TRATAMIENTO DE FICHEROS EN DISCO

F. L. Frontau

Una de las mayores ventajas, por no decir la mayor, del uso de unidades de disco, es la velocidad de acceso a los datos almacenados en él. Esto se pone de manifiesto, sobre todo, en el tratamiento de ficheros.

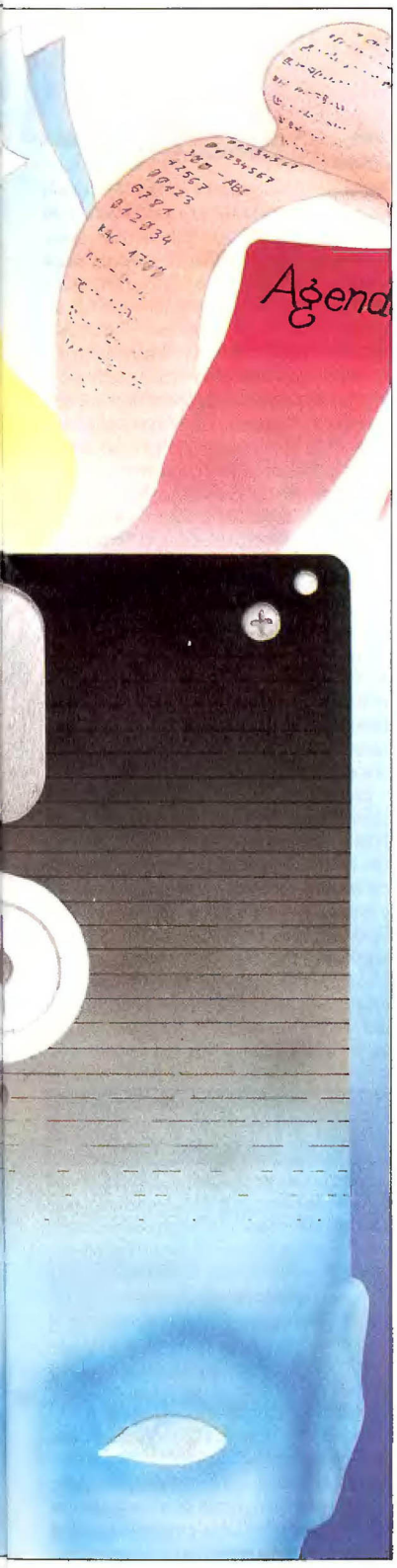
Javier Barceló

Hay tres maneras de acceder a un fichero: secuencialmente, por índices y directamente. De ahí el nombre de ficheros secuenciales, indexados y aleatorios o de acceso directo. De estos tres, los ficheros indexados son los únicos que no podemos utilizar en el AMSTRAD por no estar preprarados para ellos. En el **AMSTRAD** CPC 464 sin unidad de disco, sólo se pueden utilizar ficheros secuenciales, mientras que en el CPC 664 y CPC 6128 así como en el CPC 464 con unidad de disco, no tenemos por qué limitarnos a usar el acceso secuencial y podemos optar por el acceso directo. Pero **¡CUIDADO!** No nos engañemos. No siempre éste es más útil que aquél. Veremos que un programa que gestione ficheros de acceso directo ofrece más posibilidades, pero a cambio resulta más complejo que si maneja ficheros secuenciales. Seamos pues prácticos y veamos las posibilidades de cada uno, las similitudes y las diferencias entre ambos, y elijamos el que mejor se acople a nuestras necesidades.

Ficheros, registros y campos

A lo largo del artículo, estas tres palabras van a salir muchas veces, por lo que es esencial conocer bien su significado. Al hacer un programa, se nos puede plantear la necesidad de almacenar una serie de datos, de manera más o menos permanente para poder consultarlos o realizar otras operaciones posteriormente. Si los datos son pocos, y no va a haber necesidad de aumentarlos ni de modificarlos, se pueden poner en sentencia DATA, pero si por el contrario son muchos los datos o son datos que hay que modificar con cierta frecuencia, habrá que crear un FICHERO. Esto significa crear un archivo, independiente del programa, donde los tenemos almacenados, y el programa es una herramienta que nos permite modificarlos, consultarlos, listarlos o cualquier otra cosa que necesitemos. A cada ficha que utilicemos la llamamos REGISTRO. Una agenda telefónica, con nombre, apellidos, dirección y teléfono, es un fichero, y el conjunto de todos los datos de una persona en este fichero compone un registro del fichero. Un registro no es más que un conjunto de campos: Nombre, dirección y teléfono son los 3 campos que componen el registro del ejemplo de la agenda telefónica. Pongamos otro ejemplo, un archivo de biblioteca. Las fichas de todos los libros formarían un fichero. Cada ficha de un libro por separado, sería un registro de nuestro fichero. Y cada apartado de esa ficha *(nombre del autor, título, referencia, editorial...)* sería un campo del registro. Ahora veremos las diferentes maneras de crearlos, modificarlos y consularlos que nos permite el BASIC de **AMSTRAD.**

