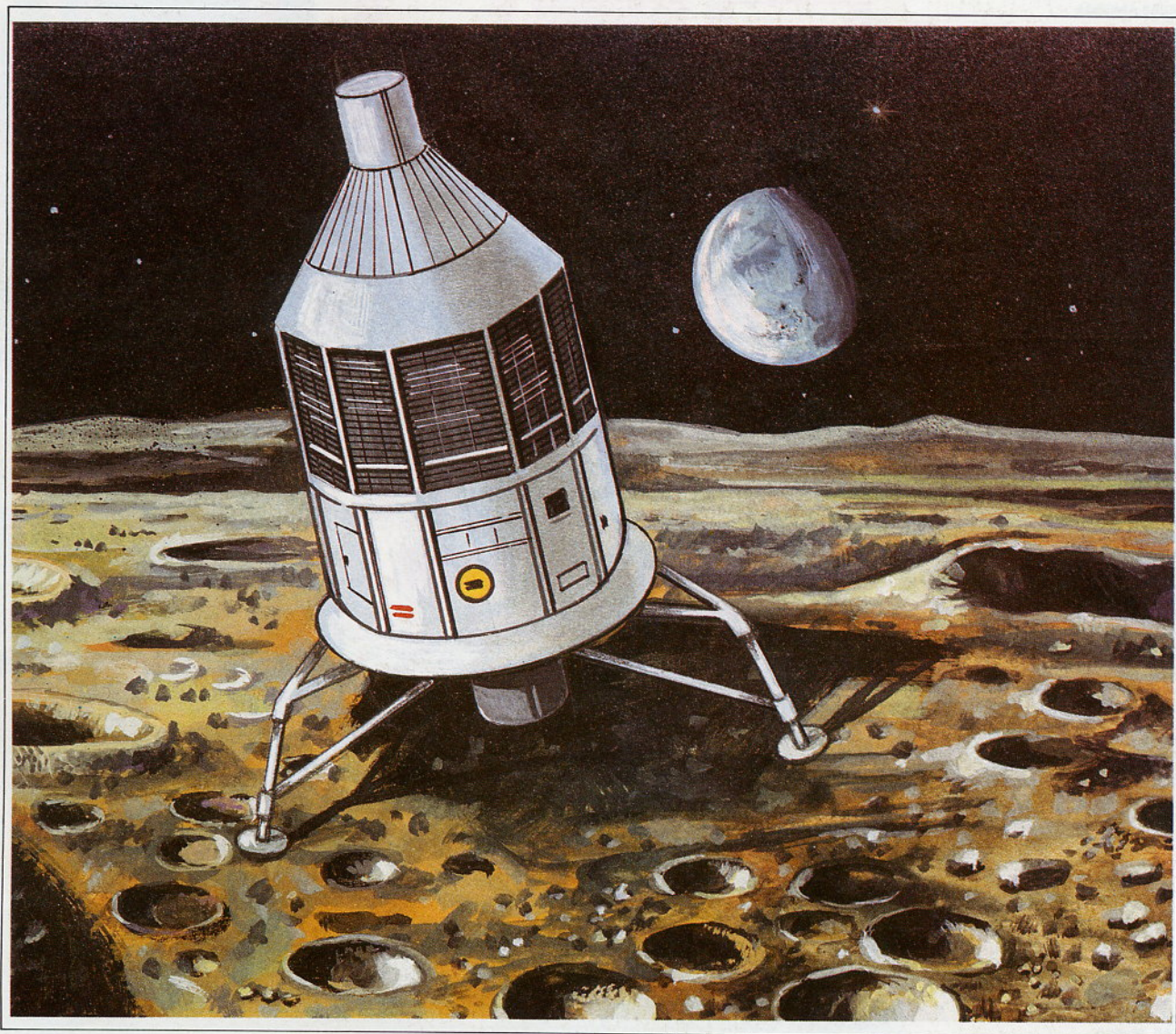


8
150pts.

AVAN

Enciclopedia Práctica del Spectrum



Nueva Lente/Ingelek





UNA Y OTRA VEZ



En los ejemplos que hemos visto hasta ahora se plantea siempre una estructura cíclica, en la que el programa acepta datos del teclado (INPUT), efectúa determinados cálculos o evaluaciones con ellos y, por último, produce una salida impresa de los resultados, antes de volver a ejecutarse.

A esta estructura de programa la denominamos ciclo: en ella, las cosas se repiten una y otra vez hasta que no ordenamos lo contrario, con la respuesta a un mensaje de si deseamos continuar o no, o simplemente interrumpiendo la ejecución mediante BREAK.

Por supuesto que también podemos dejar que estos programas terminen después de la impresión de los resultados y que, en el caso de que se desee continuar, vuelvan a ponerse en ejecución desde la primera sentencia, por medio del comando RUN.

REPITIENDO

Dentro de esta gran estructura cíclica, nos es muy conveniente poder incluir otro tipo de construcciones, a las que denominamos BUCLES, y que resultan de gran utilidad cuando deseamos repetir un bloque de instrucciones un determinado número de veces.

El formato que nos va a permitir codificar estas estructuras de bucle, se basa en tres palabras clave y el nombre de variable que designemos para controlar dicha estructura, adoptando la forma:

```
FOR variable = valor-inicial TO valor-final STEP
incremento
instrucción
...
instrucción
NEXT variable
```

Supongamos que queremos obtener la suma de los diez primeros números naturales; codificando el programa de la forma que hasta ahora conocemos, podríamos obtener algo como esto:

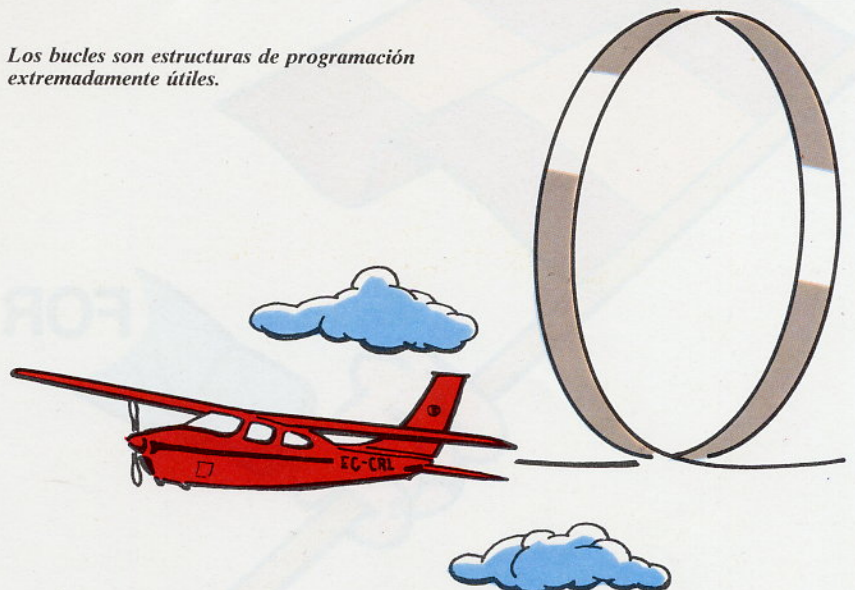
```
10 REM - SUMA DE NUMEROS
20 LET S=0
30 LET N=0
40 LET N=N+1
50 LET S=S+N
60 IF N<10 THEN GO TO 40
70 CLS
80 PRINT "La suma es: ";S
```

Aplicando los conocimientos recientemente adquiridos, podríamos ahorrar instrucciones y ganar en claridad escribiendo:

```
10 REM - SUMA DE NUMEROS
20 LET S=0
30 FOR N=1 TO 10
40 LET S=S+N
50 NEXT N
60 CLS
70 PRINT "La suma es: ";S
```

La estructura del bucle que hemos propuesto se encuentra en las líneas 30 a 50; la primera contiene el nombre de la variable y los valores mínimo y máximo que debe adoptar, así como el incremento por cada nueva pasada. La segunda es la operación que debe realizarse para los diferentes valores de N. Por último, la tercera contiene el final del bucle, especificando el nombre de la variable que lo controla.

Los bucles son estructuras de programación extremadamente útiles.



Al agrupar instrucciones en una misma línea, las sentencias de bifurcación sólo pueden dirigirse al grupo completo.

*

No existe limitación en cuanto a los toques inferior y superior marcados para el FOR, ni en cuanto al STEP o paso que se fija. Estos pueden ser números enteros, fraccionarios, negativos o decimales, así como variables.

i!

Los bucles **FOR NEXT** no pueden entremezclarse. O bien se encuentra uno totalmente en el interior del otro, o son independientes.

*

Con el signo de puntuación dos puntos (:), pueden codificarse cuantas instrucciones se deseen agrupadas bajo un mismo número de línea.

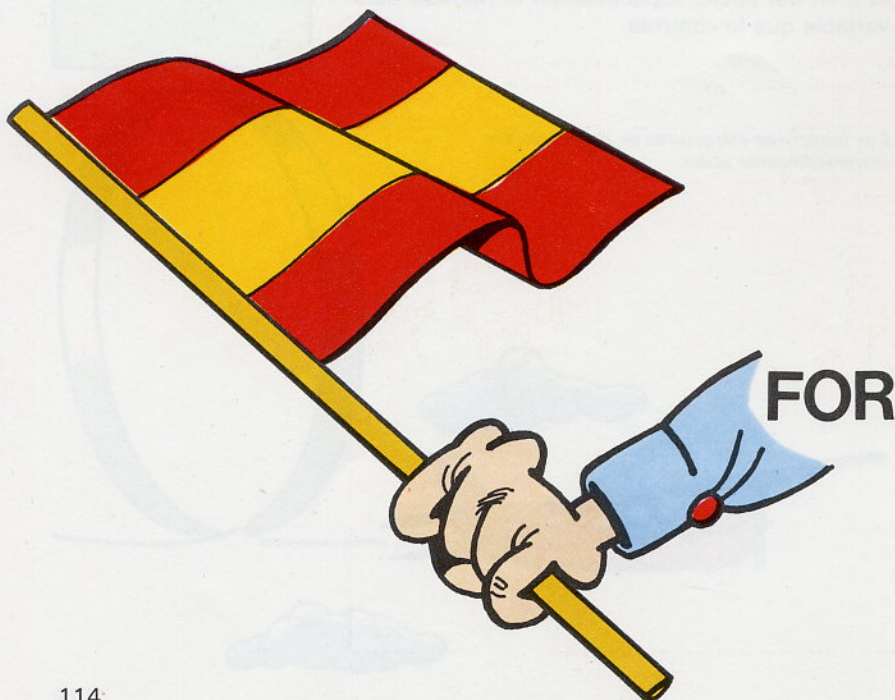
En nuestro ejemplo, el programa va a recorrer las instrucciones 30 a 50 diez veces, siendo los respectivos valores de **N** en cada pasada 1, 2, 3, 4 ... 10. Como el caso más normal en los bucles **FOR-NEXT** es que el incremento de la variable sea 1, puede omitirse el **STEP** cuando el incremento sea la unidad: **30 FOR N=1 TO 10**, equivaldría por tanto a **30 FOR N=1 TO 10 STEP 1**. Lo que realmente hace el ordenador cuando se encuentra con un bucle de este tipo, es muy parecido al primer ejemplo de programa propuesto por nosotros: primero asigna a la variable del **FOR** el valor mínimo y ejecuta todas las instrucciones que siguen hasta encontrar el **NEXT**, entonces suma el **STEP** a la variable, comparando si ésta es mayor que el valor máximo. En caso de que lo sea, se sale del bucle y continúa con la siguiente instrucción en secuencia, en nuestro ejemplo la 60. Si resulta ser menor o igual, efectúa una bifurcación al comienzo del bucle, inmediatamente después del **FOR**.

Para practicar un poco, obtendremos ahora la suma de los 50 primeros números pares (2550):

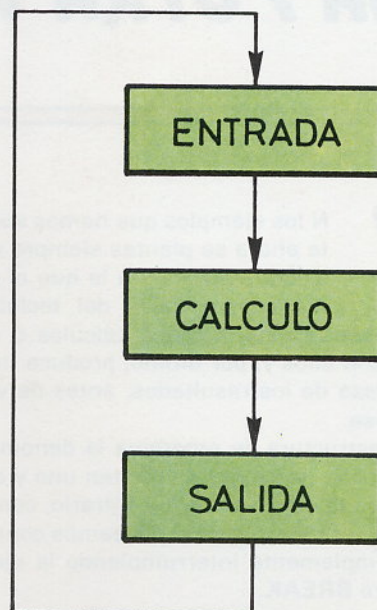
```
10 REM - SUMA DE NUMEROS PARES
20 LET S=0
30 FOR N=2 TO 100 STEP 2
40 LET S=S+N
50 NEXT N
60 CLS
70 PRINT "La suma de pares es: ";S
```

Esta vez es absolutamente necesario especificar

La sentencia FOR es la «señal de partida» de los bucles BASIC.



