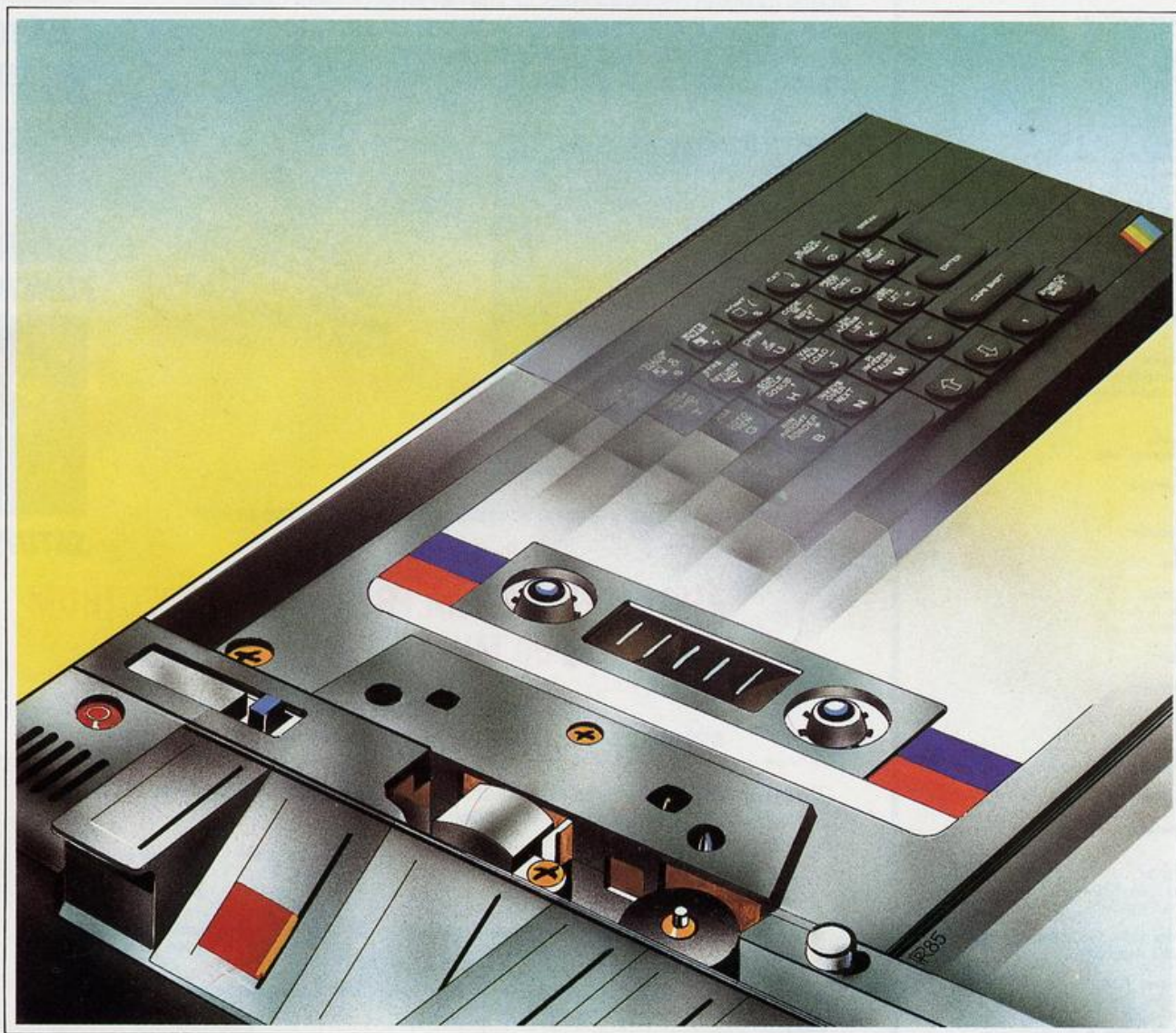


18  
150pts.

# PULN

## Enciclopedia Práctica del Spectrum



Nueva Lente/Ingelek







# LETRAS QUE SON NUMEROS

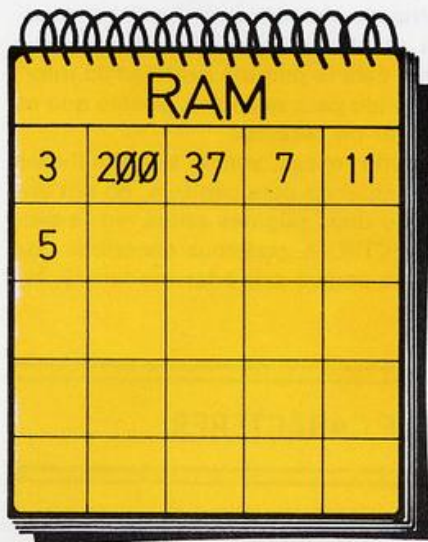
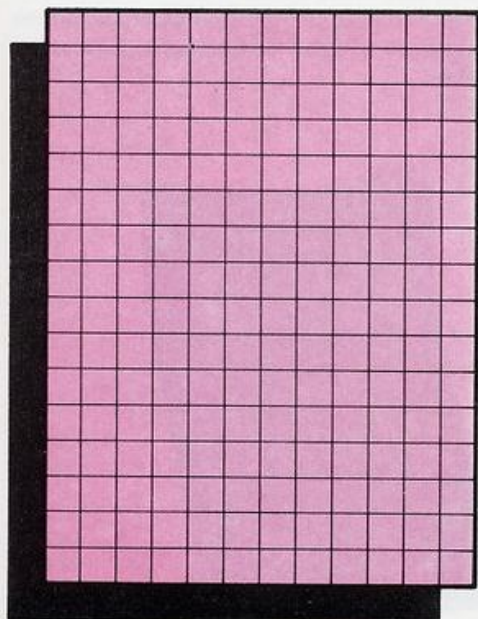


N capítulos anteriores hemos visto, de forma general, cómo el ordenador es capaz de tratar diferentes tipos de datos para realizar con ellos operaciones complicadas de cálculo, segmentación o comparación. Lo que no hemos hecho, en ningún momento, ha sido penetrar en la estructura interna de los datos, es decir, describir la forma en que éstos se encuentran almacenados dentro del ordenador.

Simplificando mucho las cosas, podemos suponer que la memoria del ordenador es como una gigantesca hoja de papel cuadriculado, en la cual podemos borrar y escribir tantas veces como deseemos. Aún más, podemos elegir exactamente el lugar donde deseamos escribir, indicando el número correspondiente a una casilla determinada, siendo realmente ésta nuestra única restricción, puesto que nunca podremos escribir «a caballo» entre dos casillas.

Al conjunto de esta memoria se le denomina **RAM** (*Random Access Memory*), cuya traducción es Memoria de Acceso Aleatorio. Con ello podremos decir que tenemos acceso a cualquier parte de ella, ya sea para leer su contenido o para

*Las memorias pueden considerarse como un papel cuadriculado, en el cual se encuentra la información.*



*En las memorias denominadas RAM, se admite tanto la lectura como la escritura de información.*

alterarlo, escribiendo uno nuevo, con sólo reseñar su posición concreta dentro de la misma. Con una estructura similar a la descrita, el fabricante ha incluido también dentro de nuestro ordenador otra hoja de papel cuadriculado, pero ingeniándoselas de forma que podemos leer su contenido, pero no alterarlo. De hecho, podemos tratar de «escribir» sobre ella, siendo inútil nuestro esfuerzo por destruirla. A esta memoria de sólo lectura se le denomina **ROM** (*Read Only Memory*).

En total, el Spectrum dispone de 65536 posiciones de memoria o cuadrículas, pertenecientes a uno u otro tipo de las memorias descritas; y en cada una de ellas puede almacenarse un carácter alfabético, numérico o cualquier símbolo. A cada una de estas posiciones descritas, se le denomina **BYTE** u **OCTETO**. Realmente, el nombre de **OCTETO** es el más descriptivo para nosotros: indica que cada una de las posiciones de memoria está compuesta a su vez por un conjunto de ocho unidades elementales, denominadas **BITS**.

Un **BIT** no es más que un impulso elemental binario, el cual puede adoptar dos únicos valores, a los cuales solemos asignar un significado lógico simbólico: **SI** o **NO**, **VERDADERO** o **FALSO**, y **1** ó **0**. Aunque esta estructura puede parecernos en principio complicada de comprender, diremos

## i!

El código A.S.C.I.I. está compuesto por 128 caracteres, numerados del 0 al 127.

## \*

Para la definición de funciones del usuario, utilizamos **DEF FN**, seguido del nombre de la función y los parámetros (en caso de que existan), entre paréntesis y separados por comas.



i!

El número total de caracteres son 256, puesto que corresponde con la máxima cantidad de información almacenable en un byte.

\*

Al tipo de memoria que permite tanto la lectura como la alteración de contenido, se le denomina R.A.M. (Random Access Memory).

\*

El Spectrum se ajusta casi totalmente al código A.S.C.I.I.

Al contrario que la RAM, la memoria ROM sólo admite operaciones de lectura.

que es, sin embargo, la única aceptable internamente por el ordenador.

En cualquier caso, no debe preocuparnos la estructura interna de almacenamiento del ordenador más que al nivel de posición de memoria (BYTE), por ser ésta la mínima cantidad de información con sentido para nosotros, puesto que es capaz de albergar un carácter.

Tras este pequeño repaso, vamos a zambullirnos en el tema principal de este capítulo, no sin antes recordar que unas páginas antes, en la sección de TU SPECTRUM, podemos encontrar una información más amplia sobre las memorias, los bits y los bytes.

