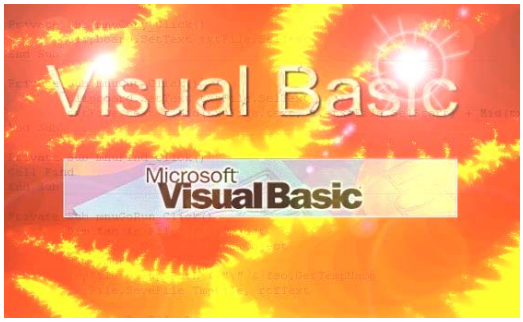
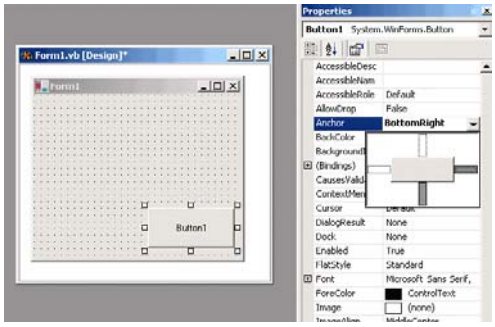
CURSO

Curso Completo de Visual Basic 6.0





Escuela Superior de Ingenieros Industriales
UNIVERSIDAD DE NAVARRA
Javier García de Jalón · José Ignacio Rodríguez
Alfonso Brazález · Patxi Funes · Eduardo Carrasco · Jesús Calleja

7.6 LECTURA Y ESCRITURA EN FICHEROS SECUENCIALES

7.6.1 Apertura y cierre de ficheros

Para poder leer o escribir en un fichero antes debe ser abierto con la sentencia ***Open***, cuya forma general es la siguiente:

```
Open filename For modo As # fileNo
```

donde:

filename es el nombre del fichero a abrir. Será una variable ***string*** o un nombre entre dobles comillas (“ ”).

modo Para acceso secuencial existen tres posibilidades: ***Input*** para leer, ***Output*** para escribir al comienzo de un fichero y ***Append*** para escribir al final de un fichero ya existente. Si se intenta abrir en modo ***Input*** un fichero que no existe, se produce un error. Si se abre para escritura en modo ***Output*** un fichero que no existe se crea, y si ya existía se borra su contenido y se comienza a escribir desde el principio. El modo ***Append*** es similar al modo ***Output***, pero respeta siempre el contenido previo del fichero escribiendo a continuación de lo último que haya sido escrito anteriormente.

fileNo es un número entero (o una variable con un valor entero) que se asigna a cada fichero que se abre. En todas las operaciones sucesivas de lectura y/o escritura se hará referencia a este fichero por medio de este número. No puede haber dos ficheros abiertos con el mismo número. ***Visual Basic*** dispone de una función llamada ***FreeFile*** que devuelve un número no ocupado por ningún fichero.

A continuación puede verse un ejemplo de fichero abierto para lectura:

```
Open "C:\usuarios\PRUEBA1.txt" For Input as #1
```

Después de terminar de leer o escribir en un fichero hay que cerrarlo. Para ello, se utilizara el comando ***Close***, que tiene la siguiente forma:

```
Close # fileNo
```

donde el ***fileNo*** es el número que se la había asignado al abrirlo con la instrucción ***Open***.

7.6.2 Lectura y escritura de datos

7.6.2.1 Sentencia Input

Existen varias formas de leer en un fichero de acceso secuencial. Por ejemplo, para leer el valor de una o más variables se utiliza la sentencia ***Input***:

```
Input # fileNo, varName1, varName2, varName3, ...
```

donde el ***fileNo*** es el número asignado al archivo al abrirlo y *varName1, varName2, ...* son los nombres de las variables donde se guardarán los valores leídos en el fichero.

Debe haber una correspondencia entre el orden y los tipos de las variables en la lista, con los datos almacenados en el fichero. No se pueden leer directamente vectores, matrices o estructuras. Si los datos del disco han de ser escritos por el propio programa, conviene utilizar la sentencia ***write*** (mejor que la ***print***) para garantizar que los valores están convenientemente separados. La sentencia ***write*** se verá posteriormente.