Muy frecuentemente, las estructuras de los programas resultan ser cíclicas.



el incremento de la variable (**STEP**), puesto que es diferente de la unidad. También hemos podido ver lo fácil que resulta adaptar el programa utilizando esta estructura de bucle.

Una de las pocas restricciones que tiene la estructura **FOR-NEXT**, es que el nombre de la variable que sirve de contador debe contener un solo carácter, es decir, no son válidas como variables de bucle las compuestas por más de una letra. Lejos de representar un inconveniente, debemos decir que es práctica común entre programadores el uso de variables de **FOR** de un solo carácter, y preferentemente: **I, J, K** ... Las variables de control de **FOR** se suelen denominar también **INDICES**.

Por lo demás, no existe limitación alguna en cuanto a los topes inferior y superior marcados en el **FOR**, ni en cuanto al **STEP** o paso que se fija. Estos pueden ser números enteros, fraccionarios, negativos o decimales, así como variables. Algunos ejemplos válidos son:

```
FOR I=9 TO -9 STEP -1
FOR I=3/4 TO 3/8 STEP -.001
FOR I=0 TO 99 STEP .001
FOR I=A TO B STEP C
```

Otra característica es que no es necesario que el límite superior sea rebasado exactamente por la unidad o por la cantidad especificada como **STEP** para que el bucle llegue a su fin, sino que basta con que el valor adquirido por la variable de control en el incremento (**NEXT**) sea mayor que el límite superior, cuando el paso (**STEP**) es positivo. De forma análoga, cuando el paso es negativo, se saldrá del bucle al alcanzar un valor que sea menor. Veamos un ejemplo:



STEP



espacial, que en atención a los más impacientes, será tan solo de 20 segundos:

```
10 REM - CUENTA ATRAS
20 FOR I=20 TO 0 STEP -1
30 CLS
40 PRINT I
50 PAUSE 50
60 NEXT I
```

La instrucción **PAUSE** con el argumento 50 produce una parada de aproximadamente un segundo, por lo que el efecto quedará bastante logrado, siempre que nadie opte por darle una velocidad fulgurante al conteo, manteniendo pulsada una tecla, lo que convertiría el retardo en prácticamente insignificante. Como ejercicio, podemos

La sentencia **STEP** sirve para marcar el paso de las variables índice en los bucles **FOR-NEXT**.

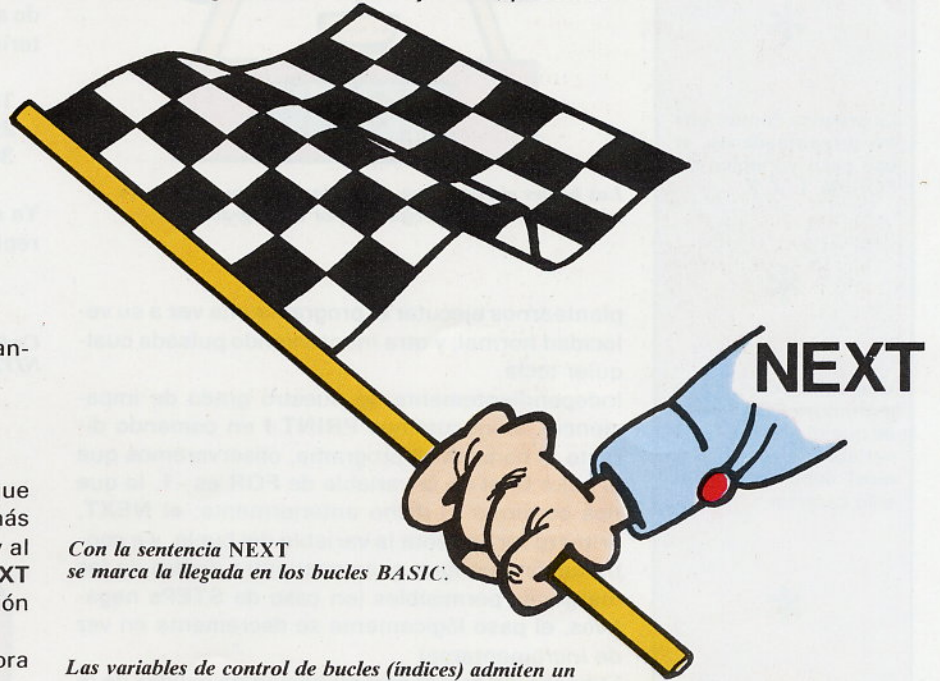
```
10 FOR I=1 TO 20 STEP 3
20 PRINT I
30 NEXT I
40 PRINT "Valor de salida: ";I
```

Los sucesivos valores de **I** que serán impresos antes de finalizar el bucle serán:

1, 4, 7, 10, 13, 16 y 19

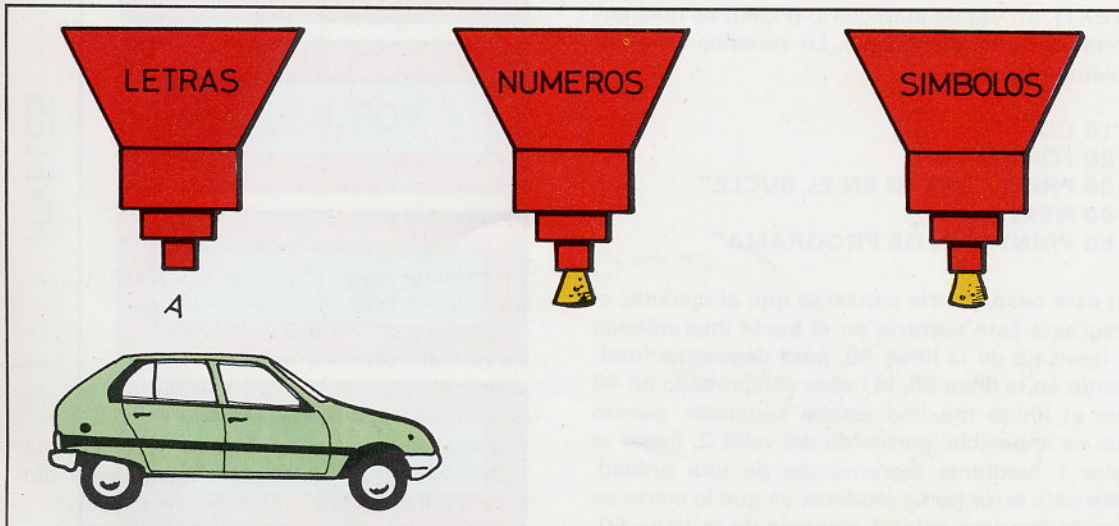
Y el valor final de la variable **I** será **22**, ya que la instrucción **NEXT** la incrementó una vez más para comparar si superaba el límite máximo, y al cerciorarse de ello, salió del bucle **FOR-NEXT** para imprimir el valor final de **I** en la instrucción 40.

Como aplicación de lo dicho, simularemos ahora la cuenta atrás para el lanzamiento de una nave



Con la sentencia **NEXT** se marca la llegada en los bucles **BASIC**.

Las variables de control de bucles (índices) admiten un solo carácter alfabético para su denominación.





i!

Existen unas estructuras de programa denominadas BUCLES, que nos permiten recorrer un determinado bloque de instrucciones un número concreto de veces.



Es práctica común entre programadores, el uso para variables de FOR de: I, J, K ...



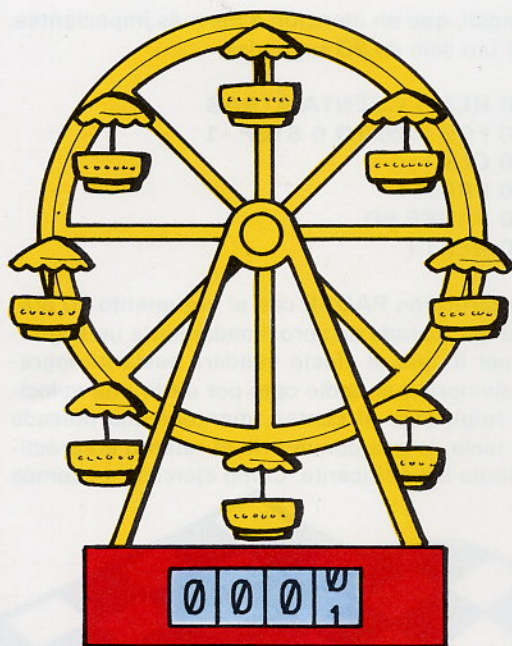
Una restricción de la estructura FOR NEXT es que el nombre de la variable de control (índice) debe ser de un solo carácter.



El formato que permite codificar las estructuras de bucle, se basa en tres palabras clave y el nombre de variable de control que designemos, adoptando la forma: **FOR** variable=valor-inic. **TO** valor-final **STEP** incremento



Cuando una estructura de bucle incluye a otro o varios se denomina «anidamiento».



Los bucles ejecutan una determinada tarea durante un número de veces especificado por el programador.

plantearnos ejecutar el programa una vez a su velocidad normal, y otra manteniendo pulsada cualquier tecla.

Independientemente de nuestro grado de impaciencia, si ejecutamos **PRINT I** en comando directo al concluir el programa, observaremos que el valor final de la variable de **FOR** es -1, lo que nos confirma lo dicho anteriormente: el **NEXT**, primero incrementa la variable del bucle, y a continuación averigua si se encuentra dentro de los márgenes permisibles (en caso de **STEPs** negativos, el paso lógicamente se decrementa en vez de incrementarse).

El hecho de que la comprobación de validez de la variable de control se realice al salir del bucle (**NEXT**), en vez de al iniciarlo (**FOR**), es más importante de lo que parece. Lo veremos en el siguiente ejemplo:

```
10 CLS
20 FOR I=2 TO 1
30 PRINT "ENTRO EN EL BUCLE"
40 NEXT I
50 PRINT "FIN DE PROGRAMA"
```

En este caso, podría pensarse que al ejecutar el programa éste entraría en el bucle imprimiendo el mensaje de la línea 30, para detenerse finalmente en la línea 50, al haber comprobado en 40 que el límite máximo estaba superado, puesto que es imposible, partiendo del valor 2, llegar al valor 1 mediante incrementos de una unidad. Pero este error no se produce, ya que lo cierto es que se imprime sólo el mensaje de la línea 50,

porque antes de entrar en el bucle, el intérprete del BASIC Spectrum comprueba si la variable del **FOR** se encuentra ya fuera del rango, y si es así, no ejecuta el bucle. Esta es una comprobación excepcional que el ordenador sólo realiza la primera vez (el resto, como ya sabemos, se efectúan al llegar al **NEXT**).

También nos puede suceder que codifiquemos un bucle infinito, especificando un incremento nulo:

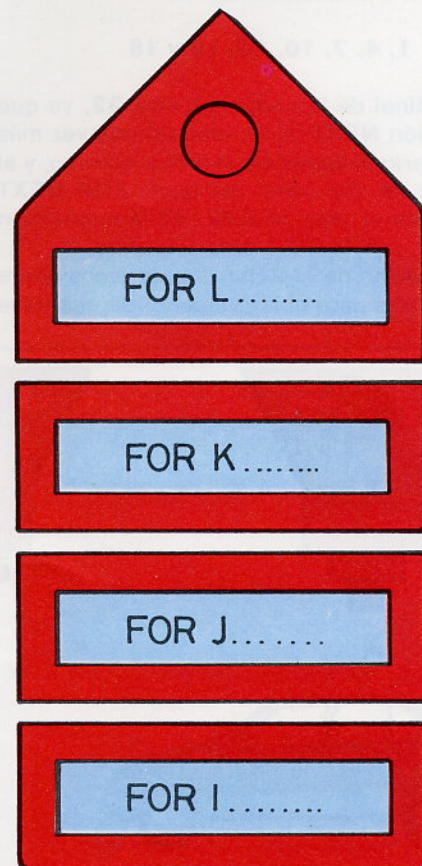
```
10 CLS
20 FOR I=0 TO 0 STEP 0
30 PRINT "OTRA VEZ"
40 NEXT I
```

Lo que realmente se consigue con esto es no emplear la sentencia imperativa **GO TO**; recurriendo a este método de bucle sin fin, el programa anterior produce los mismos resultados que:

```
10 CLS
20 PRINT "OTRA VEZ"
30 GO TO 20
```

Ya que en ambos casos la instrucción **PRINT** se repite de modo indefinido y la única forma de sa-

Cada paso en el anidamiento de bucles se conoce como **NIVEL**.



lir de este embrollo es mediante la pulsación de CAPS SHIFT y SPACE (BREAK).