## EL JUEGO DE CARACTERES

Es necesario que refresquemos ahora algunos conocimientos básicos sobre la aritmética binaria. Para empezar diremos que, del mismo modo que un dígito en base 10 (decimal) puede tener un contenido de 0 a 9, en base 8 (octal) puede albergar un número de 0 a 7, y en base 2 (binaria) únicamente los dígitos 0 y 1. Esto equivale a decir que, en cualquier base de numeración, los dígitos disponibles van desde el 0 hasta la base menos 1.

Si en base binaria sólo disponemos del 0 y el 1, con dos dígitos podemos albergar un número no superior a 3, con tres uno no superior a 7, con cuatro uno no superior a 15..., y con 8 un número que se encuentre en el rango de 0 a 255. Generalizando, podemos decir que siendo  $N$  el número de dígitos y  $B$  la base, se cumple la fórmula:  $X = B^N - 1$ . Donde  $X$  indica el máximo número que podemos almacenar.

Particularizando esta fórmula para  $B=2$  y para  $N=8$ , obtenemos el resultado de  $X=255$ . Este es precisamente el caso que nos ocupa, puesto que en un BYTE u OCTETO de memoria, podemos almacenar un número comprendido entre 0 y 255. Esta cantidad (256 posibilidades) corresponde con las dimensiones del juego de caracteres del

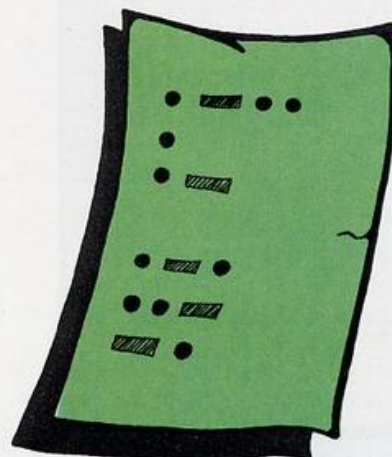
El sistema Morse sigue un tipo de codificación basado en los impulsos eléctricos.

ROM						
7	20	2	400	5		
32	70	1				
61	9					

Spectrum, es decir, el conjunto de todas las letras, números, símbolos y palabras clave que encontramos en el teclado, e incluso algunos más, aunque no representables en la pantalla.

## CODIFICACION DE CARACTERES

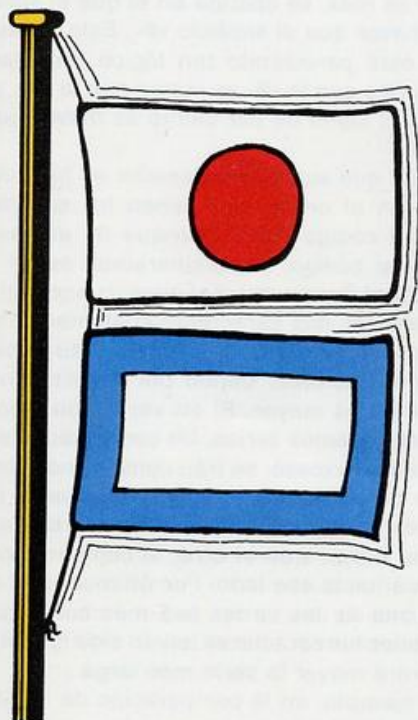
Como ya sabemos, al ordenador le es mucho más fácil llevar a cabo el tratamiento de números que de letras, y lo que es más, en realidad sólo puede efectuar un tratamiento numérico muy restringido, a base de unos y ceros. Sin embargo, a través de determinados sistemas de codificación, es capaz de llevar a cabo complejas operaciones matemáticas.





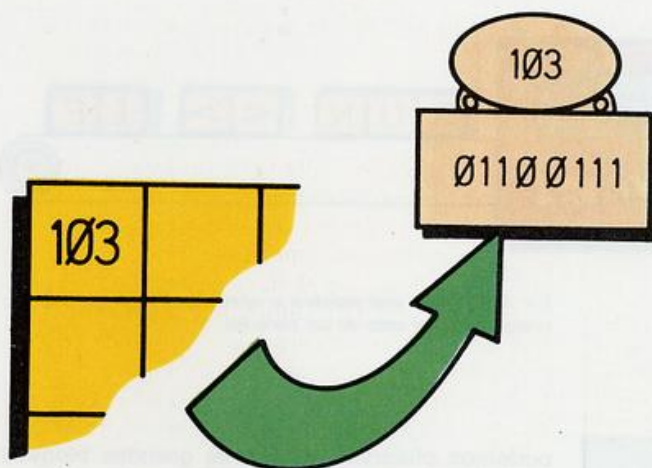


Del mismo modo, el ordenador codifica los caracteres en forma de números, lo cual le permite dar el tratamiento adecuado a los caracteres, ya sean letras, símbolos, etc. Por otra parte, este proceso de codificación no es más que una adecuación al medio, que nos es bastante familiar. Así, cuando Samuel Morse inventó su conocido código telegráfico, lo hizo obligado por la necesidad de sólo transmitir impulsos eléctricos de dos longitudes diferentes (puntos y rayas). Otra codificación similar, esta vez visual, es la que siguen las banderas de señalización en los barcos. Así pues, el hecho de codificar las letras por otro sistema, fácil de manejar por las herramientas de que se dispone, no es precisamente una de las originalidades del ordenador. En todo caso, vamos a estudiar la codificación que lleva a cabo nuestro Spectrum.



Las banderas de señalización en los barcos son un medio de codificación visual de información.

Desde la aparición de los ordenadores, se hizo necesaria la creación de un código estandarizado, para permitir la comunicación entre los diversos aparatos. De esta idea surgió el estándar de más frecuente utilización: el A.S.C.I.I. Estas siglas corresponden a *American Standard Code for Information Interchange*, que podríamos traducir como CODIGO NORMALIZADO AMERICANO PARA EL INTERCAMBIO DE INFORMACION; el mencionado código está compuesto por 128 caracteres, numerados del 0 al 127, que se detallan en el cuadro adjunto.

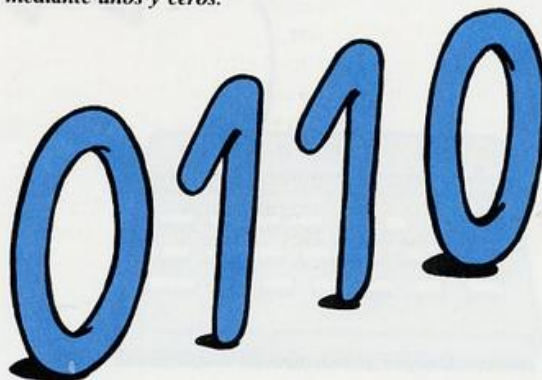


*Aunque cada una de las posiciones de memoria contiene realmente 8 bits, a nosotros únicamente nos interesa el significado del byte completo.*

Como podemos ver, este incluye tanto letras mayúsculas como minúsculas, símbolos, signos de puntuación y caracteres de control; en definitiva, todos aquellos caracteres que pueden ser necesarios para cualquier comunicación. Sin embargo, aún quedan libres ciento veintiocho códigos (del 128 al 255). ¿Qué es lo que sucede con ellos? Simplemente son particulares de cada aparato, y se escapan al código estandarizado A.S.C.I.I. Así pues, cada fabricante de ordenadores los emplea para distintas finalidades; generalmente, contienen los gráficos y símbolos especiales. En el caso concreto del Spectrum, esta zona está reservada a los gráficos predefinidos y definibles, y a los **TOKEN**, es decir, las palabras BASIC que se componen de más de una letra.

El estudio del código A.S.C.I.I. tiene un cierto interés, puesto que como hemos dicho, es empleado por una gran cantidad de ordenadores y periféricos muy importantes, como por ejemplo, la mayoría de las impresoras. Fundamentalmente,

*Todas las informaciones son codificadas en el ordenador mediante unos y ceros.*



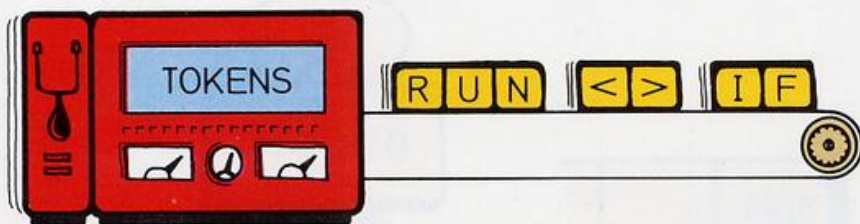
**i!**

Los códigos del juego de caracteres que comprenden desde el 128 hasta el 255, se emplean para los gráficos y **TOKENS**.

**\***

Un **TOKEN** es una palabra o símbolo BASIC completo, independientemente del número de caracteres individuales que lo compongan.





Un **TOKEN** es una palabra u operador BASIC compuesto por más de un carácter.

## i!

Para la comparación de caracteres, el ordenador se basa en la posición de los mismos dentro del código A.S.C.I.I.

## \*

La comparación de cadenas se efectúa comparando los caracteres que ocupan posiciones iguales en las cadenas a tratar.

## \*

La función de conversión de un carácter a su código es **CODE**.

podemos observar en él tres grandes bloques: El primero de ellos, se extiende desde el 0 hasta el 31, y está constituido por caracteres de control. En el caso concreto del Spectrum, sólo se utilizan los códigos 6 al 23, y de todas formas no se atienen totalmente al estándar A.S.C.I.I. A este bloque siguen una serie de símbolos y signos de puntuación, que comienzan por el código 32 (espacio), y finalizan en el 64, conocido generalmente por arroba (@). Dentro de este grupo, es importante destacar la presencia de los dígitos, que se extienden desde el 48 (el cero), hasta el 57 (el nueve).

El siguiente bloque importante comienza en el código 65 y llega hasta el 90, conteniendo todo el alfabeto inglés en mayúsculas. Una pequeña serie de tan solo seis caracteres, cuyos códigos van del 91 al 96, separan este bloque del último de importancia.