Pero antes, aclaremos otra cosa. Observaréis que los comandos de lectura y escritura de ficheros secuenciales, van seguidos de la expresión —#9—. Esto es así porque el **AMSTRAD** maneja 10 canales, que son como **«vías de comunicación»**. Las 8 primeras —del #0 al #7— se dirigen a la pantalla, la novena —#8— va la impresora, y la décima —#9— que es la que nos ocupa, se encarga de la comunicación del ordenador con el disco. Por eso, los comandos que leen y escriben datos en el disco van seguidos por la expresión —#9—. Una vez aclarada la teoría, pasemos a la práctica.



Ficheros de acceso secuencial

Empecemos por ver qué es un fichero de acceso secuencial. Su origen está ligado al método de almacenamiento de información más simple, la cinta magnética, el cassette. En este medio, los registros son grabados uno a continuación de otro, a medida que los vamos introduciendo. La primera limitación con la que nos encontramos, es que para acceder a un registro determinado, —a una ficha determinada—, tenemos que leer todas las anteriores, lo que supone una pérdida de tiempo lógicamente mayor cuanto mayor sea el archivo. Evidentemente, si una vez leído un registro queremos leer otra vez el anterior a el, no podemos retroceder y tenemos que volver a empezar. Además, el BASIC de nuestro **AMSTRAD** no dispone de ninguna instrucción que nos permita abrir un fichero ya creado para añadirle más datos o para modificar alguno intermedio. Pero tranquilos, veremos una manera de solucionar esto más adelante. Como no todo van a ser defectos, vayamos a la característica más interesante —junto a la sencillez— de estos ficheros. La longitud de los campos y de los registros del fichero no tiene que ser igual en todos ellos. Es decir, que el primer registro puede tener una longitud de 30 caracteres, el segundo de 15, y así todos. Ya veremos que en un fichero de acceso directo esto no es posible, obligándonos a dar a todos los registros la longitud del registro más largo, lo que desaprovecha espacio en el disco.

Para empezar a practicar, veámos los pasos a seguir para disponer de nuestro propio fichero.

El primer paso, es crearlo. Vamos a ir viendo los primeros comandos del BASIC, que son los que nos lo permitirán:

— OPENOUT **«nombre del fichero».** Abre el fichero, sólo para que podamos escribir en él.

— CLOSEOUT. Cierra el fichero anterior. No hay que poner el nombre del fichero.

— WRITE #9,a$. Escribe el contenido de a$, y todas las variables que pongamos después, separadas por comas, en el fichero.

Llega el momento de echarle un vistazo al Ejemplo 1. Vamos a ir paso por paso. En la línea 20, abrimos el fichero llamado **«AGENDA»**. En las líneas 30 a 50, damos a las variables NOM$,DIR$, y TEL, los datos que queremos escribir en el fichero. Y en la línea 60 pasamos estos datos al fichero. Aquí hay que señalar una cosa muy importante: el ordenador no escribe los datos en el momento en que ejecuta el comando WRITE #9 sino que los pasa a una zona de memoria intermedia, llamada BUFFER, y cuando esta zona se llena, o bien cuando cerramos el fichero, es cuando sí escribe los datos en el disco. La importancia de esto es vital, si nos olvidamos de cerrar el fichero, paramos el programa, o sacamos el disco ANTES de cerrar el fichero, nos arriesgamos a perder la información que en este momento está en el BUFFER. Las líneas 70 y 80 nos dan la posibilidad de seguir introduciendo datos, y la 90 cierra el fichero.

Bien. Supongo que ya tenemos nuestro fichero «AGENDA» en el disco con algún que otro dato. Pasemos al ejemplo 2 y observemos cómo leer los datos almacenados en él. Demos primero un repaso a los comandos BASIC para este cometido.

— OPENIN **«FICHERO».** Abre el fichero para lectura.

— INPUT #9,A$. Lee del fichero tantos datos como variables pongamos, siempre separadas por comas.

— CLOSEIN. Cierra el fichero anterior.

— EOF. Nos indica si hemos llegado al final del fichero, o si no ha sido abierto.