RIZANDO EL RIZO

También existe la posibilidad de construir estructuras más complejas, incluyendo uno o más bucles dentro de otro. A este sistema de programación se le denomina ANIDAMIENTO de bucles, y tiene como restricción que, lógicamente, los bucles no pueden entremezclarse, es decir, o bien se encuentra uno totalmente en el interior del otro (diferente nivel), o bien son totalmente independientes (del mismo nivel).

En el primer caso, las variables de control elegidas deben ser obligatoriamente distintas. En el segundo, pueden serlo o no, ya que al ser los bucles independientes es necesario que termine uno para que comience el siguiente, por lo que no puede existir problema alguno.

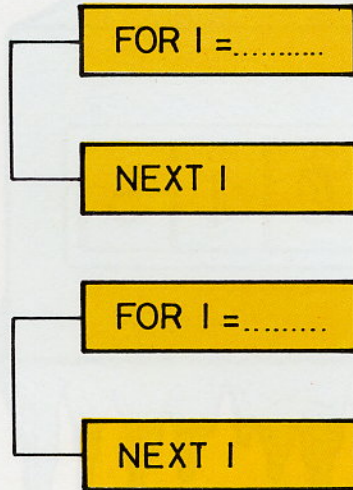
Veamos ahora un ejemplo de anidamiento de bucles, escribiendo una lista de números desde el 101 al 276, distribuida en 22 líneas de 8 números:

```
10 REM - LISTA DE NUMEROS
20 CLS
30 FOR I=101 TO 276 STEP 8
40 FOR J=I TO I+7
50 PRINT " ";J;
60 NEXT J
70 NEXT I
```

Debemos fijarnos en que los bucles son de diferentes variables I y J, y que el bucle J es de menor nivel que el I, ya que se encuentra contenido dentro de él, al estar el NEXT de J antes que el de I. Esta es la única estructura válida y correcta para codificar bucles anidados.

Cuando el programa alcanza la instrucción 30 por primera vez, I toma el valor 101. A continuación pasa a la instrucción 40 donde J toma el valor de I. En la instrucción 50 se imprime un espacio como separador y el valor de la variable J, sin efectuar el retorno de carro, dado que la instrucción concluye con un punto y coma (;). El programa alcanza entonces por primera vez la instrucción 60, donde suma 1 a J pasando a valer 2. Este momento lo aprovecha para comprobar que se encuentra dentro de límites, por lo que bifurca a la instrucción siguiente a la 40 para volver a imprimir, repitiéndose este bucle hasta 8 veces (la primera y siete más).

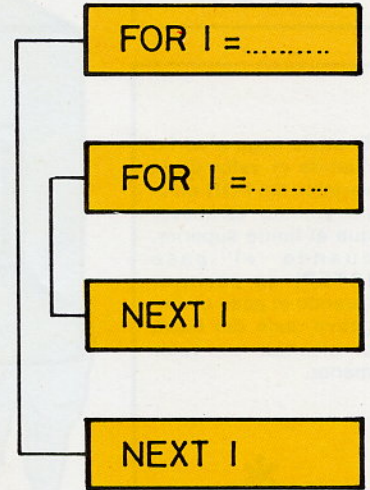
CORRECTO



Nunca podremos utilizar una misma variable de control para dos bucles anidados; sin embargo, no existirá problema en emplearla en estructuras independientes.

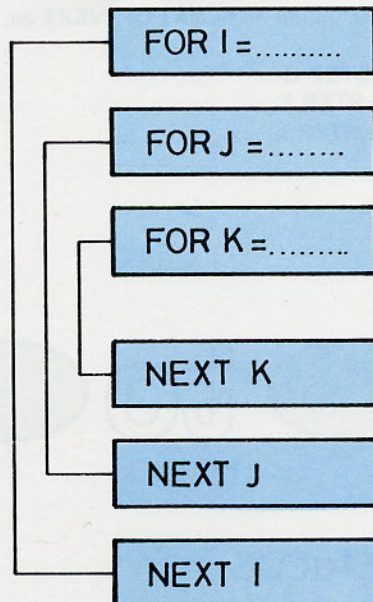
Después del último incremento de J, al volver el programa a pasar por la instrucción 60, comprueba que se encuentra ya fuera de límites, por lo que se sale del bucle de J y cae en la instrucción

INCORRECTO

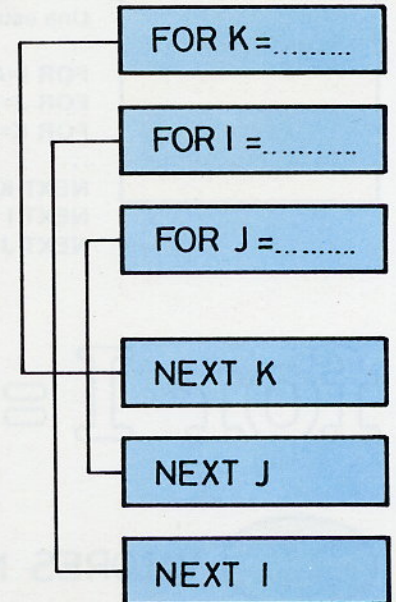


Si en una estructura de bucles anidados unimos mediante líneas las instrucciones FOR con sus correspondientes NEXT, nunca se deberán cruzar dichas líneas; de no ser así, el anidamiento se habrá efectuado de manera incorrecta.

CORRECTO



INCORRECTO



i!

El bucle llega a su fin cuando el valor de la variable del FOR en el incremento es mayor que el límite superior, cuando el paso (STEP) es positivo. Cuando el paso es negativo, sale del bucle al alcanzar un valor menor.

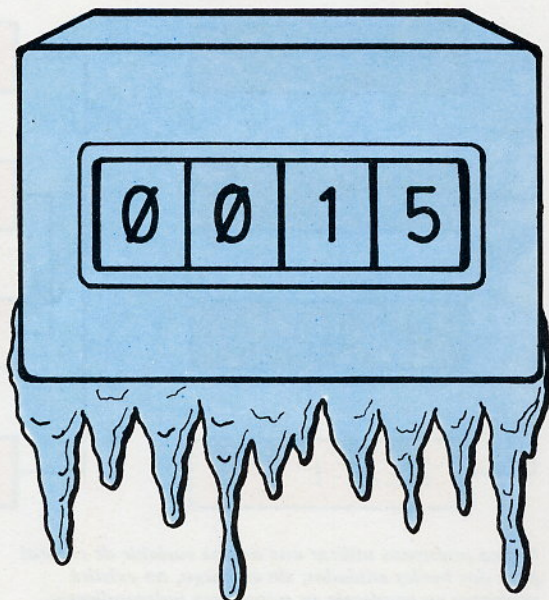
*

La instrucción PAUSE con el argumento 50 produce una parada de aproximadamente un segundo.

*

Podemos codificar un bucle infinito, especificando un incremento nulo, con lo que se consigue prescindir de la sentencia imperativa GO TO.

STEP 0



El STEP 0 bloquea la acción de NEXT, constituyendo bucles sin fin.

70 que incrementa I (el bucle exterior), y se comprueba que el valor de I se encuentra aún dentro de límites, por lo que se transfiere la ejecución del programa a la línea siguiente a la 30, donde vuelve a comenzar el bucle de J (el de menor nivel).

Este estado de cosas seguirá repitiéndose hasta que J primero e I después, sobrepasen los límites superiores fijados en sus correspondientes instrucciones FOR, en las líneas 30 y 40.

Una estructura no válida de bucle FOR NEXT es:

```
FOR I=A TO B STEP C
FOR J=D TO E STEP F
FOR K=G TO H STEP L
...
NEXT K
NEXT I
NEXT J
```

En este bucle está alterado el orden lógico de las instrucciones NEXT. Ya que el bucle de la variable J es interior (menor nivel) al de la variable I; la posición de los NEXT debe ser la contraria. Cuando los bucles están anidados (uno dentro de otro), siempre deben cerrarse primero los bucles más interiores y después los exteriores. La estructura siguiente es, por tanto, la correcta:

```
FOR I=A TO B STEP C
FOR J=D TO E STEP F
FOR K=G TO H STEP L
...
NEXT K
NEXT J
NEXT I
```



En las instrucciones FOR se inicializa el número de veces que deseamos se ejecute el bucle correspondiente.

Tanto para los límites como para el paso de una estructura de bucle, habremos de emplear forzosamente valores numéricos.

for I = ○ to ○ step ○



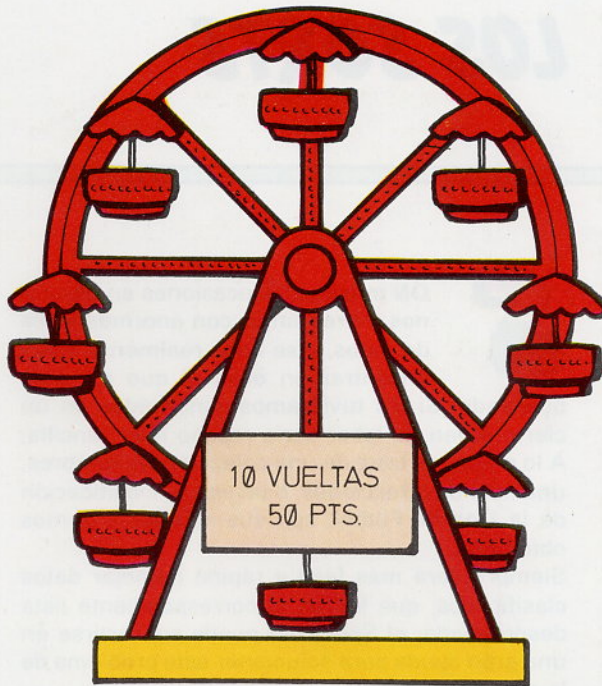
VALORES NUMERICOS



Otra estructura también correcta es la de bucles completamente separados uno de otro, es decir, no anidados:

```
FOR I=A TO B STEP C
...
NEXT I
FOR J=D TO E STEP F
...
NEXT J
FOR K=G TO H STEP L
...
NEXT K
```

Por supuesto, los resultados que se obtienen en un caso y otro son completamente diferentes, por lo que el programador debe elegir en cada momento la estructura que le conviene. Con todo ello, la idea que debe quedar clara es la de codificar los bucles correctamente, de modo que no se entrecrucen, bien de forma anidada o completamente independientes, según las necesidades.



La ejecución de un bucle supone la realización repetida de una determinada tarea; de ahí que este tipo de estructuras sea también conocido bajo el nombre de ITERATIVAS.


MAS DE UNA INSTRUCCION POR LINEA

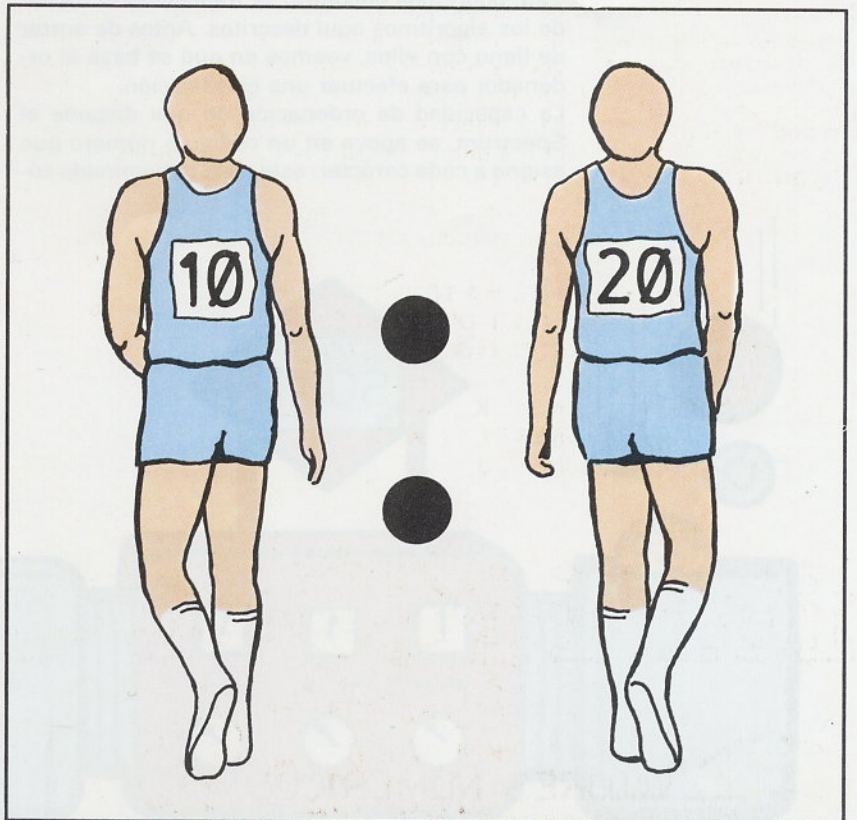
El separador dos puntos (:) se emplea para combinar varias instrucciones en una misma línea.