7.6.2.2 Función Line Input y función Input

La función ***Line Input #*** lee una línea completa del archivo y devuelve su contenido como valor de retorno. Su forma general es:

```
varString = Line Input #fileNo
```

Conviene recordar que en los ficheros de texto se suele utilizar el carácter ***return*** (o ***Intro***) para delimitar las distintas líneas. Este es el carácter ASCII nº 13, que por no ser un carácter imprimible se representa en ***Visual Basic 6.0*** como *chr(13)*. En muchas ocasiones (como herencia del MS-DOS) se utiliza como delimitador de líneas una combinación de los caracteres ***return*** y ***linefeed***, representada en ***Visual Basic 6.0*** como *chr(13)+chr(10)*. En la cadena de caracteres que devuelve ***line*** no se incluye el carácter de terminación de la línea.

Para leer todas las líneas de un fichero se utiliza un bucle ***for*** o ***while***. ***Visual Basic 6.0*** dispone de la función ***EOF*** (*End of File*) que devuelve ***true*** cuando se ha llegado al final del fichero. Véase el siguiente ejemplo:

```
Do While Not EOF(fileNo)
```

```
miLinea = Line Input #fileNo
...
Loop
```

También se puede utilizar la función ***Input***, que tiene la siguiente forma general:

```
varString = Input(nchars, #fileNo)
```

donde ***nchars*** es el número de caracteres que se quieren leer y ***varString*** es la variable donde se almacenan los caracteres leídos por la función. Esta función lee y devuelve todos los caracteres que encuentra, incluidos los ***intro*** y ***linefeed***. Para ayudar a utilizar esta función existe la función ***LOF (fileNo)***, que devuelve el nº total de caracteres del fichero. Por ejemplo, para leer todo el contenido de un fichero y escribirlo en una caja de texto se puede utilizar:

```
txtCaja.text = Input(LOF(fileNo), #fileNo)
```

7.6.2.3 Función print

Para escribir el valor de unas ciertas variables en un fichero previamente abierto en modo ***Output*** o ***Append*** se utiliza la instrucción ***print #***, que tiene la siguiente forma:

```
Print #fileNo, var1, var2, var2, ...
```

donde ***var1***, ***var2***,... pueden ser variables, expresiones que dan un resultado numérico o alfanumérico, o cadenas de caracteres entre dobles comillas, tales como *"El valor de x es..."*.

Considérese el siguiente ejemplo:

```
Print #1, "El valor de la variable I es: ", I
```

donde ***I*** es una variable con un cierto valor que se escribe a continuación de la cadena. Las reglas para determinar el formato de la función ***print #*** son las mismas que las del método ***print*** visto previamente.

7.6.2.4 Función write

A diferencia de ***Print #***, la función ***Write #*** introduce comas entre las variables y/o cadenas de caracteres de la lista, además encierra entre dobles comillas las cadenas de caracteres antes de escribirlas en el fichero. La función ***Write #*** introduce un carácter ***newline***, esto es, un ***return*** o un ***return+linefeed*** después del último carácter de las lista de variables. Los ficheros escritos con ***Write #*** son siempre legibles con ***Input #***, cosa que no se puede decir de ***Print #***. Véase el siguiente ejemplo:

```
' Se abre el fichero para escritura
```

```
Open "C:\Temp\TestFile.txt" For Output As #1
Write #1, "Hello World", 234          ' Datos separados por comas
MyBool = False: MyDate = #2/12/1969#  ' Valores de tipo boolean y Date
Write #1, MyBool; " is a Boolean value"
Write #1, MyDate; " is a date"
Close #1                              ' Se cierra el fichero
```

El fichero *TestFile.txt* guardado en *C:\temp* quedaría:

```
"Hello World",234
#FALSE#," is a Boolean value"
#1969-02-12#," is a date"
```

7.7 FICHEROS DE ACCESO ALEATORIO

Los ficheros de acceso aleatorio se caracterizan porque en ellos se puede leer en cualquier orden.
Los ficheros de acceso aleatorio son ficheros binarios. Cuando se abre un fichero se debe escribir ***For Random***, al especificar el modo de apertura (si el fichero se abre ***For Binary*** el acceso es similar, pero no por ***registros*** sino por ***bytes***; este modo es mucho menos utilizado).