Echemos un vistazo al ejemplo 2, para explicar lo que hace. En la línea 20 abre el fichero «AGENDA» para lectura. Luego inicia un bucle WHILE-WEND, utilizando EOF. Paremos aquí un instante. Cuando en el ejemplo 1 hemos cerrado el fichero, —CLOSEOUT—, el sistema operativo que gestiona el «diálogo» entre ordenador y disco, graba una señal. EOF es una función que comprueba cada carácter que lee disco y devuelve el valor (—1) si encuentra dicha señal o (0) si no es así. Por lo tanto el bucle se ejecutará hasta que lleguemos al último dato. Como veréis, es sencillo, y útil. En la línea 40 se leen los datos del archivo, y en las líneas restantes, se muestra por pantalla, y al detectarse el final del fichero, se cierra y se señala en la pantalla.

Bien, ya hemos visto lo esencial del funcionamiento de los ficheros de acceso secuencial.

Cómo introducir datos, y cómo leer dichos datos del fichero. Naturalmente, los programas se pueden, y debéis intentarlo, mejorar mucho. Por ejemplo: hemos dicho que en estos ficheros, no se pueden modificar ni añadir registros directamente. Antes de ver el ejemplo 3, me gustaría que pensaseis en alguna manera de hacerlo. El proceso es muy simple. Tenemos un fichero y queremos añadir algunos daos. El BASIC de **AMSTRAD** nos permite abrir simultáneamente dos ficheros de acceso secuencial, siempre que uno sea de entrada y el otro de salida. Supongamos que creamos otro fichero, metemos en él todos los datos del fichero antiguo, por supuesto automáticamente y no tecleándolos otra vez, y luego tecleamos los nuevos datos. Lo cerramos, borramos el fichero antiguo y le cambiamos el nombre al nuevo llamándole como al anterior. Y ya está. Así de sencillo. Esto es exactamente lo que hace el ejemplo 3. Como veréis, es un resmen de todo lo dicho hasta ahora. Lo único nuevo, son las instrucciones de las líneas 150 y 160. El comando:

|ERA,"AGENDA"

borra el primer archivo, *(el creado en el primer programa)*, y el comando:

|REN, "AGENDA", "AGENDA2"

Descripción del campo	Nombre variable	Posición inicial	Longitud del campo
Dato 1	Día$ (1)	1	20
Dato 2	Día$ (2)	21	20
Dato 3	Día$ (3)	41	20
:	:	:	:
:	:	:	:
Dato 10	Día$ (10)	181	200
Longitud total de cada registro . . .			

da el nombre del anterior al nuevo fichero creado. **¿Por qué cambiamos el nombre, en vez de dejarlo como estaba?** Al hacer un programa que maneje un fichero, con altas, bajas, modificaciones y consultas, que es lo mínimo que debe tener, el nombre debe permanecer igual, de otra manera tendríamos que estar modificando el programa constantemente, y eso es poco práctico. **¿Y las modificaciones?** Pues bien, esto lo dejo para que lo penséis vosotros. Se haría un proceso similar al del programa anterior. Creáis otro fichero, pasáis los datos del fichero antiguo al nuevo, mediante el programa y hasta llegar al que queréis modificar. Este lo modificáis y grabáis, y luego el programa sigue pasando datos hasta llegar al final del fichero.

Hemos visto lo esencial para poder crear y manejar ficheros de acceso secuencial. Os recomiendo que antes de meteros con los ficheros de acceso directo, porbeis los ejemplos, e intenteis mejorarlos, clisificando el fichero alfabéticamente por ejemplo, hasta que los comprendais y manejeis con soltura. Así, los ficheros de acceso directo no os presentarán problemas.

Ficheros de acceso directo

Hemos visto que en los ficheros de acceso secuencial, para leer un registro intermedio, tenemos que leer todos los anteriores, y para modificarlo tenemos que crear otro fichero, donde meter los datos anteriores corregidos y los nuevos. Los ficheros de acceso directo permiten éstas y otras cosas directamente y por eso son sólo factibles en disco.

Lo primero que hay que aclarar es que el Basic de **Amstrad** no tiene comandos para manejar el acceso directo, y por eso se crearon los programas RANDOM-F.BAS y RANDOM.BIN que vienen en el disco de regalo del ordenador.

Al formatear un disco, lo que el ordenador está haciendo es organizar el espacio disponible de la siguiente manera: Divide el disco en 40 círculos concéntricos, llamados pistas, y cada una de éstas a su vez en 9 partes iguales llamadas sectores, y reserva algunos de ellos para crear un índice llamado directorio. Esto hace que cuando ordenemos la carga de un programa lo que el ordenador hace es leer primero el directorio, donde encuentra el nombre del programa y las pistas y sectores donde está escrito este programa y luego es cuando lo carga. Estos sectores no tienen por qué estar seguidos y de hecho suelen estar repartidos a lo largo del disco. Esto es un ejemplo de acceso directo.