Hasta ahora hemos visto programas en los que cada línea está compuesta por una sola instrucción BASIC. Con el signo de puntuación dos puntos (:), pueden codificarse cuantas instrucciones se deseen, agrupadas bajo un mismo número de línea.

La única restricción consiste en que al agrupar instrucciones bajo un mismo número de línea, las sentencias de bifurcación sólo pueden dirigirse al grupo completo, y no a una concreta de ellas, por lo que las sentencias que agrupemos se ejecutarán siempre en bloque; todas o ninguna:

```
10 REM - SUMA DE NUMEROS
20 LET S=0:LET N=0
30 LET N=N+1:LET S=S+N
40 IF N<10 THEN GO TO 30
50 CLS:PRINT "La suma es: ";S
```

En este caso, no pueden agruparse las instrucciones de la línea 20 con las de la 30, ya que las primeras se ejecutan una única vez y las de la línea 30 diez veces, a causa de la bifurcación imperativa GO TO de la línea 40. Las agrupaciones de las líneas 20, 30 y 50, son perfectamente válidas y nos ahorran líneas de programa. 



LOS SORTS



ON muchas las ocasiones en las que nos enfrentamos con enormes listas de datos, y se hace realmente difícil encontrar en ellas el que estamos buscando. Si los tuviéramos clasificados en un cierto orden, la labor sería mucho más sencilla. A lo mejor, se trata de una colección de nombres, una lista de direcciones, o tal vez, la clasificación de la Liga de Fútbol, con sus equipos y puntos obtenidos.

Siempre será más fácil y rápido manejar datos clasificados, que los de la correspondiente lista desordenada; el Spectrum puede convertirse en una gran ayuda para solucionar este problema de la clasificación.

Los programas que gestionan este trabajo se denominan clasificadores o SORTs, y no se puede decir que unos sean netamente mejores que otros; simplemente, cada cual ofrece mayor interés en determinados casos.

A continuación expondremos los fundamentos de ocho sistemas de SORT y en las páginas 122 y 123, podremos encontrar la traducción al BASIC de los algoritmos aquí descritos. Antes de entrar de lleno con ellos, veamos en qué se basa el ordenador para efectuar una clasificación.

La capacidad de ordenación de que dispone el Spectrum, se apoya en un código o número que asigna a cada carácter; este es el denominado có-

digo A.S.C.I.I. (*American Standard Code for Information Interchange*, Código Estandarizado Americano para Intercambio de Información), en el que todos los dígitos, letras o caracteres especiales de los que dispone nuestro Spectrum, llevan asociados un número, entre 0 y 255 (su código A.S.C.I.I.).

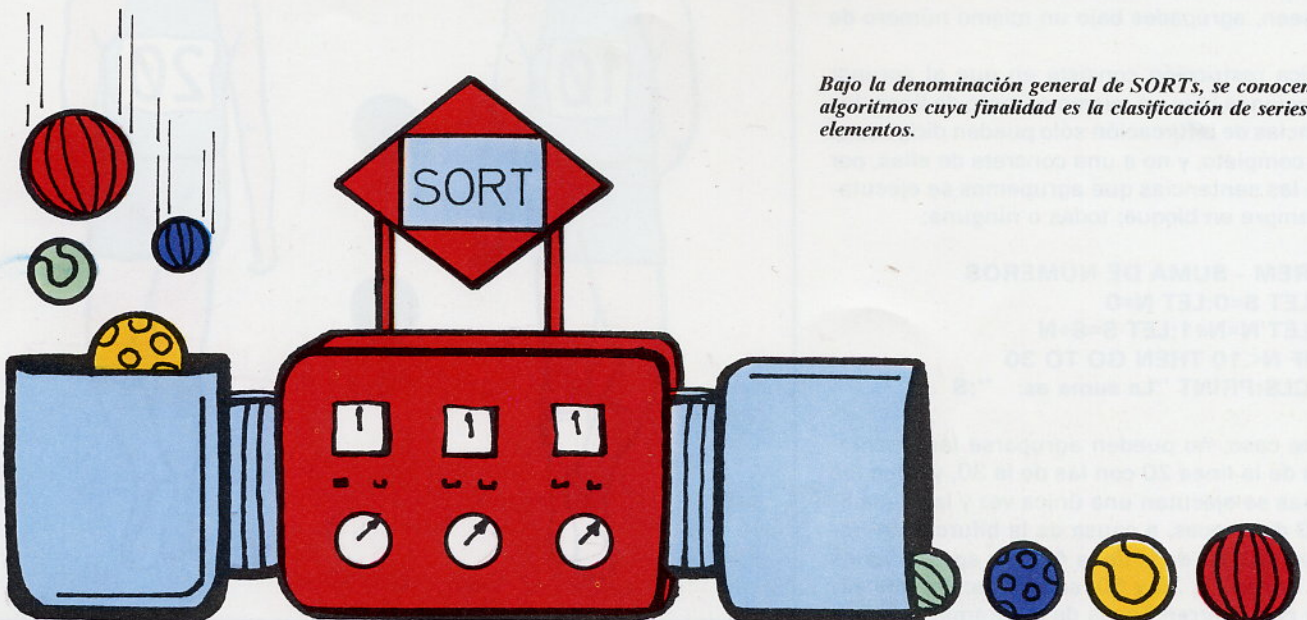
Concretamente, los dígitos del 0 al 9 tienen un código entre 48 y 57, las letras mayúsculas entre 65 y 90, las minúsculas entre 97 y 122, y los caracteres especiales, signos de puntuación y símbolos ocupan el resto de los códigos (del 0 al 47 y del 123 al 255). Así por ejemplo, la letra A (código 65) se considera menor que la B (código 66), etc...

Por tanto, podemos decir que el Spectrum lleva a cabo las clasificaciones por «orden alfabético», aunque, eso sí, según «su alfabeto». De todas formas, el código A.S.C.I.I. ha sido dispuesto de manera que coincida con el alfabeto humano en la zona dedicada a las letras. Pasemos ahora a analizar los diferentes sistemas de SORT que podemos utilizar:

* BINARIO o BURBUJA:

Quizá sea el más sencillo de todos los que vamos a tratar. Su peculiar denominación (BURBU-

Bajo la denominación general de SORTs, se conocen los algoritmos cuya finalidad es la clasificación de series de elementos.





JA) responde a la manera en que los datos van «desplazándose» por la lista, hasta ocupar su lugar correspondiente.

El método consiste en comparar el primer elemento de la serie con todos los que le siguen. Si éste es más pequeño, conserva su lugar; en caso contrario, cuando en una comparación encuentre un elemento menor, intercambian sus posiciones, siendo el primero, nuevamente, el menor de los dos. Una vez que el menor de todos los elementos se encuentra en primer lugar, repite el proceso comparando el segundo con los demás, luego el tercero y así sucesivamente, hasta completar el trabajo.

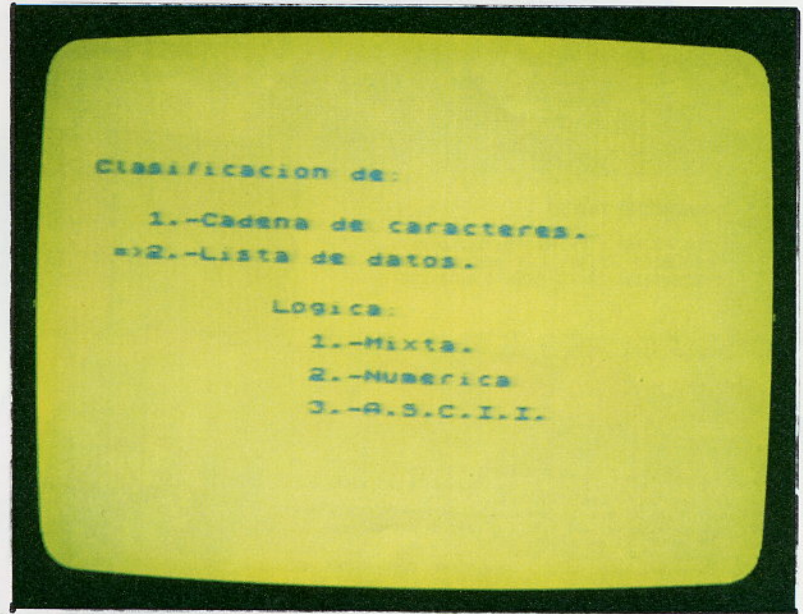
* BURBUJA CON MARCA o BUBBLE:

El método anteriormente explicado tiene un inconveniente muy importante. Supongamos que la lista a clasificar se encuentra al empezar totalmente ordenada; a pesar de ello, el sistema binario realizará todas las comparaciones, aunque no necesite efectuar ningún cambio. Esto lo convierte en un método especialmente lento, cuando se trata de organizar colecciones de datos parcialmente ordenadas.

El sistema de BURBUJA CON MARCA, compara cada elemento con el siguiente, en vez de con todos los de la lista. En el caso de que el siguiente sea mayor que su predecesor, se intercambian sus posiciones. De esta manera, el mayor va «cayendo» por la lista, hasta ocupar su lugar. Una marca dentro del SORT indica si se realizó la pasada sin llevar a cabo ningún intercambio, lo cual significa que cada elemento se encuentra en su lugar y la clasificación ha concluido.

* BUSQUEDA DE MAXIMO Y MINIMO

El funcionamiento de este SORT es también sencillo. El método consiste en seleccionar los elementos menor y mayor de la lista y colocarlos en primer y último lugar, respectivamente. A continuación, repetimos la operación para los datos comprendidos entre la segunda y la penúltima posición. Es decir, el bucle principal del clasificador, se reduce en dos elementos cada vez que realizamos una pasada.



Menú del programa ENTRADA para selección de lógica del SORT.

Para listas de datos de igual tamaño, independientemente de si están muy ordenadas o no, el número de pasadas será el mismo, puesto que este sistema no permite determinar si la lista se encuentra ya clasificada, a diferencia del método anterior.

* INTERCAMBIO RETARDADO

El siguiente SORT, es una variante del clasificador binario que, a costa de aumentar el número de instrucciones, consigue un menor tiempo de ejecución, siempre que el volumen de datos no sea muy elevado.

El adjetivo «retardado» responde a que, al efectuar una pasada por la lista, no se realizan intercambios de posición hasta que no hemos encontrado el menor elemento. Para ello, se utiliza un puntero, M en el ejemplo, que señala en cada momento hacia este elemento. Una vez determinado el menor, se coloca en primer lugar y se repite el proceso con los restantes elementos hasta finalizar la clasificación.



SHELL METZNER

```
100 FOR p=0 TO 16
110 LET k=2*p: IF k>num THEN LET l=INT ((2*p-1)/2)
: GO TO 150
120 NEXT p
200 LET a=num-1: LET b=1
210 LET m=b
220 LET c=m+1
230 IF a$(m)<a$(c) THEN GO TO 250
240 LET b=a$(m): LET a$(m)=a$(c): LET a$(c)=b$: LET
m=m-1: IF m=1 THEN GO TO 220
250 LET b=b+1: IF b=c THEN GO TO 210
260 LET l=INT (1/2): IF l THEN GO TO 200
```

CRONOMETRO INICIO

```
150 PRINT : PRINT "Calculando..."
160 BEEP .2,30: LET t1=(65536*PEEK 2367
4+256*PEEK 23673+PEEK 23672)/50
```

ENTRADA

```
1 REM CARLOS DE OSSA $ SORTS
10 LET l$="Lista ": LET c$="Cadena ": LET q$="a cla
sificar ": LET u$="clasificada:"
15 PRINT AT 4,0:"Clasificacion de:":AT 7,3:"1.-":c$
: "de caracteres.":AT 9,3:"2.-":l$:"de datos."
20 INPUT "Elige opcion.":o
25 PRINT AT 5+2*c,1:"="
30 IF o=1 THEN INPUT "Cadena?":a$: CLS : PRINT c$:
q$: PRINT : PRINT a$: LET num=LEN a$: LET str="1: GO T
O 100
35 PRINT AT 12,10:"Logica":AT 14,12:"1.-Mixa.":AT
16,12:"2.-Numerica":AT 18,12:"3.-A.S.C.I.I."
40 INPUT "Elige logica ":l
45 PRINT AT 12+2*l,c,10:"="
50 INPUT "entre el numero de datos :num
55 LET str="0: INPUT "Longitud maxima ":lg
60 LET r$="": FOR l=0 TO lg-2: LET r$=r$+CHR$(0): NE
XT l
65 DIM a$(num,lg)
70 CLS : PRINT l$:q$: PRINT : FOR l=1 TO num: INPUT
i$: PRINT i$: "
75 GO SUB 990+10*l
80 NEXT l
```