7.7.1 Abrir y cerrar archivos de acceso aleatorio

Estos archivos se abren también con la sentencia ***Open***, pero con modo ***Random***. Al final se añade la sentencia ***Len=longitudRegistro***, en bytes. Véase el siguiente ejemplo:

```
fileNo = FreeFile
size = Len(unObjeto)
Open filename For Random as #fileNo Len = size
```

donde *filename* es una variable que almacena el nombre del archivo. Se recuerda que la función FreeFile devuelve un número entero válido (esto es que no está siendo utilizado) para poder abrir un fichero. El último parámetro informa de la longitud de los registros (todos deben tener la misma longitud). ***Visual Basic 6.0*** dispone de la función ***Len(objetoName)***, que permite calcular la dimensión en bytes de cualquier objeto perteneciente a una clase o estructura.

De ordinario los ficheros de acceso directo se utilizan para leer o escribir de una vez todo un bloque de datos. Este bloque suele ser un *objeto* de una *estructura*, con varias *variables miembro*.

Los ficheros abiertos para acceso directo se cierran con ***Close***, igual que los secuenciales.

7.7.2 Leer y escribir en una archivo de acceso aleatorio. Funciones Get y Put

Se utilizan las funciones ***Get*** y ***Put***. Su sintaxis es la siguiente:

```
Get #fileNo, registroNo, variableObjeto
Put #fileNo, registroNo, variableObjeto
```

La instrucción ***Get*** lee un registro del fichero y almacena los datos leídos en una variable, que puede ser un *objeto* de una determinada *clase* o *estructura*. La instrucción ***Put*** escribe el contenido de la variable en la posición determinada del fichero. Si se omite el número de registro se lee (escribe) a continuación del registro leído (escrito) anteriormente. Véase el siguiente ejemplo:

```
FileNo=FreeFile
size=Len(unObjeto)
Open filename for Random as #fileNo Len=size
Get #fileNo, 3, size
```

Con ***este*** ejemplo, se ha abierto el fichero *filename* de la misma forma que se realizó en el ejemplo anterior, pero ahora, además se ha leído un registro de longitud *size*, y más en concreto, el tercer registro. Si se quisiera modificar el valor de este registro, no tendríamos más que asignarle el valor que se quisiera, para a continuación introducirlo en el fichero mediante la sentencia siguiente:

```
Put #fileNo, 3, size
```

7.8 FICHEROS DE ACCESO BINARIO

La técnica a emplear es básicamente la misma que con los ficheros de acceso aleatorio, con la salvedad de que en lugar de manejar ***registros***, en los ficheros de acceso binario se trabaja con ***bytes***.

Véase el siguiente ejemplo:

```
FileNo=FreeFile
Open filename for Binary as #fileNo
Get #1, 4, dato
dato = 7
Put #1, 4, dato
Close #1
```

En el anterior ejemplo se puede observar como primero introducimos en la variable ***dato*** el valor del cuarto byte del fichero ***filename***, para posteriormente asignarle el valor 7, e introducirlo de nuevo en el cuarto byte de ***filename***.

8. ANEXO A: CONSIDERACIONES ADICIONALES SOBRE DATOS Y VARIABLES

En este ***Anexo*** se incluyen algunas consideraciones de interés para personas que no han programado antes en otros lenguajes. A continuación se explican las posibilidades y la forma de almacenar los distintos tipos de variables.

8.1 CARACTERES Y CÓDIGO ASCII

Las variables ***string*** (cadenas de caracteres) contienen un conjunto de caracteres que se almacenan en ***bytes*** de memoria. Cada carácter es almacenado en un ***byte*** (8 bits). En un bit se pueden almacenar dos valores (0 y 1); con dos bits se pueden almacenar $2_2 = 4$ valores (00, 01, 10, 11 en binario; 0, 1 2, 3 en decimal). Con 8 bits se podrán almacenar $2_8 = 256$ valores diferentes (normalmente entre 0 y 255; con ciertos compiladores entre -128 y 127).

