su primera letra "a", la precederá otra letra representativa del tipo de dato. Para ello se seguirán los mismos criterios que para los tipos de datos de las variables. Ejemplos: caApellidos, naCodigos, daFechas, laAsegurados, oaGet... La única excepción está en que para el tipo de dato indeterminado no se empleará la "x", sino que el array únicamente llevará la letra "a". Ejemplo: aDatos.

Para la nomenclatura de las variables objetos de las clases utilizadas con la librería Fivewin, se utilizarán las siguientes abreviaturas:

WINDOW	->	oWnd	METER	->	oMtr
DIALOG	->	oDlg	TABS	->	oTab
BUTTON	->	oBtn	PRINT	->	oPrn
GET	->	oGet	SCROLL BAR	->	oScr
BUTTONBAR	->	oBar	PAGES	->	oPag
BITMAP	->	oBmp	MSGITEM	->	oMsg
CHECKBOX	->	oChk	GROUP	->	oGrp
COMBOBOX	->	oCmb	TREE	->	oTre
RADIOBUTTON	->	oRdo	TIMER	->	oTmr
BROWSE	->	oBrw	CLIPBOARD	->	oClp
LISTBOX	->	oLst	INI FILE	->	oIni
DATABASE	->	oDbf	ODBC	->	oDbc
CURSOR	->	oCur	MAIL	->	oEml
ICON	->	oIco	METAFILE	->	oMtf
FONT	->	oFnt	DDE	->	oDde
BRUSH	->	oBrs	COMM	->	oCom
MENU	->	oMnu	VIDEO	->	oVdo
MENUITEM	->	oMnl	VBX	->	oVbx

funciones del lenguaje

Empezarán en minúsculas. Si el nombre de la función está compuesto por dos o más palabras, el comienzo de cada una se marcará escribiendo la primera letra en mayúscula, y siempre con la primera palabra entera en minúsculas, ejs: *fieldName()*, *msgYesNo()*, *dbSetFilter()*, *fOpen()*, *aAdd()*... Si el nombre de la función está compuesto por una única palabra, está se escribirá totalmente en minúsculas, ejs: *deleted()*, *len()*, *empty()*... Si la primera letra de la función es representativa del tipo de dato que ésta devuelve, la en este caso primera palabra comenzará con letra mayúscula, siempre separando las próximas como antes he mencionado, ejs: *cMonth()*, *nRgb()*, *cGetFile()*...

funciones propias del programa

Empezarán con la primera letra en mayúscula. De esta forma las diferenciamos de las funciones del lenguaje, ejs: *CliAltas()*, *NuevoLibro()*, *Fabrir()*... Para separar las palabras "ocultas" en el nombre de la función seguiremos los mismos procedimientos que para las funciones del lenguaje de programación que usemos, a diferencia de que cuando la primera letra nos indique el tipo de dato que devolverá la función, esta irá también en mayúsculas, de forma que las dos primeras letras quedan escritas en mayúsculas, ejs: *ATipos()*, *NCodigo()*...

Diferenciar las funciones propias del programa y las del lenguaje, ayuda a que si una persona tiene que comenzar a trabajar sobre un código escrito por nosotros, sepa inmediatamente si la función que se está usando la tiene que buscar en el código fuente, o si le bastará con consultar la documentación sobre el lenguaje en concreto.

comandos y sentencias

Absolutamente todas las letras de estos dos tipos de órdenes serán mayúsculas, ejs:

<i>Comandos</i>	-> REDEFINE, CLOSE ALL, GET, @...SAY, SET COLOR, USE...
<i>Sentencias</i>	-> FUNCTION, LOCAL, IF, DO CASE, BEGIN SEQUENCE, RETURN...

directivas

La palabra completa irá en minúsculas, ejs: *#define*, *#include*, *#xcommand*...

operadores

Los operadores deberán tener a izquierda y derecha un espacio en blanco, ejs:

```
nTres := nUno + nDos
nPos := int( ( 80 - len( cTexto ) ) / 2 )
```

signos de puntuación

Aquí diferenciaremos varios comportamientos. Por una parte, las comas deberán tener siempre un espacio a su derecha, aunque no a su izquierda, ej: *< HEADERS "Nombre", "Apellidos", ...>*.

Los paréntesis tendrán un espacio a su izquierda y otro a su derecha aunque a uno de estos lados se encuentre otro paréntesis, ej: *< ACTION (if(empty(... oDlg:end()))) >*, pero sin embargo los paréntesis que acompañan a los nombres de las funciones, no deberán separarse de éstas, y solo habrá que dejar espacios en su interior en el caso de que lleven parámetros, ej: *< FUNCTION Altas() >*, *< Leer(aDatos, "Clientes") >*, *< FUNCTION Check(cDato, oDbf) >*...

Los corchetes de los arrays no requerirán un espacio para separarlos del nombre del array, pero sí en su interior, donde además de eso, si escribimos un número, es preferible que éste se complete con los ceros suficientes hasta llegar a un número de dígitos que no creamos que vamos a sobrepasar en ningún momento, ejs: *< DECLARE aDatos[] >*, *<aCliente[nCodigo()]>*, *< oaGet[03] >*...

codeblocks

Como sabeis, esto es un ejemplo de un codeblock dentro de una función *aEval()*:

```
aEval( oaBtnBmp, { |oBtn| oBtn:oCursor := oMiCursor } )
```

En estos casos, habrá que dejar un espacio a izquierda y derecha de las llaves que encierran el codeblock. También se dejará un espacio a la izquierda de la primera barra vertical que encierra los parámetros, así como a la derecha de la segunda. Como excepción, en la declaración de parámetros del codeblock no dejaremos ningún espacio.

tabulación

La tabulación tendrá que ir incrementándose según vayan anidándose más estructuras de cualquier tipo (condicionales, bucles...etc) unas dentro de otras. Sólo no habrá tabulación en las directivas indicadas al principio del código fuente y en las sentencias "FUNCTION" y "RETURN", es decir, al principio y final de cada función. Además habrá que dejar un espacio entre el contenido de la función y su comienzo y final. Ej:

```
FUNCTION Clientes()
```

```
    LOCAL .....
```

```
RETURN nil
```

Hasta aquí llega mi proposición sobre la claridad en la escritura del código fuente. Espero que sirva para quienes nunca se han decidido a adoptar una norma, y desde luego, si eres uno de ellos, hazlo, ya que te ayudará increíblemente, y sobre todo a quienes tengan que trabajar contigo.

Jaime Irurzun Graña

www.irusoft.com

Sábado, 10 de Mayo de 2003 - Versión 1.1

Sábado, 26 de Abril de 2003 - Versión 1.0

Bibliografía:

"*La programación elegante*", por Jose Alfonso Suárez Moreno

Disponible en www.olivares2000.org

"*Notación Húngara*", © Aulaware, 1999

Disponible en <http://leo.worldonline.es/acanudas/pdf/fwhungry.pdf>