CAMBIO DE TABLA

```
100 IF str THEN DIM b$(num): GO TO 120
110 DIM b$(num,lg)
120 LET p=1: LET m$=CHR$(255)
200 LET e=1: LET i=1
210 IF i>num-1 THEN GO TO 240
220 IF a$(e)<a$(i+1) THEN LET i=i+1: G
O TO 210
230 LET e=i+1: LET i=i+1: GO TO 210
240 LET b$(p)=a$(e): LET a$(e)=m$: LET
p=p+1
250 IF p<num THEN GO TO 200
350 IF str THEN LET a$=b$: GO TO 400
360 FOR l=1 TO num: LET a$(l)=b$(l): NE
XT l
```

BUSQUEDA DE MAXIMO Y MINIMO

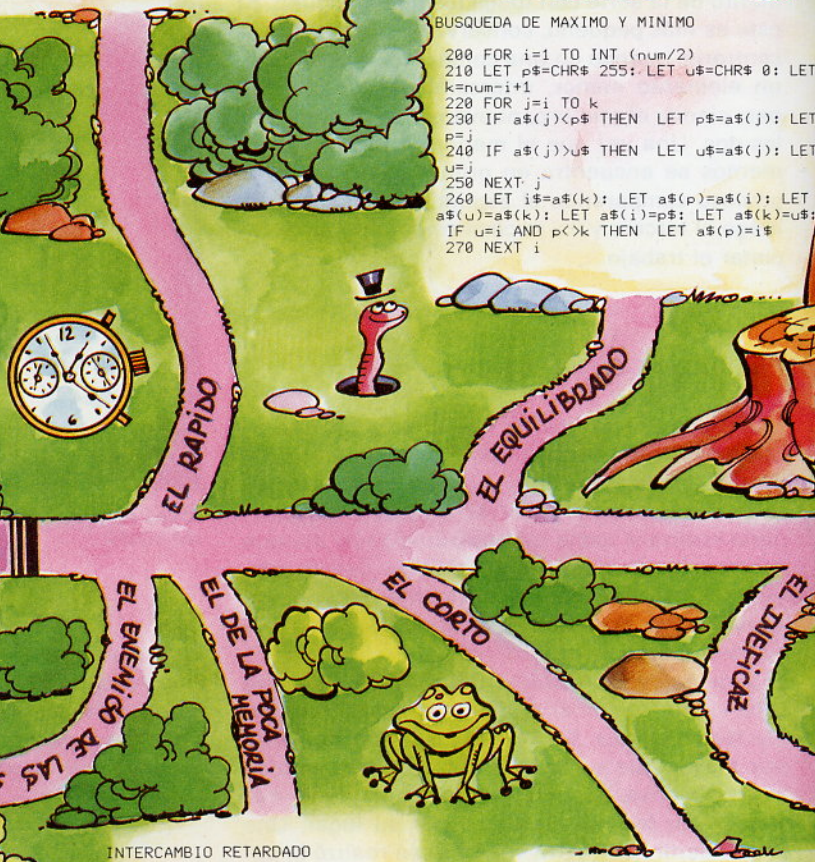
```
200 FOR i=1 TO INT (num/2)
210 LET p$=CHR$(255): LET u$=CHR$(0): LET
k=num-i+1
220 FOR j=i TO k
230 IF a$(j)<p$ THEN LET p$=a$(j): LET
p=j
240 IF a$(j)>u$ THEN LET u$=a$(j): LET
u=j
250 NEXT j
260 LET i$=a$(k): LET a$(p)=a$(i): LET
a$(u)=a$(k): LET a$(i)=p$: LET a$(k)=u$:
IF u=i AND p<k THEN LET a$(p)=i$
270 NEXT i
```

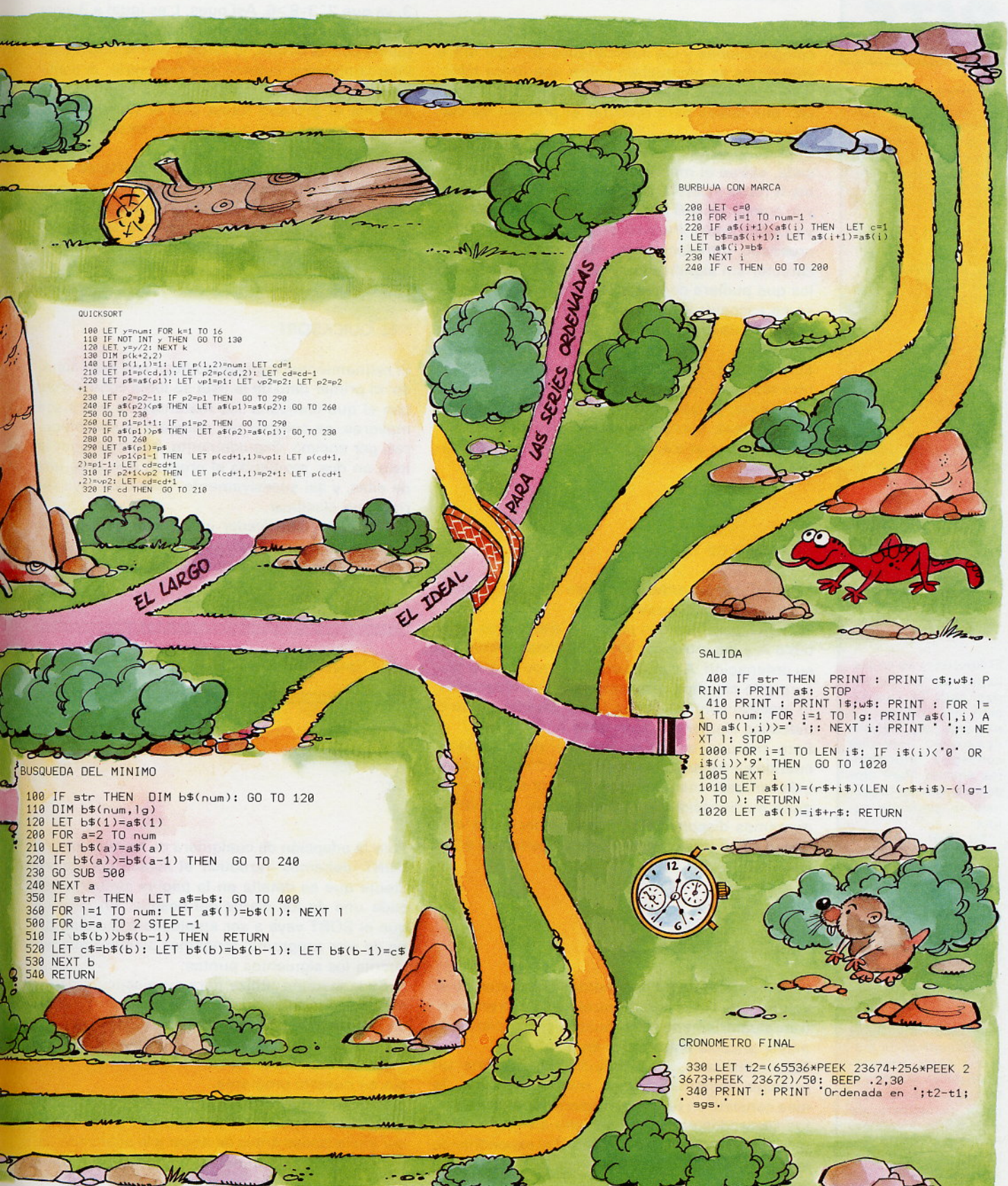
INTERCAMBIO RETARDADO

```
200 FOR i=1 TO num-1
210 LET m=i
220 FOR j=i+1 TO num: IF a$(j)<a$(
m) THEN LET m=j
230 NEXT j
240 IF m<i THEN LET b$=a$(i): L
ET a$(i)=a$(m): LET a$(m)=b$
250 NEXT i
```

BINARIO

```
200 FOR i=1 TO num-1
210 FOR j=i+1 TO num
220 IF a$(i)>a$(j) THEN LET b$=a$(
i): LET a$(i)=a$(j): LET a$(j)=b$
230 NEXT j: NEXT i
```





QUICKSORT

```

100 LET y=num: FOR k=1 TO 16
110 IF NOT INT y THEN GO TO 130
120 LET y=y/2: NEXT k
130 DIM p(k*2,2)
140 LET p(1,1)=1: LET p(1,2)=num: LET cd=1
210 LET p1=p(cd,1): LET p2=p(cd,2): LET cd=cd-1
220 LET p#=#(p1): LET vp1=p1: LET vp2=p2: LET p2=p2+1
230 LET p2=p2-1: IF p2=p1 THEN GO TO 290
240 IF a#(p2)<p# THEN LET a#(p1)=a#(p2): GO TO 260
250 GO TO 230
260 LET p1=p1+1: IF p1=p2 THEN GO TO 290
270 IF a#(p1)>p# THEN LET a#(p2)=a#(p1): GO TO 230
280 GO TO 260
290 LET a#(p1)=p#
300 IF vp1<p1 THEN LET p(cd+1,1)=vp1: LET p(cd+1,2)=p1-1: LET cd=cd+1
310 IF p2+1<vp2 THEN LET p(cd+1,1)=p2+1: LET p(cd+1,2)=vp2: LET cd=cd+1
320 IF cd THEN GO TO 210
    
```

BURBUJA CON MARCA

```

200 LET c=0
210 FOR i=1 TO num-1
220 IF a$(i+1)<a$(i) THEN LET c=1
: LET b#=#(i+1): LET a$(i+1)=a$(i)
: LET a$(i)=b#
230 NEXT i
240 IF c THEN GO TO 200
    
```

EL LARGO

EL IDEAL

PARA LAS SERIES ORDENADAS

BUSQUEDA DEL MINIMO

```

100 IF str THEN DIM b$(num): GO TO 120
110 DIM b$(num,lg)
120 LET b$(1)=a$(1)
200 FOR a=2 TO num
210 LET b$(a)=a$(a)
220 IF b$(a)>b$(a-1) THEN GO TO 240
230 GO SUB 500
240 NEXT a
350 IF str THEN LET a#=b#: GO TO 400
360 FOR l=1 TO num: LET a$(l)=b$(l): NEXT l
500 FOR b=a TO 2 STEP -1
510 IF b$(b)>b$(b-1) THEN RETURN
520 LET c#=b$(b): LET b$(b)=b$(b-1): LET b$(b-1)=c#
530 NEXT b
540 RETURN
    
```

SALIDA

```

400 IF str THEN PRINT : PRINT c#;w#: P
RINT : PRINT a#: STOP
410 PRINT : PRINT l#;w#: PRINT : FOR l=
1 TO num: FOR i=1 TO lg: PRINT a$(l,i) A
ND a$(l,i)>' ': NEXT i: PRINT ' ': NE
XT l: STOP
1000 FOR i=1 TO LEN i$: IF i$(i)<'0' OR
i$(i)>'9' THEN GO TO 1020
1005 NEXT i
1010 LET a$(l)=(r#+i$(LEN (r#+i$)-(lg-
1) TO )): RETURN
1020 LET a$(l)=i#+r$: RETURN
    
```



CRONOMETRO FINAL

```

330 LET t2=(65536*PEEK 23674+256*PEEK 2
3673+PEEK 23672)/50: BEEP .2,30
340 PRINT : PRINT 'Ordenada en ';t2-t1:
: sgs.
    
```

BITS

Ahí va un truco muy interesante: cómo hacer que las letras de un mensaje giren ante nuestros propios ojos.

```
10 CLS
20 FOR I=7 TO 0
STEP -1
30 POKE 23606, I
40 PRINT AT
0,0;"MENSAJE"
50 PAUSE 10
60 NEXT I
```



Para ciertas labores de cronometrado, nos vendrá muy bien disponer de una rutina que indique en minutos el tiempo que la máquina lleva trabajando:

```
PRINT INT
((65536*PEEK
23674+256*PEEK
23673+PEEK
23672)/3000)
```



El Spectrum no se distingue por su capacidad para la generación de sonidos, sin embargo, en algunas ocasiones podemos conseguir interesantes efectos. Para empezar a trabajar probemos la siguiente rutina:

```
10 FOR I=0 TO 9
20 FOR J=0 TO 39
30 BEEP .01,30
40 NEXT J
50 PAUSE 100
60 NEXT I
```

¿A que dan ganas de coger el teléfono?



En ocasiones el RAM TOP o tope de la memoria fijado por programación, puede ser tan bajo que no deje pasar ni un byte, ¡lo que se dice ni un byte! Para comprobarlo sólo hemos de introducir CLEAR 23821.

* BUSQUEDA DEL MINIMO

En todos los métodos vistos anteriormente, las comparaciones y los intercambios de lugar entre elementos se efectuaban sobre una única tabla. A continuación veremos dos sistemas que utilizan dos tablas para ejecutar el proceso de clasificación.