Cuando creamos el fichero por medio del programa RANDOM-F.BAS, éste crea un índice para poder localizar cualquier registro directamente. Pero hay que tener dos cosas en cuenta. La primera es que tenemos que saber el número de un registro para acceder a él y poder aprovechar las facilidades de este tipo de acceso, aunque también lo podamos consultar sin saber el número de registro como lo haríamos con un fichero secuencial, pero se reduce mucho la eficiencia. Y la segunda y más importante, es que todos los registros deben tener la misma longitud, porque sólo así se puede saber de antemano dónde empezará cada registro. Esto nos lleva a un punto fundamental en este tipo de ficheros, el diseño del registro.

Antes de empezar a hacer el programa debemos pensar los datos que van a ir en cada registro, y realizarlo de manera que los datos estén perfectamente delimitados. Este sería el diseño de registro para el ejemplo 4.

Evidentemente, aunque en este caso la longitud de todos los campos sea igual, esto no tiene por qué ser así. Cada campo puede ser y normalmente lo es, de distinta longitud. Se toma papel y lápiz, se van poniendo los campos necesarios, posición de comienzo y longitud. Al final se suman las longitudes de todos los campos, y tenemos la longitud total del registro.

El programa RANDOM-F.BAS

Una vez que ya tenemos el diseño del registro, veamos qué hacer con él. Para esto, hay que ver que hace el programa RANDOM-F.BAS. Aunque su uso no presenta ningún problema, tiene alguna peculiaridad. Dicho programa no tiene por qué estar en el disco en que se vaya a crear el fichero, sino que se puede cargar primero, y luego cambiar de disco para crearlo en este último. El programa pide primero el nombre del fichero, y hay que escribirlo sin ninguna extensión, esto es sin .BAS ni .DAT, etc. Después pide el número de fichas, y la longitud de cada ficha. Aquí, comprobará que el fichero no sea mayor que la capacidad del disco —unos 150 Kb— y luego pedirá la unidad de disco. A no ser que tengamos dos unidades de disco la respuesta debe ser A, y si las tenemos, según en que unidad de disco tengamos el disco donde debe crearlo pondremos A para la unidad integrada en el ordenador, o B para la unidad externa. Por último, pedirá si queremos modificar algún dato y luego si queremos crear otro fichero.

Debe estar claro que este programa comprueba al crear un fichero si es más grande que la capacidad TOTAL del disco, y no de la capacidad libre en ese momento. Si el fichero ocupa por ejemplo 100 Kb. y el disco tiene 50 Kb. libres, los otros 50 Kb. los escribirá encima de algún programa, inutilizándolo. De aquí el consejo de formatear el disco antes, y si no se hace así, por lo menos tener copias de seguridad de los programas importantes que haya en el disco. Si creamos el fichero en un disco que ya contenga programas, puede escribir encima de ellos y estropear alguna parte o todo un programa. Y si creamos el fichero después de borrar algunos programas, entonces la lectura de registros que no hayan sido grabados antes da problemas, dado que puede contener datos que produzcan efectos impredecibles. Recordemos que al borrar algo del disco, sólo se borra el directorio, no el contenido. Este se borra al grabar algo encima. La solución más efectiva es crearlo en un disco recién formateado, porque formatear el disco borra absolutamente todo lo grabado en él. Y una solución sólo a medias es inicializar el fichero. Esto no evitará que el fichero haya borrado algo, pero por lo menos podremos consultar registros que no hayan sido grabados sin resultados extraños.

Expliquemos la inicialización. Consiste simplemente en pre-grabar todos los registros del fichero. Veamos las líneas 280-350. Hacemos que REG$ tenga la longitud del registro *(200 caracteres)* y contenga puntos *(o espacios...)* y la grabamos en todos los registros (31) del fichero. Esta rutina también vale para borrar el contenido del fichero, y volverlo a usar cambiándole el nombre. Normalmente con tener el fichero de este mes y el del próximo nos valdrá, pero como cada fichero ocupa 7 Kb. en un disco nos caben los de todo el año.

El programa RANDOM-BIN

Pasemos al otro programa. RANDOM-BIN carga en la memoria RAM del ordenador nuevos comandos Basic que luego podemos utilizar en nuestros programas. Naturalmente que para poder utilizar estos comandos, hay que tener en el disco además del nuestro, una copia de este programa y antes de ejecutar el programa, incluir en las primeras líneas las instrucciones de carga correspondientes. Estas son siempre las mismas, sin variación y en el ejemplo 4 corresponden a las líneas 90 a 110.

Una vez en memoria este programa, cuando el ordenador lea una instrucción que lleve delante el símbolo !, irá a buscarlo a la zona de memoria RAM donde se ha cargado, en vez de acudir a la ROM donde se encuentran los comandos normales.