El último bloque a reseñar comprende los caracteres del 97 al 122, y representan de nuevo el alfabeto inglés, pero esta vez en minúsculas, seguido de los últimos caracteres especiales, los

*El código más común para la comunicación entre ordenadores e impresoras es el A.S.C.I.I.*

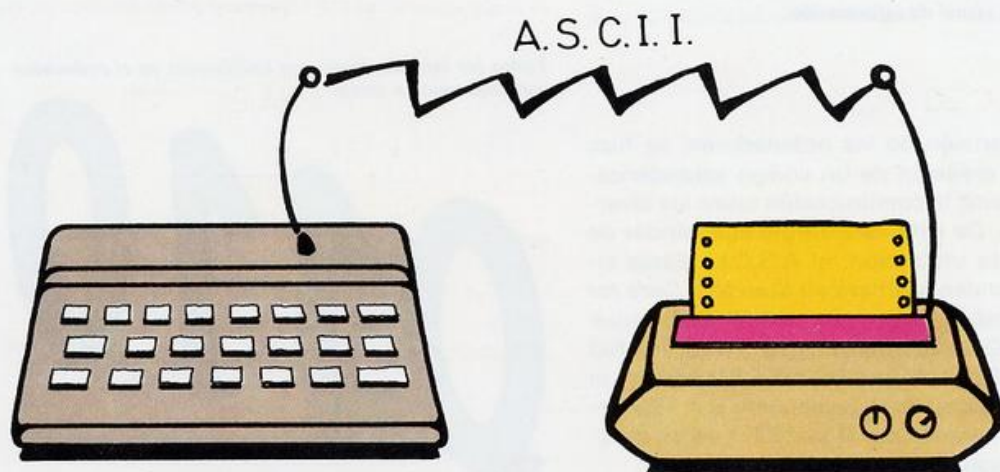
cuales abarcan desde el código 123 hasta el 127. El juego de caracteres del ZX Spectrum, no respeta completamente la norma A.S.C.I.I. aunque sí en lo referente a los tres bloques fundamentales que hemos mencionado.

El hecho de que el ordenador utilice un código u otro, puede ser que nos parezca en estos momentos carente de importancia, pues se trata de un sistema de régimen interno que en teoría no nos afecta como programadores. No obstante, éste es un punto vital en el desarrollo de gran cantidad de programas, e inmediatamente conoceremos el porqué.

Cuando el ordenador efectúa comparaciones numéricas, su lógica resulta aplastante: "4 es mayor que 1"; sin embargo, no se inmuta en absoluto al realizar comparaciones entre letras, y se muestra bien seguro de que la «B» es mayor que la «A», y lo que es más, se obstina en el que el símbolo «%» es mayor que el símbolo «\$». Esto último ya no nos está pareciendo tan lógico. Aceptamos que suponga que la «B» es mayor que la «A», pero ¿por qué el signo de por ciento es mayor que el de dólar?

El sistema que sigue el ordenador es bien claro, se basa en el orden que tienen los caracteres dentro del código ASCII (aunque no es exactamente este código, denominaremos así al empleado por el Spectrum). Así pues, cuando deseamos comparar dos caracteres cualquiera, nuestro Spectrum se limita a averiguar su posición dentro del A.S.C.I.I., dando por mayor a aquel cuyo código es mayor. Si en vez de un solo carácter comparamos series, las complicaciones no aumentan en exceso: se irán comparando los caracteres de las series que ocupen iguales posiciones; en el momento en que uno de los caracteres sea mayor que el otro, la comparación se decantará hacia ese lado. Por último, en el caso de que una de las series sea más corta que la otra, y todos los caracteres hayan sido iguales, se considerará mayor la serie más larga.

Así por ejemplo, en la comparación de "AHG" y "AHGA", se considerará mayor la segunda par-







te, puesto que los primeros, segundos y terceros caracteres son iguales entre sí, pero sin embargo, la segunda cadena es más larga que la primera. Si la comparación se puede resolver antes del final de cualquiera de las cadenas, la longitud no se tiene en cuenta para nada; un ejemplo claro de ello, es el hecho de que la cadena «B» sea mayor que la «AAAAAAA».

La comparación de cadenas se utiliza en gran cantidad de rutinas, como por ejemplo aquellas que manejan series, depuran entradas, etc... así pues, para atender las necesidades de programación sobre un tema que se ha revelado de vital importancia, el BASIC dispone de dos funciones de conversión directa, que relacionan al programador con el código que el ordenador emplea para sus caracteres. Estas dos funciones se denominan **CHR\$** y **CODE**.

## CHR\$ Y CODE

La función **CHR\$** tiene como argumento un número comprendido entre 0 y 255, proporcionando una cadena de salida con el carácter cuyo código se especifica en el argumento. El siguiente programa obtiene el juego completo de caracteres del Spectrum:

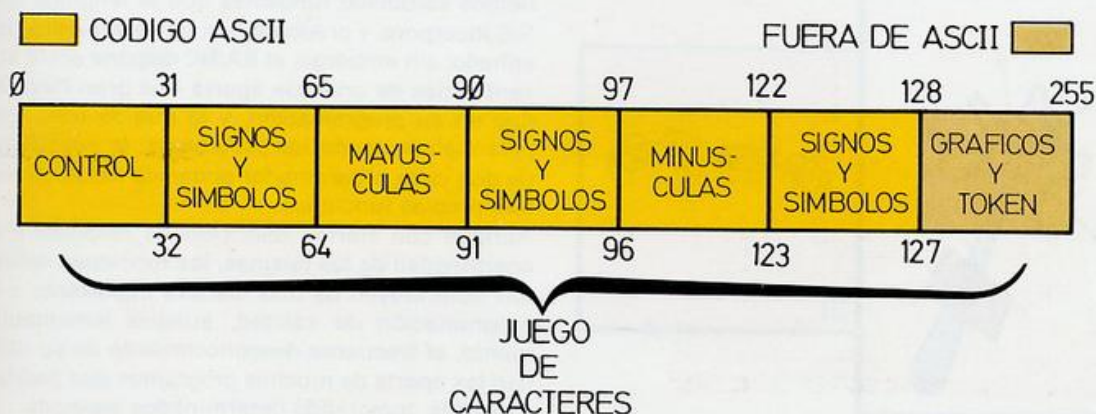
```
10 REM - JUEGO DE CARACTERES
20 PRINT "-----> CODIGOS ASCII
<-----"
30 FOR I=32 TO 127:PRINT CHR$ I;:NEXT I
40 PRINT "-----> CODIGOS RESTANTES
<-----"
50 FOR I=128 TO 255:PRINT CHR$ I;:NEXT I
```

## CODIGOS DE CONTROL

ABREVIATURA	SIGNIFICADO
NUL	Nulo.
SOH	Principio de encabezamiento.
STX	Comienzo de texto.
ETX	Fin de texto.
EOT	Fin de transmisión.
ENQ	Pregunta.
ACK	Acuse de recibo.
BEL	Timbre (señal).
BS	Retroceso.
HT	Tabulación horizontal.
LF	Cambio de renglón.
VT	Tabulación vertical.
FF	Página siguiente.
CR	Retroceso de carro.
SO	Fuera de código.
SI	En código.
SP	Espacio.
DLE	Encaje de transmisión.
ESC	Escape.
DEL	Supresión.
DC1	Mando de dispositivo auxiliar 1.
DC2	Mando de dispositivo auxiliar 2.
DC3	Mando de dispositivo auxiliar 3.
DC4	Mando de dispositivo auxiliar 4.
NAK	Acuse de recibo negativo.
SYN	Sincronización de reposo.
ETB	Fin de bloque de transmisión.
CAN	Cancelación.
EM	Fin de medio físico.
SUB	Sustitución.
FS	Separador de fichero.
GS	Separador de grupo.
RS	Separador de registro.
US	Separador de unidad.

Al margen de su utilidad como función de codificación del juego de caracteres, **CHR\$** tiene un gran interés en otro aspecto: la posibilidad de tra-

*El juego de caracteres del Spectrum puede ser dividido en 7 zonas.*



**i!**

En el momento en que un carácter de una cadena es superior al correspondiente de la otra, la primera se considera mayor que la segunda.

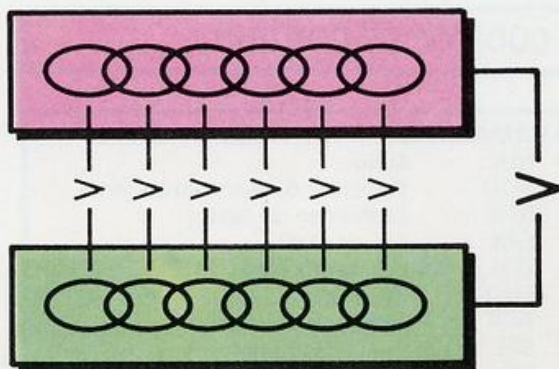
**\***

Si no existe diferencia entre los primeros caracteres de dos cadenas, se considera mayor la más larga.

**\***

La función de conversión de un código en carácter es **CHR\$**.





Las comparaciones entre cadenas, se efectúan entre los caracteres individuales que las componen, y que ocupan idénticas posiciones en cada una de ellas.

tar información de cadena como si fuese numérica, lo cual nos deja el camino abierto a las estructuras de bucle.

Supongamos que deseamos representar en columna el alfabeto inglés (mayúsculas). Nosotros reduciremos mentalmente esta tarea a una operación muy sencilla: «Para un carácter variando por orden desde la "A" hasta la "Z", escribamos cada vez el que corresponda».

Como ya sabemos, la traducción de este tipo de estructuras cíclicas al BASIC, se lleva a cabo de una forma prácticamente directa, mediante las sentencias **FOR-NEXT**. Así pues, estaremos tentados de programar la siguiente instrucción: **FOR I\$="A" TO "Z": PRINT I\$: NEXT I\$**. Sin embargo, esta instrucción sería inmediatamente rechazada por el comprobador de sintaxis, puesto que no podemos utilizar variables de cadena como índices de ciclo.