En realidad, cada ***letra*** se guarda en un solo ***byte*** como un número entero, el correspondiente a esa letra en el código ASCII (una correspondencia entre números enteros y caracteres, ampliamente utilizada en informática), que se muestra en la Tabla 8.1 para los caracteres estándar (existe un código ASCII extendido que utiliza los 256 valores y que contiene caracteres especiales y caracteres específicos de los alfabetos de diversos países, como por ejemplo las *vocales acentuadas* y la *letra ñ* para el castellano).

	0	1	2	3	4	5	6	7	8	9
0	nul	soh	stx	etx	eot	enq	ack	bel	bs	ht
1	nl	vt	np	cr	so	si	dle	dc1	dc2	dc3
2	dc4	nak	syn	etb	can	em	sub	esc	fs	gs
3	rs	us	sp	!	"	#	$	%	&	'
4	(	)	*	+	,	-	.	/	0	1
5	2	3	4	5	6	7	8	9	:	;
6	<	=	>	?	@	A	B	C	D	E
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z	[	\	]	^	_	`	a	b	c
10	d	e	f	g	h	i	j	k	l	m
11	n	o	p	q	r	s	t	u	v	w
12	x	y	z	{	|	}	~	del		

Tabla 8.1. Código ASCII estándar.

Esta tabla se utiliza de la siguiente forma. La primera cifra (las dos primeras cifras, en el caso de los números mayores o iguales que 100) del número ASCII correspondiente a un carácter determinado figura en la primera columna de la Tabla 8.1, y la última cifra en la primera fila de dicha Tabla. Sabiendo la fila y la columna en la que está un determinado carácter puede componerse el número ASCII correspondiente.

Por ejemplo, la letra A está en la fila 6 y la columna 5. Su número ASCII es por tanto el 65. El carácter % está en la fila 3 y la columna 7, por lo que su representación ASCII será el 37. Obsérvese que el código ASCII asocia números consecutivos con las letras

mayúsculas y minúsculas ordenadas alfabéticamente. Esto simplifica notablemente ciertas operaciones de ordenación alfabética de nombres. Nótese que todas las mayúsculas tienen código ASCII anterior a cualquier minúscula.

En la Tabla 8.1 aparecen muchos caracteres no imprimibles (todos aquellos que se expresan con 2 ó 3 letras). Por ejemplo, el **ht** es el *tabulador horizontal*, el **nl** es el carácter *nueva línea*, etc.

En realidad la ***Versión 6.0*** de ***Visual Basic*** no utiliza para representar caracteres el código ASCII sino el ***Unicode***, que utiliza dos bytes por carácter, con lo cual tiene capacidad para representar 65536 caracteres, pudiéndose adaptar a las lenguas orientales. Para los caracteres latinos el byte más significativo coincide con el del código ASCII, mientras que el segundo byte está a cero.

8.2 NÚMEROS ENTEROS

Los números enteros en ***Visual Basic 6.0*** se guardan en 1, 2 ó 4 bytes.

- Con 8 bits (1 byte) se podrían guardar 2_8 números: desde 0 hasta 255.

- Con 16 bits (2 bytes) se podrían guardar 2_{16} números: desde 0 hasta 65.535.
  Si se reserva un bit para el signo se tendrían 2_{15} números: desde **-32768** hasta **32767**

- Con 32 bits (4 bytes) se podrían guardar 2_{32} números: desde 0 hasta 4.294.967.295.
  Si se reserva un bit para el signo se tendrían 2_{31}: desde **-2.147.483.648** hasta **2.147.483.647**

8.3 NÚMEROS REALES

En muchas aplicaciones hacen falta variables reales, capaces de representar magnitudes que contengan *una parte entera y una parte fraccionaria* o *decimal*. Estas variables se llaman también de ***punto flotante***. De ordinario, en base 10 y con notación científica, estas variables se representan por medio de la ***mantisa***, que es un número mayor o igual que 0.1 y menor que 1.0, y un ***exponente*** que representa la potencia de 10 por la que hay que multiplicar la mantisa para obtener el número considerado. Por ejemplo, el número π se representa como $0.3141592654 \cdot 10_1$. Tanto la ***mantisa*** como el ***exponente*** pueden ser positivos y negativos.