El primero de ellos, cada vez que realiza una pasada, extrae el menor de los elementos de la tabla A (tabla FUENTE con los datos desordenados) y lo coloca en la B (tabla DESTINO para los datos ya clasificados). Para recordar que éste se haya ordenado, introduce en su lugar de la tabla A un carácter sin significado, superior a cualquiera de los que pudiera contener la tabla. En el ejemplo, se ha escogido a tal fin un CHR\$ 255, pero podría ser cualquier otro que desempeñara la misma labor.

Este método, que tan solo se presenta aquí como otra técnica a tener en cuenta, se revela bastante ineficaz, ya que los tiempos empleados en clasificar listas, parcialmente ordenadas o no, son los más altos entre todos los sistemas que estamos repasando.

* CAMBIO DE TABLA

Al igual que el anterior, utiliza dos tablas para efectuar la clasificación.

El SORT toma los dos primeros elementos de la tabla A y los introduce en la B, donde procede a su ordenación. A continuación, añade en B el tercero y lo clasifica respecto a los dos anteriores. El proceso continúa hasta haber clasificado en B todos los elementos de la tabla A.

Este método es especialmente recomendable cuando se tratan listas bastante ordenadas. Además, en este caso, la tabla A no ha sufrido ninguna modificación, lo que puede ser muy útil si queremos disponer simultáneamente en la memoria de la colección de datos clasificada y sin clasificar.

* SHELL-METZNER

Finalmente, he aquí dos métodos que, cuando el volumen de datos a tratar es elevado y se encuentran aleatoriamente distribuidos, proporcionan la mayor eficacia entre todos los analizados.

Más sofisticado, el SHELL-METZNER tiene un principio matemático algo más complicado: la idea consiste en realizar comparaciones entre datos separados en la lista por una distancia constante $L = \text{INT}((2^P - 1) / 2)$; siendo P el exponente de la menor potencia de 2 que hace 2^P superior al número de elementos de la serie. Es decir, si por

ejemplo tenemos 6 palabras a clasificar, P sería 3, ya que $2^3 = 8 > 6$. Así pues, L es igual a 3 y por tanto comparamos la primera con la cuarta, la segunda con la quinta, la tercera con la sexta y las intercambiamos cuando sea necesario. A continuación, reducimos la distancia L a la mitad y repetimos el proceso hasta que la distancia sea nula, momento en que la lista estará ya clasificada.

Para utilizarlo, debemos pagar el precio de añadir algunas instrucciones más que en los métodos anteriores. Pero a cambio, el número de comparaciones se reduce sensiblemente, lo que proporciona un ahorro de tiempo considerable.

* EL QUICKSORT

El refinamiento clasificando llega con este método: el primer dato de la lista se toma como pivote con el que comparar los restantes. Si éstos son mayores, se colocan por debajo y si son menores, por encima. Seguidamente, fragmentamos la lista en dos, repitiendo el proceso con cada una de ellas, y así sucesivamente, mientras haya series para clasificar.

Se trata del clasificador que mayor cantidad de memoria necesita, pero su eficacia queda justificada en función del tiempo de proceso empleado.

LA FIGURA

Para la adopción de cualquiera de los métodos de SORT, debemos introducir el listado correspondiente que se detalla en la página siguiente, en cada uno de los claros del bosque. En caso de que el SORT vaya a ser empleado como subrutina de otro programa, tengamos siempre muy en cuenta los siguientes puntos:

- * Cada sistema utiliza para su ejecución determinadas variables que pueden tener otra misión en el programa principal; de ser así, podemos optar por seguir dos caminos: o cambiamos las variables afectadas en el SORT, o hacemos lo propio en nuestro programa.

- * Existen determinadas variables que los SORT utilizan como parámetros y que, por tanto, deben ser inicializadas antes de llegar a la subrutina; tales son: STR, que indica el tipo de serie a tratar (STR<>0, supone una cadena y STR=0 supone

un dimensionado alfanumérico). NUM, que es siempre el número de elementos de la serie y LG, que se emplea sólo en casos de dimensionado (STR=0) para indicar la segunda dimensión de la tabla (longitud de los elementos). Por supuesto, estas variables pueden ser cambiadas de nombre siempre y cuando el cambio sea realizado en toda la subrutina.

* Como soporte de la serie a clasificar se ha elegido la cadena o dimensionado A\$, y como puente en los métodos que precisan de dos tablas, B\$. Al igual que en los casos anteriores, estas variables pueden ser cambiadas de nombre, pero siempre respetando los cambios en toda la subrutina. Aunque en los ejemplos se utilizan series alfanuméricas, la adaptación de las subrutinas para que funcionen con tablas numéricas es extremadamente sencilla: prácticamente sólo es necesario escoger el sistema de clasificación de series de datos, eliminando el parámetro LG y los símbolos dólar (\$) que encontremos en la subrutina.

Como ya hemos dicho, la eficacia de cada SORT depende en su mayor parte de las características especiales de la serie a tratar (número de elementos y estado de desorden en que se encuentra). Para aquellos que quieran comprobar por sí mismos el funcionamiento de cada método, hemos dispuesto una interesante gira turística por el bosque de los SORT.

La etapa siempre comenzará en un mismo punto: la ENTRADA del bosque; una vez introducido este listado, podremos continuar nuestra andadura hasta el control de tiempo. Si deseamos conocer el tiempo en que el SORT va a clasificar la serie de datos, deberemos poner en marcha el cronómetro introduciendo las dos líneas que se señalan en este punto del recorrido. Siguiendo siempre los caminos rosas, podremos acceder al SORT que más nos convenga, y una vez introducido el correspondiente listado, abandonaremos el claro del bosque mediante el camino amarillo de salida.

Este tipo de caminos nos llevarán siempre al control de fin de cronómetro, cuyas líneas deberemos introducir si deseamos conocer el tiempo de ejecución del SORT escogido. Finalmente, el programa SALIDA, marcará el fin de nuestro recorrido y gracias a él se mostrará en la pantalla la serie ordenada.

Al llegar a la salida del bosque, tendremos en la memoria de nuestro aparato un programa que podremos ejecutar para comprobar la eficacia del SORT escogido. Sólo queda aclarar las preguntas que el programa ENTRADA nos efectuará para realizar la inicialización del sub-programa de clasificación.

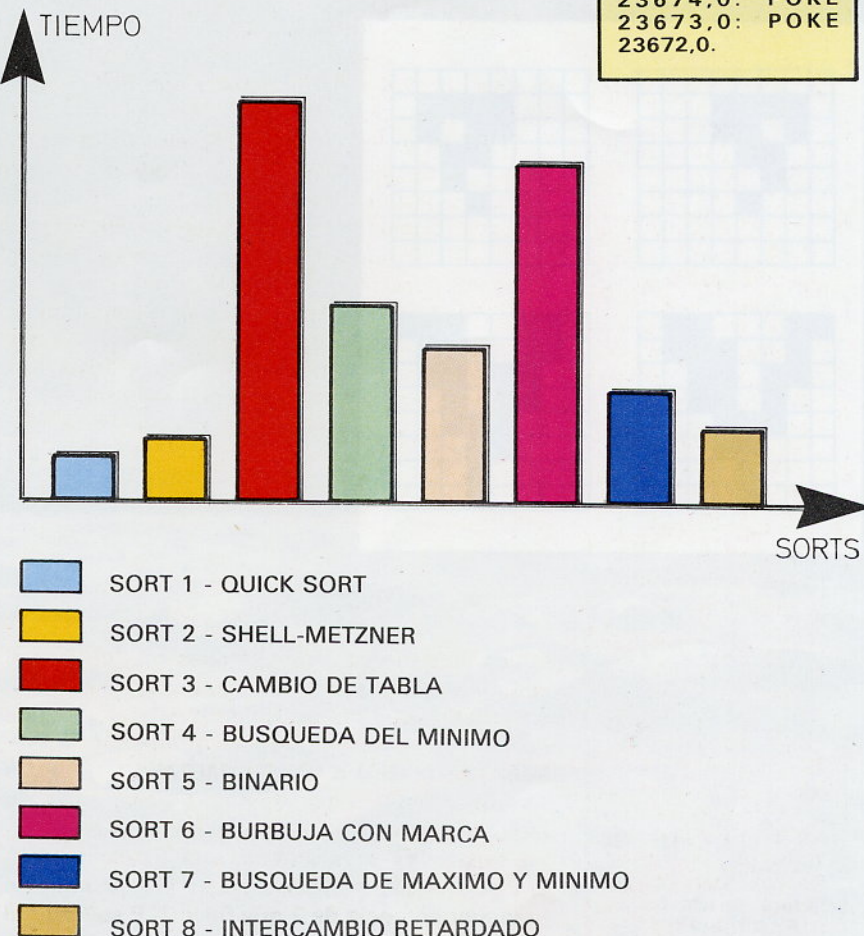
En primer lugar, en la ENTRADA se nos interrogará sobre el tipo de serie a clasificar: Cadena de caracteres (cadena) o cadena de datos (dimensionado). Si se escoge el primer punto, el programa

nos facilitará la introducción de la cadena a clasificar y comenzará su desenfadada carrera por el itinerario del bosque que hayamos escogido. Cuando se selecciona el punto segundo (cadena de datos), el programa, además de interrogar sobre dos cuestiones bien necesarias: número de datos y longitud de los mismos, presentará un nuevo menú de tres opciones, bajo el título de «Lógica». Si se selecciona el tercer punto (lógica A.S.C.I.I.), el programa elegirá como criterio de clasificación de la serie, la posición de los caracteres en el código A.S.C.I.I.

Si se selecciona la lógica numérica (punto segundo), los datos de la serie, aunque introducidos como cadenas, serán numéricos y por tanto serán clasificados como tales. Si por el contrario, el punto escogido es el primero (lógica mixta), los datos que componen la serie podrán ser tanto numéricos como textos, por lo que se realizará la clasificación pertinente para cada uno de los casos: los numéricos como tales y los textos como cadenas de caracteres.



Gráfico de tiempos medios de los diversos SORTs, tomados en la clasificación de series aleatorias de 100 elementos.



BITS

Es curioso lo que ocurre cuando llevamos nuestro ordenador a situaciones extremas, ¡hasta le bailan los caracteres! Probemos por ejemplo a forzar el **RAM TOP** a su punto más alto, es decir, **CLEAR 65535** en el modelo de 48 K. o **CLEAR 32767** en el de 16 K. Si a continuación escribimos los caracteres gráficos entre la L y la U, los veremos bailar al son de cualquier tecla.



Cuando queramos poner a cero el cronómetro, sólo tendremos que ejecutar: **POKE 23674,0: POKE 23673,0: POKE 23672,0.**



MODULO LUNAR



TRANSCURRE el año 2020; la conquista del espacio es una realidad y decenas de estaciones espaciales circunvalan la Tierra. Hace un decenio se construyó la primera base lunar de utilización conjunta internacional, LUNA-1.

La función de LUNA-1 es purificar el plutonio que se extrae de las minas colindantes. Dicho mineral ha de ser enviado a otra base para su posterior proceso de elaboración; este transporte se efectúa a bordo de un módulo que tiene que sobrevolar las afiladas aristas de las cordilleras lunares.