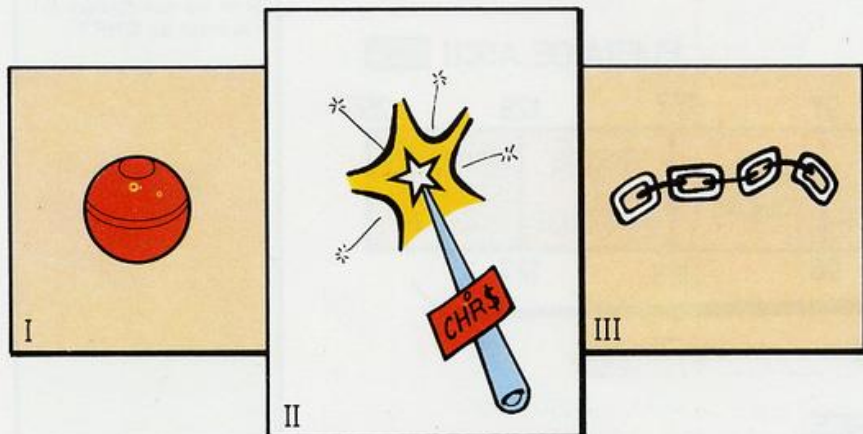
La solución a nuestro problema consiste en en-

contrar una relación numérica entre los caracteres a representar. Esta bien puede ser el código que les corresponde dentro del juego de caracteres, y por tanto, será la función **CHR\$** la que vendrá a sacarnos del apuro. La rutina anteriormente expuesta podría codificarse de la siguiente manera: **FOR I=65 TO 90: PRINT CHR\$ I: NEXT I**. Esta instrucción se puede traducir al Castellano, diciendo: «para los códigos variando del 65 al 90, escribe el carácter cuyo código corresponde». La función **CODE** es inversa a **CHR\$**: a partir de un argumento de cadena, produce el código numérico del carácter correspondiente. Más estrictamente diremos que obtiene el código del primer carácter de una cadena, devolviendo el valor 0 si la cadena es vacía. Así, obtendríamos el mismo resultado (65) en los dos supuestos siguientes: **PRINT CODE "A"** y **PRINT CODE "ABC"**. También podemos comprobar que las funciones **CHR\$** y **CODE** son inversas, escribiendo lo siguiente: **PRINT "A" y PRINT CHR\$ CODE "A"**.



Mediante **DEF FN** los usuarios podemos definir nuestras propias funciones.

La función **CHR\$** nos permite obtener el carácter que corresponde a determinado código.



El hecho de que **CHR\$** opere en base a argumentos numéricos, no sólo reporta el beneficio de poder ser empleado en estructuras cíclicas, sustituyendo a los caracteres, sino que también le facilita el camino hacia las funciones definidas por el usuario. Efectivamente, hasta ahora, hemos estudiado funciones que el lenguaje BASIC incorpora, y proporcionan un determinado resultado; sin embargo, el BASIC dispone entre sus sentencias de una que aporta una gran flexibilidad en su programación, y lo que es más, una personalización de los programas: la posibilidad de que cada programador podamos definir nuestras propias funciones.

Aunque con ciertas restricciones respecto a la operatividad de las mismas, las funciones definidas contribuyen de una manera importante a la programación de calidad, aunque lamentablemente, el frecuente desconocimiento de su utilidad las aparta de muchos programas que podrían incluirlas, mejorando determinados aspectos.





## DEFINICION DE FUNCIONES

La definición de funciones propiamente dicha, se canaliza a través de la sentencia **DEF FN** (*DEFine FuNction*) seguida del nombre de la función y los parámetros necesarios, entre paréntesis, y separados por comas si existe más de uno. De comienzo parece un poco lioso, pero pronto veremos que la realidad es otra; estudiemos separadamente cada uno de los componentes expuestos.

Los nombres de función deben constar de una sola letra, seguida o no del símbolo dólar (\$), dependiendo del tipo de resultado que esperemos obtener de ella. De este modo, son funciones de cadena aquellas que terminan en dólar (\$), y numéricas las demás.

Los parámetros pueden ser ninguno, uno, o más de uno, en cuyo caso deberán ir separados por comas, y siempre encerrados entre paréntesis a continuación del nombre de la función; incluso si no existen parámetros deberán indicarse los paréntesis, aunque sin ningún contenido. Estos parámetros representan el argumento de la función definida y, por tanto, el resultado de ésta dependerá directamente del valor de los mismos.

Los parámetros pueden ser de cualquier tipo (numéricos o de cadena), y no existe ningún problema para combinarlos en una misma función. La única restricción estriba en que los parámetros de cadena deben ser nombres válidos de variable de este tipo —una letra seguida del símbolo dólar (\$)—, y los numéricos, nombres de una sola letra. Como en el caso de las variables, es indiferente que el parámetro se escriba en mayúsculas o minúsculas; de esto se desprende que el máximo número de parámetros de un tipo que puede incluir una función son veintiséis (las letras del alfabeto), y en total cincuenta y dos (veintiséis de un tipo y veintiséis del otro).

Vamos a definir una función muy simple. La llamaremos **P**, y llevará a cabo el producto de dos números; así pues, la línea de definición será: **1 DEF FN P(X,Y)=X\*Y**. Donde **X** e **Y** son los parámetros. Si nos damos cuenta, hemos antepuesto un número de línea a la definición, esto se debe a que esta sentencia opera de forma similar a **DATA**, y no puede ser empleada en el modo directo.

La forma en que ejecutamos una función definida es mediante la función BASIC **FN** (*FuNction*), la cual lógicamente tiene una sintaxis muy similar a la definición: debe ir seguida del nombre de la función a ejecutar, y de los parámetros entre

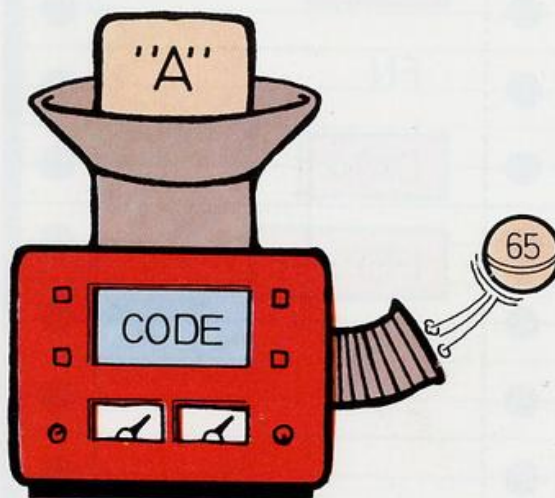
## EL CODIGO ASCII

BITS 3210	BITS 654							
	000	001	010	011	100	101	110	111
0000	NUL	DLE	SP	0	@	P	£	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	\$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	/	l	
1101	CR	GS	-	=	M	]	m	}
1110	SO	RS	.	>	N	_	n	~
1111	SI	US	/	?	O	—	o	DEL

paréntesis. Como cualquier función BASIC, precisa de una palabra clave anterior que indique el destino del resultado. Por ejemplo, vamos a escribir el valor de **P** creada anteriormente, para **X=5** e **Y=7**. La instrucción será en este caso **PRINT FN P(5,7)**.

El resultado obtenido será 35, pero no es esto lo importante, sino que no hemos utilizado ninguna sentencia de asignación (**LET**, **INPUT** o **READ**), para otorgar valores a **X** e **Y**. Esto es de considerable importancia, puesto que de ello se desprende que no existe ninguna relación entre los parámetros de una función y unas posibles variables con el mismo nombre.

Esto no quiere decir que no se pueden utilizar variables dentro de la definición de una función, sino que, cuando se ha utilizado un determinado nombre como parámetro, el BASIC no considera



CODE es la función BASIC que nos permite averiguar el código que corresponde a determinado carácter.



i!

Al tipo de memoria que sólo admite la lectura, pero no la alteración de contenido, se le denomina R.O.M. (Read Only Memory).

\*

Las siglas A.S.C.I.I. corresponden a *American Standard Code for Information Interchange*, cuya traducción es Código Normalizado Americano para el Intercambio de Información.

\*

En el código A.S.C.I.I. podemos diferenciar tres bloques principales: caracteres de control, alfabeto en mayúsculas y alfabeto en minúsculas.

PARAMETROS

DEF FN

RESULTADO

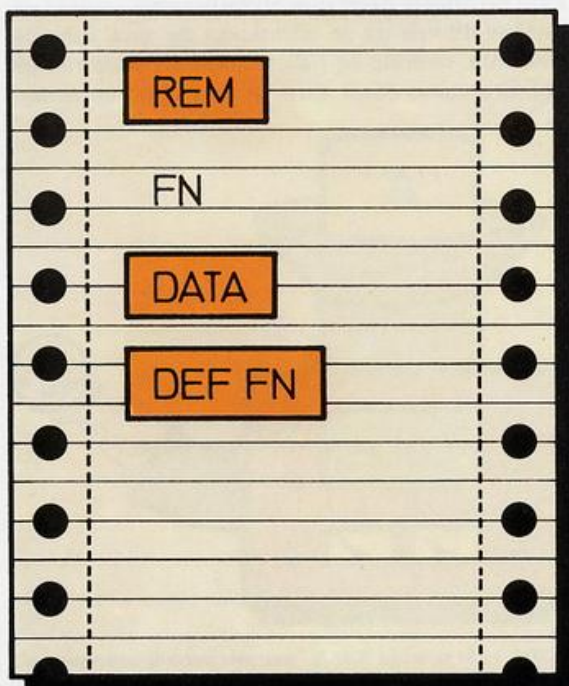
El resultado de una función definida depende de determinados parámetros, y es obtenido mediante la sentencia FN.

la existencia de ninguna variable con ese mismo nombre. De no ser así, es decir, en caso de que una variable, cuyo nombre no se emplea en los parámetros, sí tome parte en la definición de la función, su valor se obtendrá del que tuviera dicha variable en el momento de la ejecución de FN. Veamos un ejemplo:

```
10 DEF FN P(X,Y)=X*Y+A
20 LET A=15
30 PRINT FN P(2,3)
```

El resultado obtenido por esta función será 21 ( $2*3+15=21$ ), puesto que los valores de X e Y se tomarán de los parámetros, y el de A, al no encontrarse en la definición dentro de los mismos, será extraído del área de variables. Por supuesto, no es necesario utilizar constantes al evaluar una

Las sentencias DEF FN, al igual que DATA y REM, son ignoradas por el ordenador durante la interpretación de la línea en que se encuentran.



función definida (FN), sino que podemos utilizar variables o expresiones. La única restricción es que figuren exactamente los parámetros precisos; en caso contrario, obtendremos el error **Q Parameter error** (Q Error de parámetro).