8.3.1 Variables tipo Single

Los computadores trabajan en base 2. Por eso un número con decimales de tipo ***Single*** se almacena en 4 bytes (32 bits), utilizando *24 bits para la mantisa* (1 para el signo y 23 para el valor) y *8 bits para el exponente* (1 para el signo y 7 para el valor).
Es interesante ver qué clase de números de punto flotante pueden representarse de esta forma. En este caso hay que distinguir el ***rango*** de la ***precisión***. La *precisión* hace referencia al número de cifras con las que se representa la *mantisa*: con 23 bits el número más grande que se puede representar es,

$$2^{23} = 8.388.608$$

lo cual quiere decir que se pueden representar todos los números decimales de 6 cifras y la mayor parte – aunque no todos – de los de 7 cifras (por ejemplo, el número 9.213.456 no se puede representar con 23 bits). Por eso se dice que las variables tipo ***Single*** tienen entre 6 y 7 cifras decimales equivalentes de precisión.

Respecto al *exponente de dos* por el que hay que multiplicar la *mantisa* en base 2, con 7 bits el número más grande que se puede representar es 127. El ***rango*** vendrá definido por la potencia,

$$2^{127} = 1.7014 \cdot 10^{38}$$

lo cual indica el orden de magnitud del número más grande representable de esta forma.

En el caso de los números ***Single*** (4 bytes) el rango es: desde **-3.402823E38** a **-1.401298E-45** para los valores negativos y desde **1.401298E-45** a **3.402823E38** para los valores positivos.

8.3.2 Variables tipo Double

Así pues, las variables tipo ***Single*** tienen un *rango* y –sobre todo– una *precisión* muy limitada, insuficiente para la mayor parte de los cálculos técnicos y científicos. Este problema se soluciona con el tipo ***Double***, que utiliza 8 bytes (64 bits) para almacenar una variable. Se utilizan *53 bits para la mantisa* (1 para el signo y 52 para el valor) y *11 para el exponente* (1 para el signo y 10 para el valor). La ***precisión*** es en este caso,

$$2^{52} = 4.503.599.627.370.496$$

lo cual representa entre 15 y 16 cifras decimales equivalentes. Con respecto al ***rango***, con un exponente de 10 bits el número más grande que se puede representar será del orden de 2 elevado a 2 elevado a 10 (que es 1024):

$$2^{1024} = 1.7977 \cdot 10^{308}$$

Si se considera el caso de los números declarados como ***Double*** (8 bytes) el rango va desde **-1.79769313486232E308** a **-4.94065645841247E-324** para los valores negativos y desde **4.94065645841247E-324** a **1.79769313486232E308** para los valores positivos.

8.4 SISTEMA BINARIO, OCTAL, DECIMAL Y HEXADECIMAL

A contnuación se presentan los primeros números naturales expresados en distintos sistemas de numeración (bases 10, 2, 8 y 16). Para expresar los números en base 16 se utilizan las seis primeras letras del abecedario (A, B, C, D, E y F) para representar los números decimales 10, 11, 12, 13, 14 y 15. La calculadora que se incluye como accesorio en ***Windows*** ofrece posibilidades de expresar un número en distintos sistemas de numeración.

Decimal	Binario	Octal	Hexadecimal
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10
17	10001	21	11

Tabla 3.4 Expresión de un número en los distintos sistemas.

Fin del curso...

Continuará.....

Nota de Radacción: El lector puede descargar este capítulo y capítulos anteriores del curso desde la sección ***"Artículos Técnicos"*** en el sitio web de **EduDevices (www.edudevices.com.ar)**