¡Atención a una cosa! Si cargamos dos veces el programa RANDOM.BIN sin apagar el ordenador previamente, nos encontraremos con la «agradable» sorpresa de que el ordenador se nos queda «colgado» y sin posibilidad de recuperar lo que haya en la memoria. De ahí que haya que incluir de algún modo la manera de que el programa no pase dos veces por dichas líneas. La línea 80 del programa IV realiza esto, mirando si la posición de la memoria &9COO tiene el valor 1, si es así, se salta estas líneas y en caso contrario carga el programa. Recalco esto porque es francamente desagradable cargar un programa a medio hacer, probarlo para observar lo que hay hecho y lo que queda por hacer, pasarse alguna hora trabajando, probarlo otra vez sin haberlo salvado en disco y ver que nos ha bloqueado el ordenador sin posibilidad de recuperar el programa. Divertido ¿no?

Vayamos ya a la descripción de los comandos que nos permiten manejar ficheros de acceso directo. Así como en los ficheros secuenciales había dos instrucciones para abrir el fichero, una para escribir y otra para leer, en ficheros de acceso directo sólo hace falta una. Veámosla en el ejemplo 4:

```
260 !OPEN, @ me$ (mes), 1, 200, 1
```

La barra delante de OPEN indica que es un comando extendido. OPEN abre el fichero para lectura y escritura indistintamente. Luego hay que poner una coma, el signo @ y una variable a la que previamente hayamos asignado el nombre del fichero (no hay que poner el punto al final). Fijarse cómo las líneas 180-230 seleccionan este nombre. Después hay que poner otra coma, el número de orden del fichero, ya que podemos tener abiertos y manejar hasta quince ficheros distintos a la vez. Después, otra coma y la longitud del registro que tiene que coincidir con la longitud que dimos al programa RANDOM-F.BAS al crear el fichero. Luego otra coma, y un número que será (1) si el fichero está en la unidad integrada de disco, y (2) si tenemos una unidad externa y tenemos el disco en ella. Esta respuesta no tiene por qué coincidir con la que dimos en el programa RANDOM-F.BAS.

Podemos crear el fichero en una de ellas, y luego utilizarlo en la otra. En esta instrucción hay que poner la unidad de disco donde se vaya a acceder al fichero.

Leer y escribir en un fichero

Bien, ya tenemos abierto el fichero y podemos leer y/o escribir en él. Las instrucciones que lo permiten, referidas al ejemplo 4 son:

```
400 !READ, @ reg$, x, 1
330 !WRITE, @ reg$, x, 1
```

La instrucción READ, leerá el registro X, del archivo que hayamos abierto con el número 1, y la almacenará en la variable REG$.

La instrucción WRITE, escribirá el contenido de la variable REG$ en el registro X del archivo abierto con el número 1.

Para evitar errores la variable REG$ debe haber sido definida antes de utilizar READ. *(Ver línea 150.)* Una vez que ya tenemos la información en dicha variable, la repartiremos en los campos según el diseño de registro. Fijaros en la línea 420.:

```
420 días$ (X) = mid$ (reg$,z,20)
```

Almacenamos en dia$(1) los primeros 20 caracteres de reg$, en dia$(2) los siguientes 20... Naturalmente si el diseño de registro es más completo y los campos tienen distinta longitud, y distinto nombre, variarán los parámetros de la instrucción MID$, de una a otra variable. Fijaros en la importancia del diseño de registro. Si nos fiamos sólo de nuestra memoria podemos organizar un lío bastante grande, con sólo equivocarnos en un número. Siempre hay que poner UNA sola variable en las instrucciones READ y WRITE, y tiene que estar claro cómo lo hacemos.

El BASIC de **Amstrad** sólo permite 255 caracteres en una variable, luego nuestros registros sólo pueden tener una longitud menor de 255 caracteres. Claro que si utilizamos simultáneamente los quince ficheros permitidos, podemos manejar hasta 3.750 caracteres repartidos en quince o más campos. Pero esto limitaría la capacidad de cada fichero a unos 40 registros, dado que todos los ficheros deben estar en la misma cara del disco si los queremos tener abiertos a la vez.