La utilidad de DEF FN es bastante grande, puesto que nos permite definir algunas funciones interesantes, presentes en otros BASIC, pero de los que carece el de Sinclair. Así por ejemplo, la función **MOD**, da como resultado el resto de la división entre dos números, y puede ser simulada mediante: **DEF FN M(A,B)=A-B\*INT (A/B)**.



FN es la función BASIC que se emplea para acceder a los resultados de una función definida.

En el siguiente ejemplo, mostramos la función que permite tomar de la cadena X\$ los X caracteres de su derecha; esta función se denomina en otros BASIC **RIGHT\$**: **DEF FN R\$(A\$,A)=A\$(LEN A\$-A+1 TO)**. Como último ejemplo, podemos definir la función de redondeo a cualquier número de cifras, donde X es el número a redondear, e Y la cantidad de decimales requeridos: **DEF FN A(X,Y)=INT ((X+(5/10 ↑ (Y+1)))\* 10 ↑ Y)/10 ↑ Y**.

En definitiva, DEF FN es una herramienta de mucha utilidad, con la cual debemos practicar.





# MANEJO DE SERIES



AJO la denominación general de manejo de series, incluimos habitualmente tres actividades bien diferentes: la ordenación de series o *sort*, la

busqueda en las mismas, y su desordenación o *SHUFFLING*. Anteriormente, nos ocupamos de la primera de estas tareas, y seguramente la más importante de todas ellas, puesto que, como veremos más adelante, en algunas ocasiones, antes de efectuar una búsqueda es conveniente someter a la serie a un proceso previo de *sort*. Por otra parte, la aplicación del *shuffling*, método totalmente opuesto al *sort*, se da casi únicamente en determinados programas de juegos; por tanto, en un ámbito bastante más restringido que los *sort*, debido a lo cual, no le dedicaremos la misma atención que a su antónimo.

En primer lugar, nos ocuparemos del estudio de las técnicas de búsqueda.

## BUSQUEDA EN SERIES: SHUFFLINGS

El fin último de las técnicas de manejo de series es un acceso cómodo y rápido a la información que éstas contienen. En este sentido, debemos de procurar someter a las mismas a ciertas operaciones, encaminadas a la facilitación de la posterior labor de acceso. Estas operaciones suelen ser de clasificación, aunque como veremos más adelante existen determinadas ocasiones en que la serie de partida está ya ordenada y lo que se pretende es someterla a un *shuffling*, puesto que es conveniente que los valores a acceder se encuentren desordenados.

Una vez realizado el tratamiento de la serie, el acceso a cualquier elemento de ella depende del tipo de almacenamiento de la información escogido. En este caso, podemos hablar del acceso directo, o de la necesidad de una búsqueda. La primera circunstancia es la ideal, dado que aumenta al máximo la comodidad del acceso, y disminuye al mínimo el tiempo en conseguir el dato. En el acceso directo, utilizamos una simple he-

rramienta BASIC para recuperar la información contenida en determinado punto de un colectivo (serie). Pongamos un ejemplo bien sencillo: la serie a tratar será toda la memoria, y el elemento a acceder el 32000. En este caso, el problema será bien fácil de resolver; sólo necesitamos ejecutar **PRINT PEEK 32000**. Esto ha sido posible, sólo porque la referencia del elemento a buscar coincidía con el índice por el cual se accede a la serie. Veamos otro ejemplo algo más práctico. Esta vez, en un juego en el cual se lanza un dado, nos interesa saber si determinada puntuación ha aparecido ya. Para ello podemos utilizar un simple sistema de banderas, consistente en hacer alguna variable, fácilmente accesible, igual a un valor indicador de la aparición anterior o no del elemento al cual representa. En este sentido, podemos dimensionar un vector numérico de seis elementos, los cuales contendrán inicialmente el valor 0. Asimismo, adoptaremos el convenio de cambiar el elemento correspondiente al valor a uno (o a cualquier valor distinto de cero), para señalar que dicho valor ya ha sido extraído.

A ninguno se nos escapa que este tipo de acceso sólo es posible con determinadas estructuras de datos, y sobre todo, presenta el inconveniente de la gran cantidad de memoria utilizada. Para solapar un poco este inconveniente, podemos emplear vectores de cadena, que como ya sabemos ocupan menos memoria que los numéricos, pero esto no supondrá una solución definitiva.

Una vez que nos vemos obligados a realizar una búsqueda, podemos optar por dos sistemas: el de búsqueda secuencial, y el de extrapolación de in-

i!

Por *shufflings* entendemos el conjunto de técnicas que permiten, partiendo de una Serie ordenada, obtener una serie distribuida aleatoriamente.

\*

El método de búsqueda por extrapolación de intervalos supera en eficacia, por lo general, al secuencial, si bien es cierto que sólo puede ser aplicado sobre series ordenadas.



**SORTS**  
(ordenación)



**BUSQUEDA**



**SHUFFLINGS**  
(desordenación)

Bajo la denominación general de manejo de series, se incluyen fundamentalmente tres procesos: ordenación (*sort*), búsqueda, y desordenación (*shuffling*).



## i!

Fundamentalmente, existen dos métodos de búsqueda en series: secuencial y por extrapolación de intervalos.

La búsqueda secuencial funciona con series desordenadas, pero puede ser aumentada su eficacia si se aplica sobre listas ordenadas.

Intercambiar los valores de dos elementos, ya sean variables, componentes de una serie, etc... se denomina *swapping*.

Para efectuar correctamente un *swapping*, hemos de tener muy presente la necesidad de emplear una variable puente, en la cual poder almacenar temporalmente el valor de uno de los elementos.

Uno de los sistemas para eliminar un elemento de una serie, consiste en solaparlo mediante el desplazamiento de una posición, del resto de los componentes de la serie.

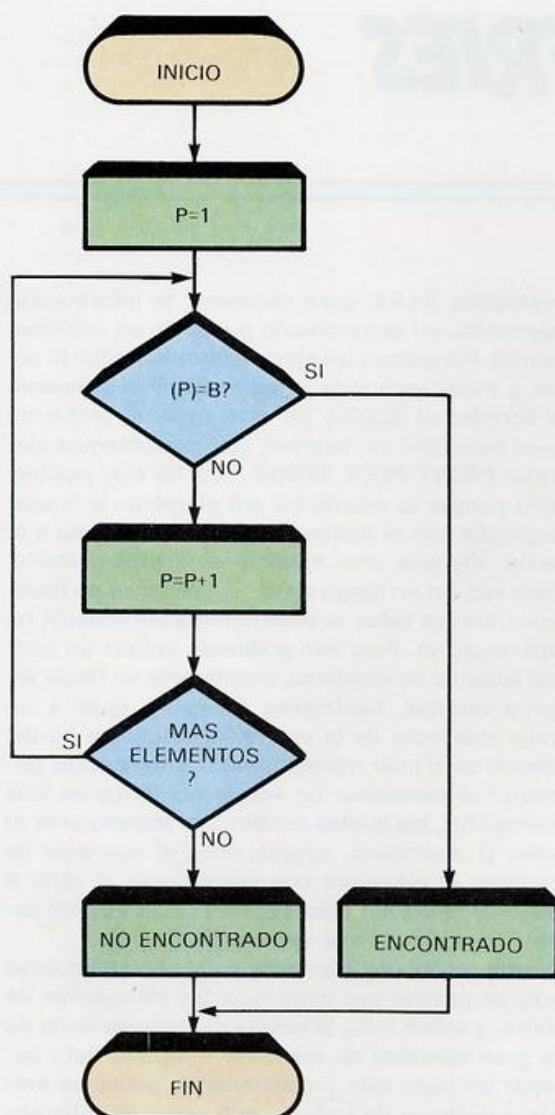


Diagrama de flujo de la búsqueda secuencial para series desordenadas.

tervalos. El primero de ellos es el más simple, y también el más ineficaz, si bien es cierto que el de extrapolación de intervalos no puede ser empleado sobre series desordenadas.

La búsqueda secuencial, como su propio nombre indica, consiste en ir analizando cada uno de los elementos de la serie, desde el primero hasta el último, siguiendo un orden de índices. De esta manera, estaremos seguros de localizar el elemento buscado, en caso de que exista. En el siguiente ejemplo, intentamos localizar un imaginario elemento **B**, en el vector **V (20)**:

```

10 FOR P=1 TO 20
20 IF V(P)=B THEN PRINT "ENCONTRADO
EN ";P: STOP
30 NEXT P
40 PRINT "NO ENCONTRADO EN LA SE-
RIE"
  
```

Si se dispone de la ventaja de una serie ordenada, el método de búsqueda secuencial puede mejorarse, evitando la necesidad de recorrer toda la serie para saber si el elemento existe. Suponiendo, por ejemplo, que la serie mantuviera un orden ascendente, sólo deberemos analizar los elementos, hasta el punto en el cual éstos sean superiores a **B**.

```

10 FOR P=1 TO 20
20 IF V(P)=B THEN PRINT "ENCONTRADO
EN ";P: STOP
30 IF V(P)>B THEN GO TO 50
40 NEXT P
50 PRINT "NO ENCONTRADO EN LA SE-
RIE"
  
```

El último sistema de búsqueda a estudiar es algo más complejo, y en la mayoría de los casos mucho más eficaz. Recordemos que presenta el inconveniente de sólo poder ser utilizado en series ordenadas, al igual que la segunda versión de nuestra rutina de búsqueda secuencial.