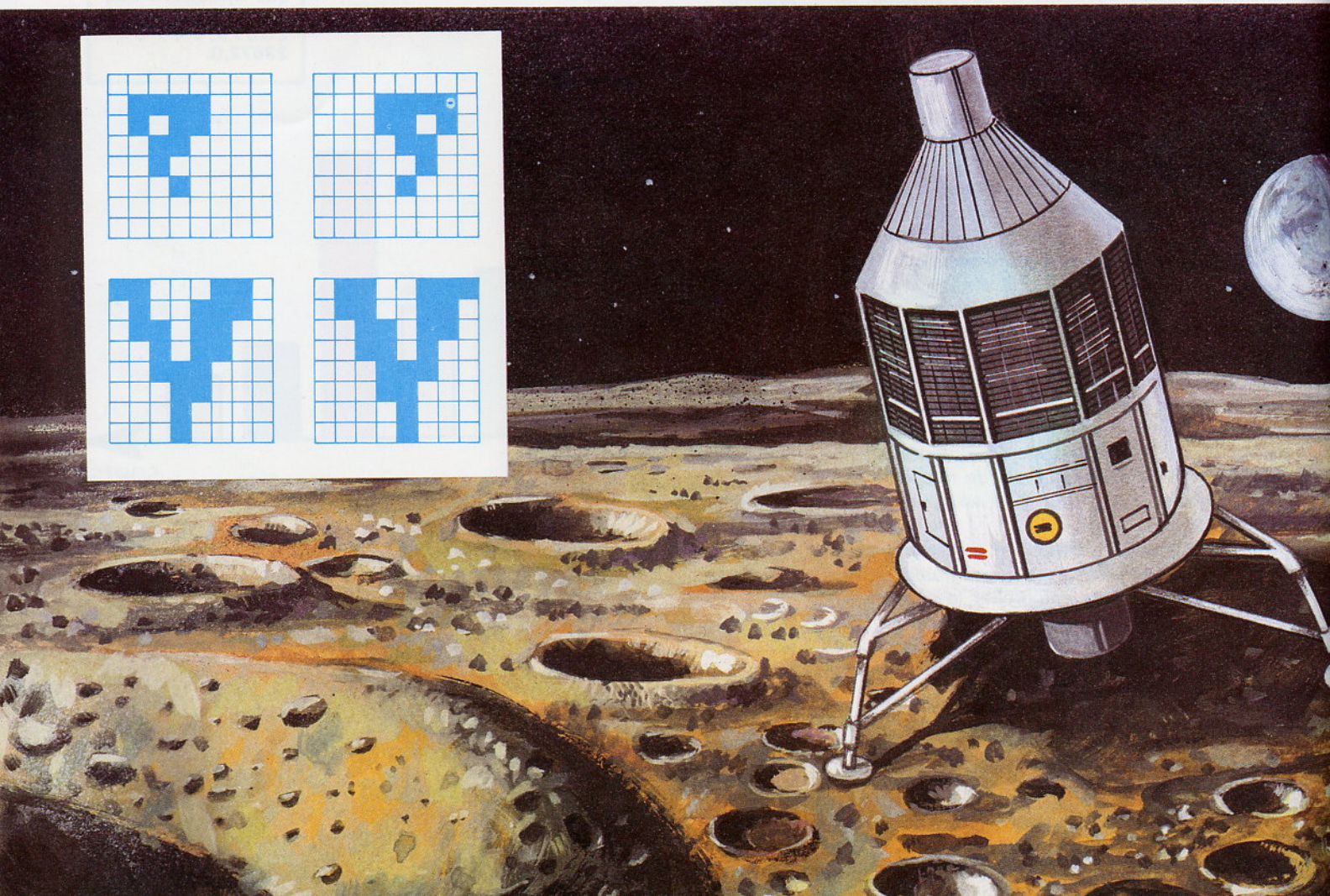
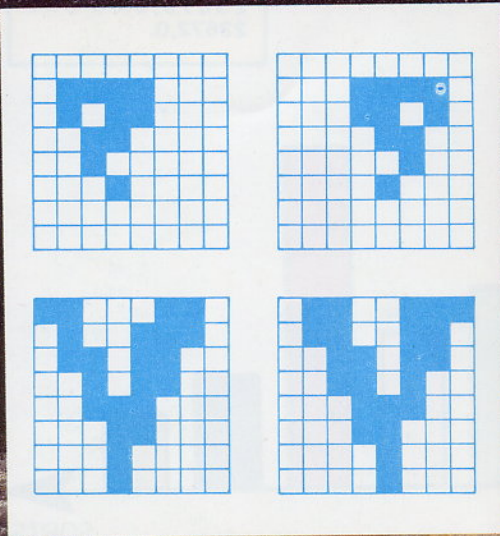
Nosotros somos los encargados de pilotar el módulo lunar y posarnos, lo más suavemente posible, en la plataforma de destino, evitando destruir la nave contra las montañas. Para el control del módulo disponemos tan solo de dos mandos:

uno de ellos la hace ascender en vertical y el otro, descender en diagonal izquierda. Hay que tener en cuenta que la caída inercial de la nave cuando no le brindamos empuje, es en diagonal derecha.

Este tipo de desplazamientos de descenso, un tanto extraños, pueden atribuirse a poderosísimas razones, tales como el desplazamiento de las cargas de plutonio por el interior del módulo, alteraciones en los sistemas giroscópicos o perturbaciones de campos magnéticos. Ahora bien, lo cierto y verdad, es que obedecen a un único y transparente objetivo: complicar la tarea del alunizaje.

Así pues, los controles de la nave responden a las siguientes teclas: P => ARRIBA y L => ABAJO/IZQUIERDA. Al comienzo de cada intento de alunizaje, seremos interrogados acerca del nivel

Gráficos para los fogonazos de propulsión.






de dificultad con que deseamos operar (de 0 a 8). De este parámetro dependerá lo escarpado del perfil lunar que a continuación se presente.

La arquitectura del programa es completamente estructurada. El bloque de programa (PROGRAMA PRINCIPAL) que controla todas y cada una de las situaciones (SUBROUTINAS) de la nave es muy simple y estricto. Como punto a destacar en el programa, podemos hablar de la animación conseguida mediante superposición de gráficos definidos.

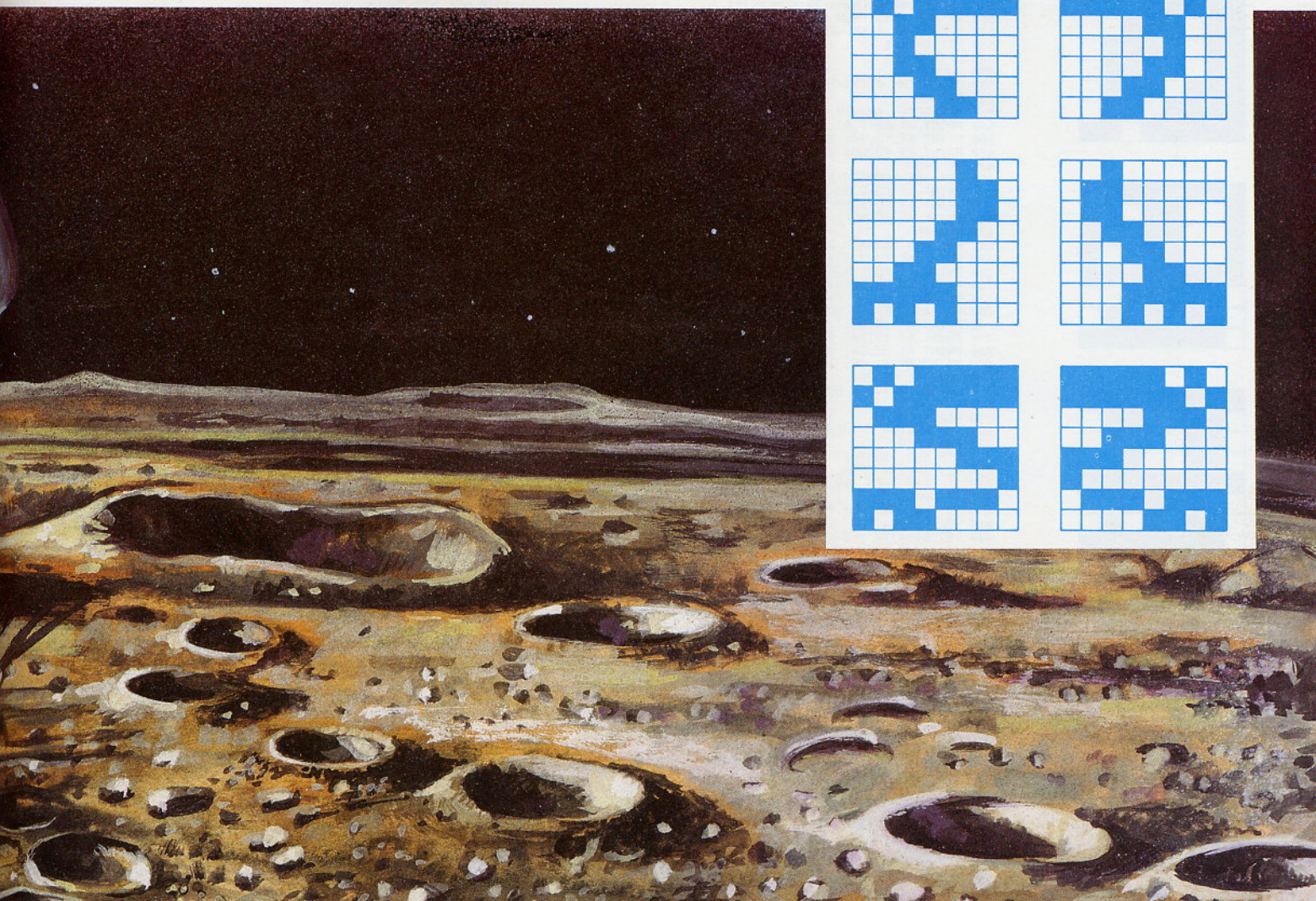
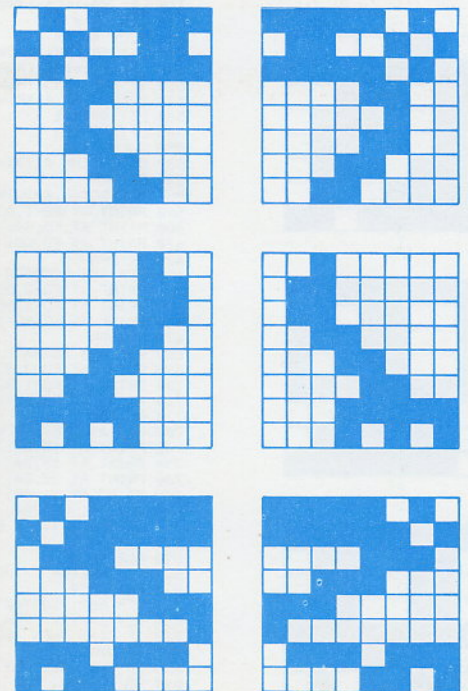
El programa tiene definidos dieciséis gráficos de usuario, de los cuales los ubicados en los caracteres CD y QR son los que simulan la rotación del anillo central de nuestro módulo lunar, y los definidos en los caracteres GH e IJ asemejan la amortiguación de las plataformas de alunizaje. El resto de los gráficos de usuario definen: zona su-

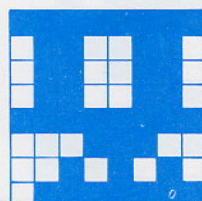
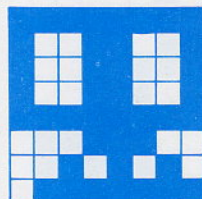
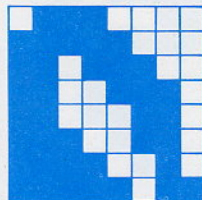
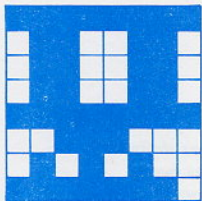
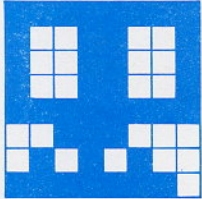
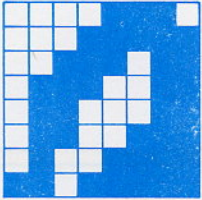
perior de las plataformas de despegue y alunizaje (gráficos en EF), zona superior de la nave (gráficos en AB), llamas de propulsión (gráficos en KL y MN).

Aunque ya es costumbre, no está de más advertir, que en el listado, por motivos de comodidad a la hora de la introducción, los gráficos definidos han sido sustituidos por las letras subrayadas de las teclas mediante las cuales se obtienen. Así por ejemplo, una A deberá ser introducida como un carácter gráfico de la A, es decir: **GRAPHICS A GRAPHICS**.

Para la adopción del programa procederemos por el sistema habitual: **SAVE "MODULO"** o bien **SAVE "MODULO" LINE 10** (si deseamos que el programa se autoejecute al cargarlo). 

Gráficos definidos para las bases de despegue y alunizaje.