Es necesario cuidado en la operación de escritura

Vamos con la parte más delicada de estos ficheros. La manera de escribir en ellos. Cuando nosotros definimos un registro, lo hacemos dando una longitud determinada a cada campo. Y luego, al escribir, almacenamos todas las variables en una, que es la que grabamos en el disco. Vamos a poner un ejemplo. Supongamos que tenemos un fichero con sólo dos datos, nombre y teléfono. Al campo nombre, le damos una longitud de 20 caracteres, y al de teléfono de 7 caracteres. Antes de agregar los dos campos a la variable que vamos a grabar, es fundamental comprobar su longitud, si es mayor recortarla y si es menor rellenarla de espacios (o puntos...) hasta llegar a la longitud determinada. Con los números pasa lo mismo. Al convertir un número en una cadena, la cadena tiene la longitud del número más un espacio para el signo. Esto hay que tenerlo en cuenta a la hora de reagrupar las variables en una sola. Fijaros en el ejemplo 4 líneas 980 y 990.

```
980 IF LEN(día$(x)) < 20 THEN
  día$(x) = día$(x) + SPACE$(20—LEN(día$)
990 IF LEN(día$(x)) > 20 THEN
        día$(x) = LEFT$(día$(x),20)
```

Si la longitud es menor de 20 caracteres, se rellena con espacios en blanco hasta llegar a esa cifra, y si es mayor de 20 caracteres toma sólo los 20 primeros, y luego la línea 1000 agrupa todas las variables en la variable que se incluye en el comando WRITE. Nunca se debe comprobar sólo la longitud de la cadena que vamos a grabar. Si —en el ejemplo— sólo se comprueba la longitud de reg$, corremos peligro de que ésta esté bien, porque tengamos un campo por ejemplo de 21 caracteres y otro de 19, por ejemplo. Pero al leer ese registro, como a los dos campos les hemos dado una longitud de 20 caracteres, habría algún dato cambiado de campo. Hay que comprobar siempre, pues, que estemos escribiendo lo que luego vamos a leer.

El último comando de acceso directo que queda por ver es CLOSE. Se puede poner con un número después, o sin él. En el ejemplo, está en la línea 1130.

```
1130 !CLOSE,1
```

Si ponemos después el número de un fichero, sólo cerrará éste, y si no ponemos ninguno, cerrar TODOS los ficheros que estén abiertos en ese momento.

Otra cosa a tener en cuenta es que si el programa es largo, es conveniente mantener sus ficheros abiertos el menor tiempo posible. Si sólo una parte de un programa maneja ficheros, se debe abrir el fichero justo antes de necesitar acceder a él, y luego de haberlo utilizado cerrarlo. Hay que tener en cuenta que un fallo eléctrico, sacar el disco, o un error de otro tipo que suceda antes de cerrar un fichero puede impedirnos el acceso a ese fichero. Sólo si se cierra correctamente tendremos asegurado el funcionamiento correcto de dicho fichero.

Por último, una serie de puntualizaciones sobre estos ficheros. El programa del ejemplo 4 es un ejemplo simple en el que siempre sabemos qué registro leer o escribir, dado que coincide el número de registro con el día del mes, y el acceso a cada registro no presenta complicaciones. Desgraciadamente no siempre es tan fácil. Otros programas necesitarán de algún algoritmo para calcular el número de un registro, o de alguna clave, y su escritura deberá ser secuencial.

En otras palabras, el programa deberá saber en todo momento cuál es el último registro grabado, para continuar en el siguiente.

El programa 4

Imaginaros un fichero que deba tener todas las facturas que da un comercio. Al hacer y grabar una factura, habrá que darla en número siguiente al de la última factura hecha. El número de factura podrá coincidir con el del registro, pero hay que saber el último número dado. Para esto hay varias soluciones. Una sería grabar una señal después del último registro grabado, y si se necesita el último, leer desde el primer registro, hasta que encontremos dicha señal. Normalmente se usa el asterisco.

Otra solución, la más cómoda, es reservar el primer registro de cada fichero para informar que nos pueda ser útil. Ahí es donde puede ir el número del último registro escrito, la fecha de la última actualización de un fichero, o cualquier dato que podamos necesitar. Naturalmente esto exigiría que cada vez que escribamos un registro, modifiquemos a su vez el primero, y lo pongamos al día. Pero si se ha comprendido todo lo explicado hasta ahora, y con un poco de práctica, esto no será ningún problema. No obstante, es aconsejable prever no sólo la localización de un registro por su número, sino también por algún campo importante, nombre, ciudad, etc. De esta manera habrá que leer registro a registro, comparando el campo correspondiente de cada uno con el dato que se busca, de manera secuencial. Al hablar de **«manera secuencial»** me refiero a leer todos los registros en orden, pero no a utilizar mezclados comandos de acceso secuencial y comandos de acceso directo, cosa nada recomendable.

Espero que lo dicho hasta ahora, os sirva para usar correctamente los tipos de acceso a ficheros, y os anime a hacer vuestros propios programas.