El método consiste en escoger el punto medio de la serie, y averiguar hacia qué mitad se encuentra el valor a localizar. La siguiente investigación considerará esta parte de la lista como el intervalo a estudiar, volviendo a realizar la división de éste en dos zonas. De esta manera, la longitud de la serie a estudiar se reduce cada vez a la mitad, con la seguridad de que, si existe, el elemento será localizado, y de no ser así, el programa se detendrá al alcanzar el intervalo la longitud cero. En el ejemplo, utilizaremos la variable **LI** para señalar el límite inferior del intervalo, **LS** para el superior, **L** para la longitud del mismo, y **P** como puntero de mitad de serie (elemento a analizar). Como en el caso anterior, suponemos la búsqueda de un valor dado **B**, en la serie ordenada **V (20)**.

```

10 LET L=20: LET LI=0
20 LET L=INT (L/2+.5)
30 LET P=LI+L
40 IF B>V(P) THEN LET LI=P: GO TO 70
50 IF B<V(P) THEN GO TO 70
60 PRINT "ENCONTRADO EN ";P:STOP
70 IF L>1 THEN GO TO 20
80 PRINT "NO ENCONTRADO EN LA SE-
RIE"
90 STOP
  
```

Tengamos en cuenta, que las rutinas de ejemplo anteriores no funcionarán directamente, dado que no se encuentra definido el valor a localizar **B**, ni tan siquiera la serie a tratar **V (20)**. Por otra parte, el tratamiento se ha particularizado en todos los casos para veinte elementos, aunque lógicamente este valor puede ser alterado a cualquiera que sea la longitud de la serie.



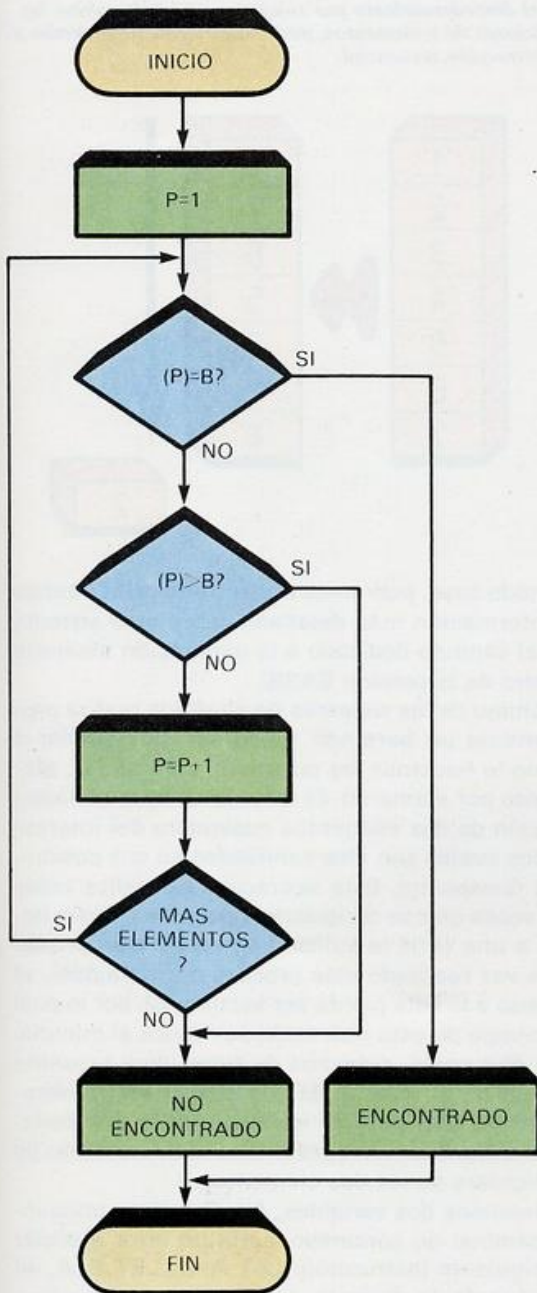


Diagrama de flujo de la búsqueda secuencial en una serie ordenada.

Una vez vistos los métodos de clasificación y búsqueda, nos ocuparemos por último de los *shuffling*, o sistemas de desordenación. Como ya hemos dicho, estos se emplean fundamentalmente en los juegos de azar que parten de series ordenadas; de hecho, su denominación inglesa (*barajado*), se deriva de la prácticamente ineludible utilización en los juegos de cartas. Como en el caso de las búsquedas, comentaremos los dos sistemas de desordenación más am-

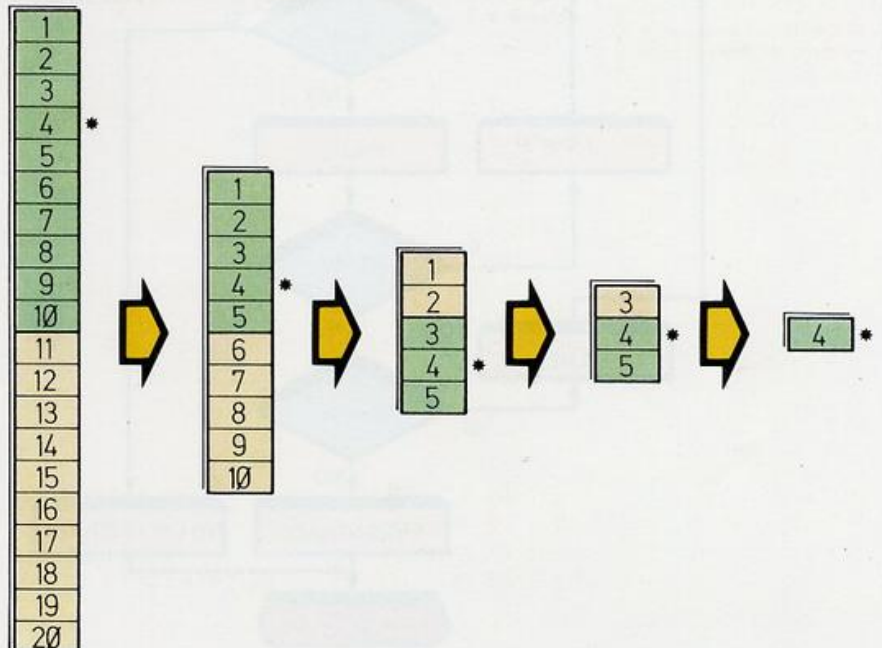
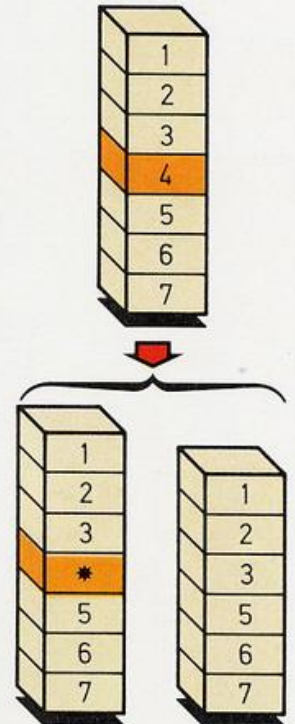
pliamente extendidos: el primero de ellos se basa en la extracción aleatoria de elementos, sin recurrir a la desordenación de la serie, debido a lo cual no se puede decir que sea exactamente un método de *shuffling*, aunque lo incluiremos en este epígrafe debido a que su efecto es similar; por el contrario, la desordenación por excelencia consiste en someter a la serie a un intercambio de elementos (*SWAPPING*).

Si siguiendo el sistema de extracción aleatoria, podemos optar por dos formas diferentes de tratar la serie una vez extraído determinado elemento: o bien se sustituye por una marca que señale la anterior extracción del mismo, o bien se elimina de la serie.

El primero de los métodos es en principio el más rápido, pero sin embargo, cuando aumenta el número de extracciones, va haciéndose más y más lento, puesto que aumentan las probabilidades de que el elemento extraído aleatoriamente haya aparecido con anterioridad. Por tanto, este sistema sólo se debe utilizar partiendo de un número bajo de extracciones en relación a la longitud de la serie.

Si por el contrario se pretende extraer todos los números de la serie, lo más conveniente será tomarnos el trabajo de eliminar el elemento extraído, por ejemplo, desplazando los superiores sobre él. De esta manera, el intervalo para la elección del número aleatorio se va reduciendo, y cualquier número aleatorio que se obtenga será admisible.

Por el método de extrapolación de intervalos, en cada paso reducimos a la mitad la longitud de la serie a investigar.