Gráficos para la animación del módulo lunar.

```

10 REM *****
20 REM * J.M.MAYORAL SERRANO *
30 REM *****
40 POKE 23658,8
50 GO SUB 1910: GO TO 1570
60 LET TO=0: LET FD=20: LET FD1=20: LET SW=0
70 REM POSIC. ALEAT. BASES
80 LET BD=INT (RND*13)+1
90 LET BD1=BD
100 LET BA=INT (RND*14)+1
110 LET BA1=BA
120 RETURN
130 REM POSIC. B. DESPEGUE.
140 PRINT AT 21,BD; INK 5; 'IJ'
150 RETURN
160 REM POSIC. B.ALUNIZ.
170 PRINT AT 20,BA; INK 5; 'EF'
180 PRINT AT 21,BA; INK 5; 'GH'
190 RETURN
200 REM DIBUJO PAISAJE.
210 LET XP=(BD*8)-3
220 LET YP=0
230 PLOT XP,YP
240 LET XP1=XP+((-5*RND)+47)
250 LET YP1=INT (RND*(50+10*ND))+1
260 IF XP1<0 THEN GO TO 310
270 DRAW INK 4;XP1-XP,YP1-YP
280 LET XP=XP1
290 LET YP=YP1
300 GO TO 240
310 IF XP1=0 THEN GO TO 320
320 DRAW INK 4;-XP,INT (RND*130)-YP
330 REM
340 LET XP=((BD+1)*8)+10
350 LET YP=0
360 PLOT XP,YP
370 LET XP1=XP+((-5*RND)+47)
380 LET YP1=INT (RND*(50+ND*10))+1
390 IF XP1>BA*8-3 THEN GO TO 440
400 DRAW INK 4;XP1-XP,YP1-YP
410 LET XP=XP1
420 LET YP=YP1
430 GO TO 370
440 DRAW INK 4;(BA*8-3)-XP,-YP
450 REM
460 LET XP=((BA+1)*8)+10
470 LET YP=0
480 PLOT XP,YP
490 LET XP1=XP+((-5*RND)+47)
500 LET YP1=INT (RND*(50+ND*10))+1
510 IF XP1>255 THEN GO TO 560
520 DRAW INK 4;XP1-XP,YP1-YP
530 LET XP=XP1
540 LET YP=YP1
550 GO TO 490
560 DRAW INK 4;255-XP,134-YP
570 RETURN
580 REM ROTAC.&MOVIM.NAVE
590 PRINT AT FD-1,BD; INK 6; 'AB'
600 PRINT AT FD,BD; INK 6; 'CD'
610 PRINT AT FD,BD; INK 6; 'QR'
620 RETURN
630 REM SBR.BORRADO 1 -NAVE
640 PRINT AT FD-2,BD-1;
650 PRINT AT FD-1,BD-1;
660 PRINT AT FD,BD-1;
670 RETURN
680 REM SBR.BORRADO 2- NAVE
690 PRINT AT FD-2,BD;
700 PRINT AT FD-1,BD+2;
710 PRINT AT FD,BD+2;
720 RETURN
730 REM ATERRIZAJE NAVE
740 PRINT AT 18,BA;
750 PRINT AT 19,BA; INK 5; 'AB'
760 PRINT AT 20,BA; INK 5; 'CD'
770 PRINT AT 21,BA; INK 6; 'QR'
780 FOR N=1 TO 200
790 BORDER 6::: BORDER 5::: BORDER 4:::
800 NEXT N
810 BORDER 0
820 IF INKEY$="" THEN GO TO 820
830 RUN 1570
840 REM SBR. TORTAZO
850 LET C=ATTR (FD+1,BD+1)
860 IF C<=5 THEN GO TO 880
870 RETURN
880 GO SUB 640: LET TO=1
890 GO TO 950
900 LET C1=ATTR (FD+1,BD)
910 IF C1<=5 THEN GO TO 930
920 RETURN
930 GO SUB 690: LET TO=1
940 REM EXPLOSION
950 DIM D(4): LET FD1=FD: LET BD1=BD
960 LET FD2=FD-1: LET BD2=BD
970 LET FD3=FD-1: LET BD3=BD+1
980 LET FD4=FD: LET BD4=BD+1
990 PRINT AT FD1,BD1; : LET D(1)=1
1000 PRINT AT FD2,BD2; : LET D(2)=1
1010 PRINT AT FD3,BD3; : LET D(3)=1
1020 PRINT AT FD4,BD4; : LET D(4)=1
1030 IF FD1>20 OR BD1<1 THEN LET D(1)=0
1040 IF FD2<1 OR BD2<1 THEN LET D(2)=0
1050 IF FD3<1 OR BD2>30 THEN LET D(3)=0
1060 IF FD4>20 OR BD4>30 THEN LET D(4)=0
1070 IF D(1)=0 OR D(2)=0 OR D(3)=0 OR D(4)=0 THEN BE
EP .5,30: BEEP .5,10: GO TO 820
1080 IF D(1)=1 THEN BEEP .05,20: LET FD1=FD1+1: LET
BD1=BD1-1: PRINT AT FD1,BD1; INK 6; 'C': GO TO 1090
1090 IF D(2)=1 THEN BEEP .05,20: LET FD2=FD2-1: LET
BD2=BD2-1: PRINT AT FD2,BD2; INK 6; 'A': GO TO 1100
1100 IF D(3)=1 THEN BEEP .05,20: LET FD3=FD3-1: LET
BD3=BD3+1: PRINT AT FD3,BD3; INK 6; 'B': PAUSE 2: GO T
O 1110
1110 IF D(4)=1 THEN BEEP .05,20: LET FD4=FD4+1: LET
BD4=BD4+1: PRINT AT FD4,BD4; INK 6; 'D': PAUSE 2: GO S
UB 1120: GO TO 1030
1120 PRINT AT FD1,BD1; :
1130 PRINT AT FD2,BD2; :
1140 PRINT AT FD3,BD3; :
1150 PRINT AT FD4,BD4; :
1160 RETURN
1170 REM ARRIBA NAVE
1180 IF FD=1 THEN LET FD=2
1190 LET FD=FD-1
1200 GO SUB 590
1210 PRINT AT FD+1,BD; INK 6; 'KL'; AT FD+1,BD; INK 2;
'MN'
1220 IF FD<=18 THEN PRINT AT FD+2,BD;
1230 IF FD=19 OR FD=18 THEN GO TO 1250
1240 RETURN
1250 PRINT AT FD1,BD1; INK 2; 'EF'
1260 PRINT AT FD1+1,BD1; INK 2; 'GH'
1270 RETURN
1280 REM ABAJO IZQUIERDA
1290 LET FD=FD+1
1300 LET BD=BD-1: IF TO=1 THEN RETURN
1310 IF BD<=0 THEN LET BD=0
1320 GO SUB 690: GO SUB 590
1330 IF BD=BA AND FD=19 THEN GO SUB 690: GO SUB 740
1340 GO SUB 900
1350 GO SUB 690
1360 RETURN
1370 REM ABAJO DERECHA
1380 LET FD=FD+1
1390 LET BD=BD+1: IF TO=1 THEN RETURN
1400 IF BD<=31 THEN LET BD=30
1410 GO SUB 640: GO SUB 590
1420 IF BD=BA AND FD=19 THEN GO SUB 640: GO SUB 740
1430 GO SUB 850
1440 GO SUB 640
1450 RETURN
1460 REM NIVEL DIFICULTAD
1470 BORDER 0: PAPER 0: INK 9
1480 CLS
1490 PRINT PAPER 6; BRIGHT 1; AT 13,3; ' NIVEL DE DIF
ICULTAD
1500 PRINT AT 15,11; '0 - 8'
1510 LET N$=INKEY$
1520 IF CODE N$<48 OR CODE N$>56 THEN GO TO 1510
1530 BEEP .2,20
1540 LET ND=VAL N$
1550 CLS
1560 RETURN
1570 REM
1580 REM PROGRAMA PRINCIPAL.
1590 REM
1600 GO SUB 1470
1610 GO SUB 60
1620 GO SUB 140
1630 GO SUB 170
1640 GO SUB 210
1650 LET K$=INKEY$
1660 GO SUB 590
1670 IF K$='P' THEN LET SW=1: GO SUB 1180
1680 GO SUB 590
1690 IF K$='L' AND SW=1 THEN GO SUB 1290
1700 GO SUB 590
1710 IF K$='.' AND SW=1 THEN GO SUB 1380
1720 GO SUB 590
1730 GO TO 1650
1740 REM GRAFICOS USUARIO
1750 DATA 95,166,95,48,32,48,24,12
1760 DATA 250,101,250,12,4,12,24,48
1770 DATA 4,6,6,12,24,48,248,168
1780 DATA 32,96,96,48,24,12,31,21
1790 DATA 95,191,112,28,7,1,238,176
1800 DATA 250,253,14,56,224,128,119,13
1810 DATA 14,31,59,115,99,103,79,208
1820 DATA 112,248,220,206,198,230,242,11
1830 DATA 255,153,153,153,255,56,84,254
1840 DATA 255,153,153,153,255,28,42,127
1850 DATA 0,128,88,48,32,16,0,0
1860 DATA 0,30,26,12,4,8,0,0
1870 DATA 0,102,108,44,56,24,16,16
1880 DATA 0,114,54,52,28,24,8,8
1890 DATA 255,102,102,102,255,56,84,254
1900 DATA 255,102,102,102,255,28,20,127
1910 LET G$='EFGHIJABCDKLMNQR'
1920 RESTORE 1750
1930 GO SUB 2030
1940 FOR M=1 TO LEN G$
1950 FOR N=0 TO 7
1960 READ D
1970 POKE USR G$(M)+N,D
1980 BEEP .01,30
1990 NEXT N
2000 BEEP .1,10
2010 NEXT M
2020 RETURN
2030 PRINT FLASH 1; INK 1; PAPER 6; AT 10,3;
2040 PRINT FLASH 1; PAPER 6; INK 1; AT 11,3; ' : PAP
ER 1; INK 6; ' UN MOMENTO POR FAVOR '; INK 1; PAPER 6;
2050 PRINT FLASH 1; INK 1; PAPER 6; AT 12,3;
2060 PRINT PAPER 2; INK 7; AT 14,6; ' ~GRAFICOS USUARI
O'
2070 RETURN

```

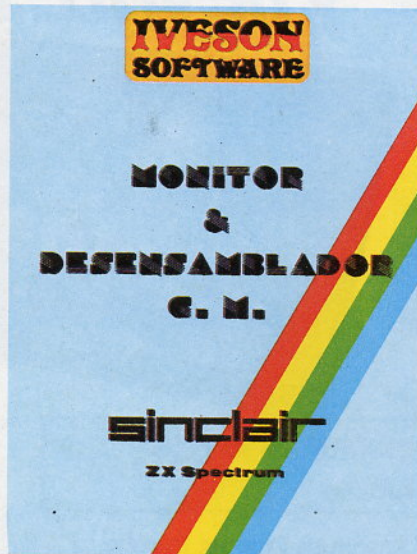
- Garantizamos nuestros programas por 5 meses.
- Condiciones Especiales para Comercios.

- Traducidos al castellano.
- Todos nuestros programas son originales.



Ref. 1007 P.V.R. 2.500 ptas.
PROCESADOR DE TEXTOS

Permite redactar cartas y documentos en su Spectrum con 64 caracteres por línea, búsquedas, inserciones, márgenes, sustitución de palabras.
Adaptado para todo tipo de interfaces, centronics y RS 232.
Caracteres españoles y letras acentuadas.



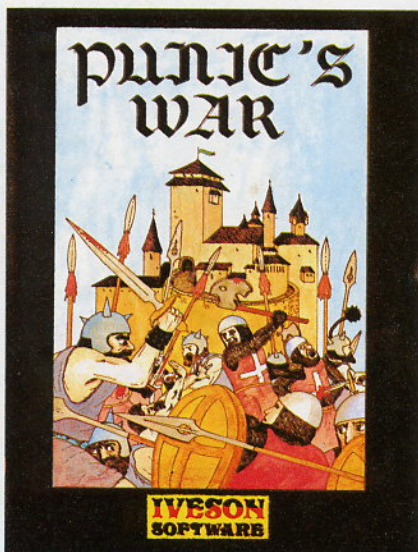
Ref. 1008 P.V.R. 3.000 ptas.
MONITOR & DESENSAMBLADOR

El mejor programa para introducir y desensamblar programas en código máquina.
Permite introducir cadenas, mover bloques de programa, verificarlos, ejecutarlos, corregirlos, ver los registros, listarlos, desensamblados, por caracteres, tabulados, además de otras muchas opciones.



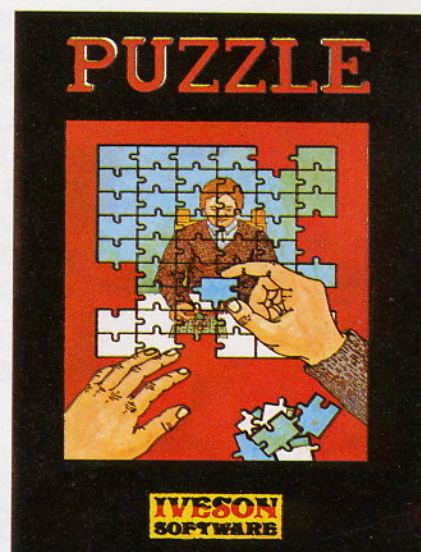
Ref. 1009 P.V.R. 1.900 ptas.
100 RUTINAS CM

Rutinas en código máquina de todo tipo para incluirlas en sus programas.
Scroll de todo tipo (por tercios de pantalla, por pantalla) en todas las direcciones, nuevos tipos de letra, protecciones, reenumeradores, conversores, Hex-Dec. Dec-Hex, y un largo etc. de posibilidades.



Ref. 1010 P.V.R. 1.900 ptas.
PUNIC'S WAR

Juego de estrategia tipo ward game. Deberá enfrentarse a Imperios Enemigos, Guerras Civiles, Bárbaros, Plagas, Fuertes, Puertos, etc.
5 niveles de juego más uno para expertos.
Sólo con su inteligencia y estrategia podrá llegar a conquistar Europa.



Ref. 1011 P.V.R. 1.600 ptas.
PUZZLE

El clásico juego llevado a su ordenador, apto para todas las edades.
5 tipos de fichas y 2 niveles dificultad gráficos excelentes.