Notas sobre el funcionamiento del programa cuatro

Tanto la opción ESCRIBIR como BORRAR, sólo lo hacen en la pantalla. Para que se realice en el fichero, después de afectuar las modificaciones o añadir datos, hay que utilizar la opción G GRABAR. Por otra parte, la inicialización de ficheros, borra todo su contenido. Cuando acabe un mes, se puede inicializar el fichero, cambiarle el nombre, y volver a utilizarlo sin necesidad de usar RANDOM-F.BAS otra vez. Sólo habrá que cambiarle el nombre. Normalmente con un fichero para el mes actual y otro para el próximo será suficiente. Si sólo se tiene el fichero de un mes, la opción M CAMBIO DE MES será inútil.

PROGRAMAS

```
10 REM EJEMPLO 1 CREACION F.SEC.
20 OPENOUT "AGENDA"
30 INPUT "NOMBRE    ..:";nom$
40 INPUT "DIRECCION..:";dir$
50 INPUT "TELEFONO ..:";tel
60 WRITE #9,nom$,dir$,tel
70 INPUT "MAS DATOS? (S/N).:";re$
80 IF re$="S" OR re$="s" THEN 30
90 CLOSEOUT
```

```
10 REM EJEMPLO 2 LECTURA F.SEC.
20 OPENIN "AGENDA"
30 WHILE NOT EOF
40 INPUT #9,nom$,dir$,tel
50 PRINT "NOMBRE    ..:";nom$
60 PRINT "DIRECCION..:";dir$
70 PRINT "TELEFONO ..:";tel
80 INPUT "Deseas ver el siguiente?
(S/N).:";re$
90 IF re$="N" OR re$="n" THEN 110
100 WEND:PRINT "No hay mas datos."
110 CLOSEIN:END
```

```
10 REM EJEMPLO 3 FUSION F.SEC.
20 OPENIN "agenda"
30 OPENOUT "agenda2"
40 WHILE NOT EOF
50 INPUT #9,nom$,dir$,tel
60 WRITE #9,nom$,dir$,tel
70 WEND:
75 CLOSEIN:
80 INPUT "NOMBRE    ..:";nom$
90 INPUT "DIRECCION..:";dir$
100 INPUT "TELEFONO ..:";tel
110 WRITE #9,nom$,dir$,tel
120 INPUT "Mas datos? (S/N)..:";re$
130 IF re$="S" OR re$="s" THEN 80
140 CLOSEOUT:
150 !ERA,"agenda"
160 !REN,"agenda.","agenda2."
```

```
10 REM EJEMPLO 4 fichero aleatorio
20 '    Nombre del fichero.: ENERO
30 '    Longitud del registro.:200 c
aracteres
40 '    Numero de registros.: 31
50 '    Disgio del registro
60 '    10 Campos de 20 caracteres
70 '-------------------------------
---------
80 IF PEEK (&9C00)=1 THEN GOTO 130:
' Comprueba si ya se ha cargado "RA
NDOM"
90 MEMORY &9BFF
100 LOAD "random.bin"
110 CALL &9C00
120 '-------------------------------
---------
130 REM principio
140 '-------------------------------
---------
150 CLS:MODE 1: DIM me$(12),dia$(10
):reg$="": z=1
160 WINDOW #1,1,40,1,17
170 WINDOW #2,1,40,18,25
180 FOR x=1 TO 12
190 READ d$:me$(x)=d$
200 NEXT x
210 DATA ENERO,FEBRERO,MARZO,ABRIL,
MAYO,JUNIO,JULIO
220 DATA AGOSTO,SEPTIEMBRE,OCTUBRE,
NOVIEMBRE,DICIEMBRE
230 INPUT #1,"    Que mes desea ver
 (1..12).:";mes
240 IF mes>12 OR mes<1 THEN SOUND 3
,200:GOTO 230
250 INPUT #1,"    Desea inicializarl
o  (S/N)..:";in$:in$=UPPER$(in$)
260 !OPEN,9 ME$(MES),1,200,1: 'APER
TURA DEL FICHERO
270 IF in$="N" THEN GOTO 380
280 '-------------------------------
---------
290 REM inicializacion del fichero
300 '-------------------------------
---------
310 reg$=STRING$(200,".")
320 FOR x=1 TO 31
330 !WRITE,9 reg$,x.1
340 NEXT x
350 '-------------------------------
---------
360 REM fin inicializacion
370 '-------------------------------
---------
380 PRINT #1,"    Abierto fichero "i
me$(mes)
390 INPUT #1,"    Que dia desea ver
 (1..31).:";dia