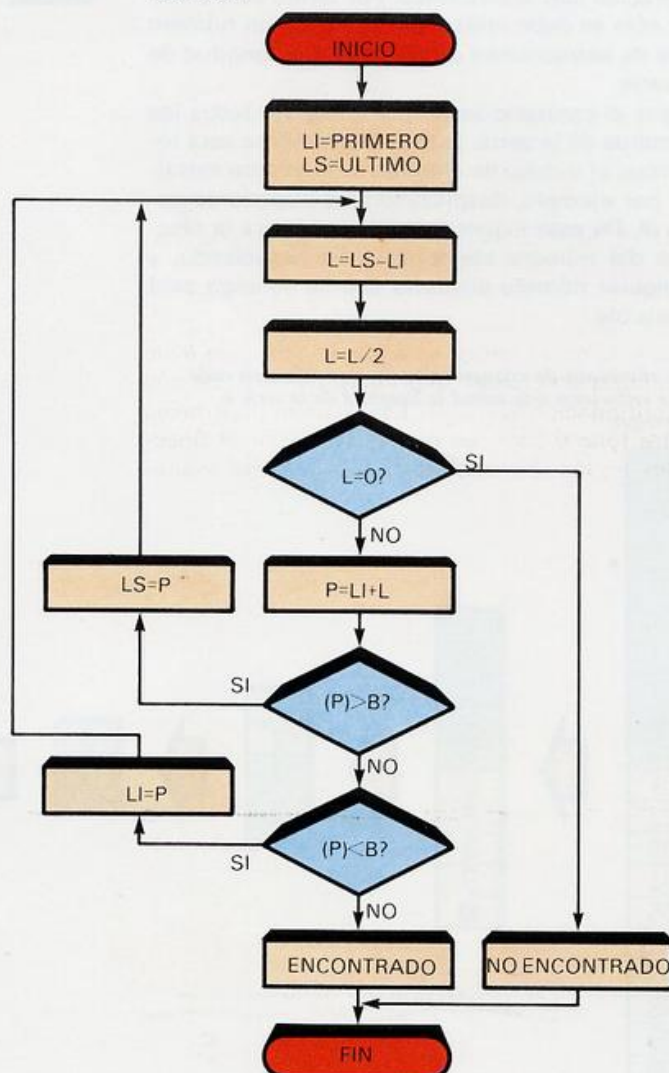


```

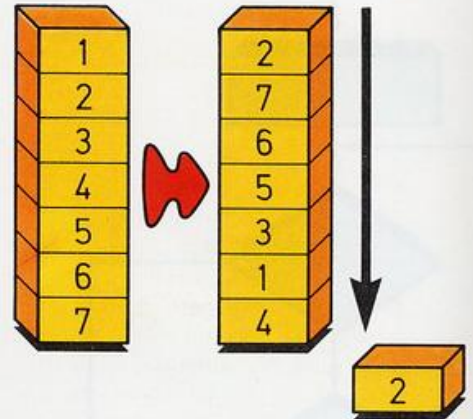
10 DIM D(6)
20 BORDER 4: PAPER 4: INK 0: BRIGHT 0:
FLASH 0: CLS
30 PLOT 79,128: DRAW 97,0: DRAW 0,-97:
DRAW -97,0: DRAW 0,97
40 PAPER 6: BRIGHT 1
50 FOR I=6 TO 17: PRINT AT I,10;"
  ": NEXT I
60 PRINT AT 6,10;"RESULTADO: 0";AT 8,1
3;"VECTOR";
70 PRINT AT 8,13; OVER 1;"-----"
80 FOR I=1 TO 6: PRINT AT 9+I,14;I;"-
0": NEXT I
90 LET R=INT (RND*6)+1
100 PRINT AT 6,21;R
110 IF D(R)=1 THEN PRINT AT 17,10;"SI
HA SALIDO": GO TO 130
120 LET D(R)=1: PRINT AT 17,10;"NO HA S
ALIDO"
130 PRINT AT 9+R,14; OVER 1; INVERSE 1;
  ;
140 PRINT AT 9+R,14; OVER 1; INVERSE 0;

150 IF INKEY$="" THEN GO TO 130
160 PRINT AT 9+R,14;R;" ";D(R)
170 FOR I=0 TO 9: NEXT I: GO TO 90
    
```

Diagrama de flujo del método de extrapolación de intervalos.



En el desordenamiento por *swapping*, se intercambian las posiciones de *n* elementos, para posteriormente proceder a su extracción secuencial.



En todo caso, podremos encontrar ciertas rutinas e información más detallada sobre este sistema en el capítulo dedicado a la generación aleatoria dentro de la sección BASIC.

El último de los sistemas de *shuffling* realiza propiamente un barajado, de forma muy similar a como lo hacemos las personas, pero eso sí, elemento por elemento. El método se basa en la selección de dos elementos cualquiera del intervalo, los cuales son intercambiados en sus posiciones (*swapping*). Esta operación se realiza todas las veces que se consideren oportunas, hasta llegar a una serie lo suficientemente desordenada. Una vez realizado este proceso de «barajado», el acceso a la lista puede ser secuencial, por lo cual el tiempo de esta operación se reduce al mínimo. Por otra parte, debemos de tener muy presente al realizar el intercambio de elementos, la necesidad de una variable «puente» en la que podamos almacenar temporalmente la información de cualquiera de los dos elementos.

Si tenemos dos variables, *A* y *B*, y deseamos intercambiar su contenido, sería un error ejecutar la siguiente instrucción: *LET A=B: LET B=A*. Al efectuarse la primera asignación, *A* pasaría a adoptar el contenido de *B*, perdiéndose su valor anterior. Por tanto, es necesaria la existencia de una variable «puente», la cual denominaremos *C* en el ejemplo, con la única misión de preservar el contenido de *A* contra la pérdida por la primera asignación.

Siguiendo este último sistema, formaremos una asignación «triangular» entre las variables *A*, *B* y *C*:

```

10 LET C=A: REM CONSERVA EN C EL VALOR INICIAL DE A
20 LET A=B: REM DEPOSITA EN A EL VALOR INICIAL DE B
30 LET B=C: REM DEPOSITA EN B EL VALOR INICIAL DE A
    
```



# INDEX



A mayoría de los microordenadores utilizan los casetes como unidad de almacenamiento masivo, tanto para datos como para programas; y nuestro Spectrum no es diferente en este sentido. Por otra parte, sigue siempre un mismo método para la lectura/escritura de información en este soporte magnético. Inicialmente, envía un pequeño bloque de datos de vital importancia, tales como: nombre para la identificación de la información, dirección de carga, longitud, etc. Este conjunto de datos se denomina CABECERA, o **HEADER** (pronunciado *híder*).

Cuando al Spectrum le ordenamos grabar una CABECERA, emite un primer tono de unos cinco segundos de duración, denominado **LEADER** (pronunciado *líder*) o simplemente **LEAD** («*lid*»), cuyo objetivo es, en la posterior lectura, anunciar al ordenador la llegada de una serie de datos, con el fin de que éste no se vea sorprendido por el próximo alubión de bits. Por otra parte, cuando se efectúa la grabación, el *lead* generado por el Spectrum sirve a su vez para regular el volumen de grabación de aquellos casetes que efectúan esta operación automáticamente.

Tras el *lead*, llega la cabecera, portando una información que estudiaremos unas líneas más abajo de una forma más detallada. El *header* es examinado por el Spectrum, y en caso de admitirse la información a introducir, se continúa la lectura. De no ser así, o bien se produce una detención por error, o bien se ignora el bloque de datos siguiente.

Las detenciones por error pueden ser de muy diverso tipo, y van desde las muy generales, como por ejemplo el *2 Variable not found* (2 Variable no encontrada), el cual aparece al intentar grabar una matriz inexistente, hasta los muy específicos, como es el caso de *F Invalid file name* (F Nombre no válido para la grabación), el cual hace acto de presencia cuando intentamos grabar un programa sin nombre, o con una denominación cuya longitud excede los diez caracteres máximos permitidos.

Seguido de la cabecera, y tras un pequeño intervalo de un segundo, se emite un nuevo *lead*, igual al primero y con la misma finalidad que éste, para inmediatamente después pasar a transferirse el bloque principal de información. Todo este proceso es realizado por una serie de rutinas implementadas en el **FIRMWARE** de nuestro Spectrum (ROM), conocidas bajo la denominación general

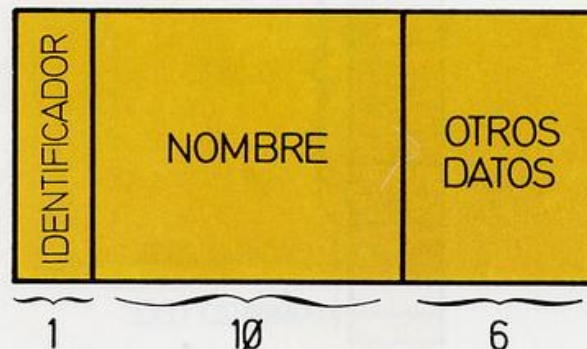


La estructura de la información almacenada en cinta por el Spectrum, se detalla en el gráfico superior.

de **CASSETTE HANDLING ROUTINES** (rutinas de manejo del casete). Esta zona de la ROM incluye el tratamiento de todos los comandos concierntes al casete (**SAVE, VERIFY, LOAD y MERGE**), así como las posibles modalidades de ellos (**CODE, SCREEN\$, etc...**), desde su sintaxis hasta su ejecución completa.

El espacio de un segundo que el Spectrum se toma entre la cabecera y el bloque principal de información, le obliga a incluir un segundo *lead*, con duración distinta al primero por motivos que a continuación veremos. Este tiempo intermedio entre tono y tono, es aprovechado por el Spectrum, en las operaciones de lectura, para exami-

Los diecisiete bytes de un header, se distribuyen en: identificador, nombre y datos diversos.





**!**

El programa carece de gráficos definidos, debido a lo cual no existen caracteres subrayados en el listado del mismo.

**\***

Para la grabación del programa introduciremos el siguiente comando: **SAVE "INDEX" LINE 10.**

**\***

El bloque inicial de datos que envía el Spectrum a la hora de almacenar información, se denomina **CABECERA**.

**\***

Toda la información almacenada en la ROM recibe el nombre de **FIRMWARE**.

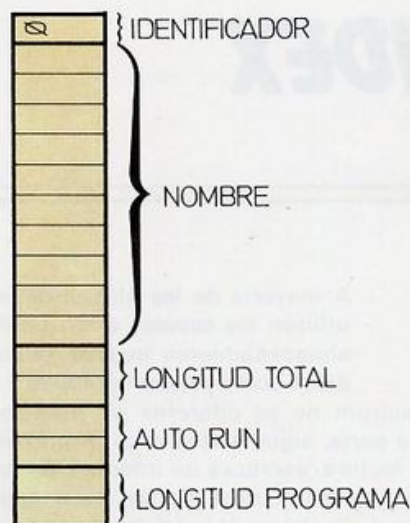
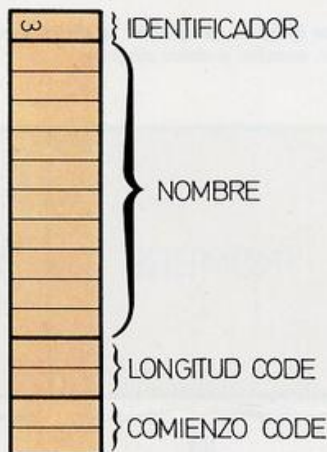
nar la información aportada por el *header* y operar en consecuencia.

Así, por ejemplo, averigua si existe suficiente memoria para la entrada de la información, si el nombre de los datos coincide con los que se desean leer, o si el tipo es el adecuado (programa, bytes, matrices)... En este sentido, cuando se decide que el bloque siguiente no es el indicado, se desecha su lectura, quedando el ordenador «a la escucha» de un nuevo *lead* de cabecera; de ahí que ambos tonos, los anteriores a cabeceras y a datos, sean diferentes, con el fin de permitir al Spectrum distinguir entre la llegada de uno u otro tipo de información.

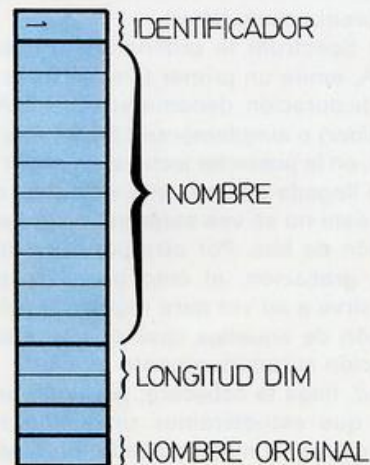
## EL PROGRAMA

Nuestro programa **INDEX**, nos permite conocer toda la información contenida en una cabecera, así como el índice o directorio de una cinta completa. Esta segunda misión del programa es muy útil, fundamentalmente para todos los que tengamos una impresora a nuestro servicio. Gracias a él, podremos dejar funcionando el casete con la cinta de la que deseamos conocer el contenido, y conforme vayan apareciendo las cabeceras, irán siendo escritas en la impresora.

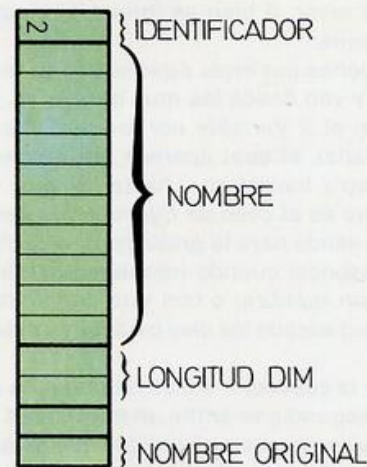
Al comienzo del programa, el Spectrum nos preguntará si deseamos la salida de datos por la pantalla o por la impresora. A continuación, si hemos escogido la segunda opción, se nos interrogará sobre el nombre de la cinta (identificador), con el fin de ser escrito al comienzo del índice.



*Header de una información del tipo programa.*



*Header de una información del tipo matriz numérica.*



*Header de una información del tipo matriz de cadena.*

*Header de una información del tipo bytes.*





## LISTADO ENSAMBLADOR

HEADER	175 55	XOR A SCF	;Pone a cero el acumulador. ;Conecta la bandera de ;acarreo.
	221 33 10 125	LD IX,\$32010	;Sitúa en 32010 la dirección ;de carga de la cabecera.
	205 86 5	CALL \$1366	;Llama a la subrutina ROM de ;lectura de cabeceras.
END	201	RET	;Retorna al BASIC.

En el caso de que elijamos la salida por pantalla, el Spectrum nos preguntará si deseamos que el mensaje *scroll* detenga la impresión cada vez que se llene la pantalla.

Una vez cumplidos estos requisitos previos, sólo

resta poner en marcha el casete con la cinta a examinar, y por supuesto conectar los cables EAR del Spectrum, como si de una lectura cualquiera se tratase. El programa esperará pacientemente las cabeceras de la cinta, y nos brindará por el





```

5 REM *****
10 REM * J.M.MAYORAL SERRANO *
15 REM *****
16 REM * INDEX 1985 *
17 REM *****
20 CLEAR 31999
30 FOR I=32000 TO 32009
40 READ J: POKE I,J
50 NEXT I
60 DATA 175,55,221,33,10,125,205,86,5,201
70 POKE 23658,8: LET S=0: LET M=0
80 BORDER 1: PAPER 1: OVER 0: INK 7: CLS
90 GO SUB 1000
100 PRINT AT 2,0;"DESEA SALIDA POR IMPRESORA?"
110 PRINT AT 2,29; OVER 1; INVERSE 1;" "
120 IF INKEY$="S" THEN PRINT AT 2,29;"SI": LET P=1:
GO TO 210
130 IF INKEY$="N" THEN PRINT AT 2,29;"NO": LET P=0:
GO TO 150
140 GO TO 110
150 IF INKEY$(">") THEN GO TO 150
160 PRINT AT 4,0;"CON MENSAJE DE ~SCROLL~?"
170 PRINT AT 4,29; OVER 1; INVERSE 1;" "
180 IF INKEY$="S" THEN PRINT AT 4,29;"SI": GO TO 24
0
190 IF INKEY$="N" THEN PRINT AT 4,29;"NO": LET S=1:
GO TO 240
200 GO TO 170
210 INPUT "IDENTIFICADOR DE CINTA?";N$
220 OPEN #2,"P": PRINT N$: PRINT
230 GO TO 250
240 GO SUB 1000: PRINT
250 POKE 32010,255
260 RANDOMIZE USR 32000
270 IF PEEK 32010>3 THEN GO TO 260
280 PRINT "NOMBRE.....";: FOR I=32011 TO 32020:
PRINT CHR$ PEEK I;: NEXT I
290 PRINT : PRINT "TIPO....."
300 IF PEEK 32010 THEN GO TO 370
310 PRINT "PROGRAMA"
320 PRINT "LONGITUD TOTAL.:";: LET I=PEEK 32021+256*
PEEK 32022: PRINT I: LET M=M+I
330 PRINT "LONGITUD PROG.:";PEEK 32025+256*PEEK 320
26
340 PRINT "LINEA "AUTORUN":";: LET I=PEEK 32023+25
6*PEEK 32024
350 IF I=9999 THEN PRINT "NO HAY": GO TO 370
360 PRINT I: GO TO 490
370 IF PEEK 32010>1 THEN GO TO 400
380 PRINT "'DIM' NUMERICO"
390 GO TO 420
400 IF PEEK 32010>2 THEN GO TO 450
410 PRINT "'DIM' ALFANUM."
420 PRINT "LONGITUD 'DIM'.:";: LET I=PEEK 32021*25
6*PEEK 32022: PRINT I: LET M=M+I
430 PRINT "NOMBRE ORIGINAL.:";CHR$(64+32*(PEEK 32024
/32-INT (PEEK 32024/32)));"$" AND PEEK 32010=2
440 GO TO 490
450 PRINT "BYTES"
460 PRINT "LONGITUD CODE.:";: LET I=PEEK 32021+256*
PEEK 32022: PRINT I: LET M=M+I
470 PRINT "COMIENZO CODE.:";: LET J=PEEK 32023+256*
PEEK 32024: PRINT J
480 IF I=6912 AND J=16384 THEN PRINT "IMAGEN DE PAN
TALLA"
490 PRINT : PRINT : IF S THEN POKE 23692,255
500 GO TO 250
1000 CLS : PRINT INVERSE 1;" INDEX
1010 RETURN

```

medio escogido toda la información de interés: Si se trata de un programa, indicará su nombre, longitud total (programa más variables), longitud del área de variables, y línea de autoejecución (si ésta existiera).

En la lectura de bytes, anunciará su denominación, longitud y dirección de carga, así como la particularidad de si se trata de una imagen de pantalla (**SCREEN\$**).

Por último, ante la lectura de matrices, nos indicará el tipo de que se trata (numérica o de cadena), su nombre, ocupación en bytes, y nombre que tenía en el área de variables en el momento de la grabación.

Además de todo esto, el programa inicializa una variable al comenzar el índice, en la cual va acumulando las longitudes de todos las informaciones examinadas; de esta manera, al llegar al final de la cinta, podemos conocer el total de bytes de información neta contenidos en ella (sin contar con los diecisiete de cada cabecera), sólo con preguntar el contenido de la variable **M**.

## LECTURA DE CABECERAS

El código máquina del programa es el encargado de efectuar la lectura de las cabeceras, mientras que el BASIC sólo se ocupa de la interpretación de la información que contienen. Tal como se puede observar en el listado ensamblador, se trata de una pequeña subrutina que indica la dirección de memoria en la que se quiere almacenar la cabecera, para su posterior examen (de 32011 en adelante), y prepara el acumulador y la bandera de acarreo, para la llamada a la subrutina de la ROM que efectúa la lectura del **header** (1366 decimal).

La información de cabecera está compuesta por diecisiete bytes, de los cuales los once primeros siempre tienen una misma misión: el primero de ellos es un identificador del tipo de información recogida (0=programa, 1=matriz numérica, 2=matriz alfanumérica y 3=bytes), y los diez siguientes el nombre con que fue almacenada la información, completada con espacios en blanco por la derecha (código 32), en caso de que no llegue a los diez caracteres.

Los últimos seis bytes de información tienen un significado variable, según el identificador de datos. En todo caso, son los datos indispensables para el correcto funcionamiento de todos los comandos de casete: **SAVE**, **VERIFY**, **LOAD** y **MERGE**.