400 !READ,9 reg$,dia.1
410 FOR x=1 TO 10
420 dia$(x)=MID$(reg$,z,20)
430 z=z+20
440 NEXT x
450 '-------------------------------
---------
460 '    -PANTALLA PRINCIPAL
470 '-------------------------------
---------
480 CLS #1:
490 LOCATE #1,5,1:PRINT #1,"DIA "jd
ia;" de ";me$(mes)
500 LOCATE #1,1,3:PRINT #1,"No. HOR
A          APUNTE"
510 LOCATE #1,1,4:PRINT #1,"=== ===
===   ================="
520 LOCATE #1,1,5:PRINT #1," 1/ 09.
00 -> ";dia$(1)
530 LOCATE #1,1,6:PRINT #1," 2/ 10.
00 -> ";dia$(2)
540 LOCATE #1,1,7:PRINT #1," 3/ 11,
00 -> ";dia$(3)
550 LOCATE #1,1,8:PRINT #1," 4/ 12.
00 -> ";dia$(4)
560 LOCATE #1,1,9:PRINT #1," 5/ 13.
00 -> ";dia$(5)
570 LOCATE #1,1,10:PRINT #1," 6/ 16
.00 -> ";dia$(6)
580 LOCATE #1,1,11:PRINT #1," 7/ 17
.00 -> ";dia$(7)
590 LOCATE #1,1,12:PRINT #1," 8/ 18
.00 -> ";dia$(8)
600 LOCATE #1,1,13:PRINT #1," 9/ 19
.00 -> ";dia$(9)
610 LOCATE #1,1,14:PRINT #1,"10/ 20
.00 -> ";dia$(10)
620 '-------------------------------
---------
630 REM seleccion de opciones
640 '-------------------------------
---------
650 LOCATE #2,1,1:PRINT #2,"E=Escri
bir.    B=Borrar.      G=Grabar."
660 LOCATE #2,1,3:PRINT #2,"D=Cambi
o dia. M=Cambio mes. F=Acabar."
670 LOCATE #2,1,5:INPUT #2,"SELECCI
ONE OPCION (E,B,G,D,M,F).:",OP$:op$
=UPPER$(op$)
680 L=INSTR("EBGDMF",op$)
690 IF L=0 THEN SOUND 3,200:GOTO 67
0
700 ON L GOTO 710,830,920,1040,1090
,1150
710 '-------------------------------
---------
720 REM OPCION E ESCRIBIR
730 '-------------------------------
---------
740 CLS #2:
750 LOCATE #2,1,1:INPUT #2," Numero
 de apunte (1..10).:",dat
760 IF dat<1 OR dat>10 THEN SOUND 3
,200:GOTO 750
770 LOCATE #2,1,3:PRINT #2," Apunte
s (20 C)..: ....................."
780 LOCATE #2,20,3:INPUT #2,"",AP$
790 IF LEN(ap$)>20 THEN SOUND 3,200
:GOTO 770
800 LOCATE #1,14,dat+4:PRINT #1,ap$
;SPACE$(20-LEN(ap$))
810 dia$(dat)=ap$
820 CLS #2:GOTO 630
830 '-------------------------------
---------
840 REM OPCION B BORRAR
850 '-------------------------------
---------
860 CLS #2:
870 LOCATE #2,1,1:INPUT #2," Numero
 de apunte (1..10).:",dat
880 IF dat<1 OR dat>10 THEN SOUND 3
,200:GOTO 870
890 LOCATE #1,14,dat+4:PRINT #1,SPA
CE$(20)
900 dia$(dat)=STRING$(20," ")
910 CLS #2:GOTO 630
920 '-------------------------------
---------
930 REM OPCION G GRABAR
940 '-------------------------------
---------
950 CLS #2:PRINT #2,"         GRABAND
O REGISTRO ";DIA:
960 REG$=""
970 FOR X=1 TO 10
980 IF LEN (DIA$(X))<20 THEN DIA$(X
)=DIA$(X)+SPACE$(20-LEN(DIA$(X)))
990 IF LEN (DIA$(X))>20 THEN DIA$(X
)=LEFT$(DIA$(X),20)
1000 REG$=REG$+DIA$(X)
1010 NEXT X
1020 !WRITE,9 REG$,DIA.1
1030 CLS #2:GOTO 630
1040 '-------------------------------
---------
1050 REM OPCION D CAMBIO DE DIA
1060 '-------------------------------
---------
1070 CLS #1:CLS #2:z=1
1080 GOTO 380
1090 '-------------------------------
---------
1100 REM OPCION C CAMBIO DE MES
1110 '-------------------------------
---------
1120 CLS #1:CLS #2:z=1
1130 !CLOSE,1
1140 GOTO 230
1150 '-------------------------------
---
1160 REM OPCION F FINAL
1170 '-------------------------------
---
1180 !CLOSE,1
1190 CLS:END
```