CAPÍTULO 13

GESTIÓN DE ARCHIVOS

Hasta el momento, toda la información (datos) que hemos sido capaces de gestionar, la hemos tomado de dos únicas fuentes: o eran datos del programa, o eran datos que introducía el usuario desde el teclado. Y hasta el momento, siempre que un programa ha obtenido un resultado, lo único que hemos hecho ha sido mostrarlo en pantalla.

Y, desde luego, sería muy interesante poder almacenar la información generada por un programa, de forma que esa información pudiera luego ser consultada por otro programa, o por el mismo u otro usuario. O sería muy útil que la información que un usuario va introduciendo por consola quedase almacenada para sucesivas ejecuciones del programa o para posibles manipulaciones de esa información.

En definitiva, sería muy conveniente poder almacenar en algún soporte informático (por ejemplo, en el disco del ordenador) esa información, y poder luego acceder a ese disco para volver a tomarla, para actualizar la

información almacenada, para añadir o para eliminar todo o parte de ella.

Y eso es lo que vamos a ver en este tema: la gestión de archivos. Comenzaremos con una breve presentación de carácter teórico sobre los archivos y pasaremos a ver después el modo en que podemos emplear los distintos formatos de archivo.

Tipos de dato con persistencia

Entendemos por tipo de dato con persistencia, o **archivo**, o **fichero** aquel cuyo tiempo de vida no está ligado al de ejecución del programa que lo crea o lo maneja. Es decir, se trata de una estructura de datos externa al programa, que lo trasciende. Un archivo existe desde que un programa lo crea y mientras que no sea destruido por este u otro programa.

Un archivo está compuesto por registros homogéneos que llamamos **registros de archivo**. La información de cada registro viene recogida mediante **campos**.

Es posible crear ese tipo de dato con persistencia porque esa información queda almacenada sobre una memoria externa. Los archivos se crean sobre dispositivos de memoria masiva. El límite de tamaño de un archivo viene condicionado únicamente por el límite de los dispositivos físicos que lo albergan.

Los programas trabajan con datos que residen en la memoria principal del ordenador. Para que un programa manipule los datos almacenados en un archivo y, por tanto, en un dispositivo de memoria masiva, esos datos deben ser enviados desde esa memoria externa a la memoria principal mediante un proceso de **extracción**. Y de forma similar, cuando los datos que manipula un programa deben ser concatenados con los del archivo se utiliza el proceso de **grabación**.

De hecho, los archivos se conciben como estructuras que gozan de las siguientes características:

1. Capaces de contener grandes cantidades de información.
2. Capaces de y sobrevivir a los procesos que lo generan y utilizan.
3. Capaces de ser accedidos desde diferentes procesos o programas.

Desde el punto de vista físico, o del hardware, un archivo tiene una dirección física: en el disco toda la información se guarda (grabación) o se lee (extracción) en bloques unidades de asignación o «clusters» referenciados por un nombre de unidad o disco, la superficie a la que se accede, la pista y el sector: todos estos elementos caracterizan la dirección física del archivo y de sus elementos. Habitualmente, sin embargo, el sistema operativo simplifica mucho esos accesos al archivo, y el programador puede trabajar con un concepto simplificado de **archivo** o **fichero**: cadena de bytes consecutivos terminada por un carácter especial llamado **EOF** ("End Of File"); ese carácter especial (EOF) indica que no existen más bytes de información más allá de él.

Este segundo concepto de archivo permite al usuario trabajar con datos persistentes sin tener que estar pendiente de los problemas físicos de almacenamiento. El sistema operativo posibilita al programador trabajar con archivos de una forma sencilla. El sistema operativo hace de interfaz entre el disco y el usuario y sus programas.

1. Cada vez que accede a un dispositivo de memoria masiva para leer o paras grabar, el sistema operativo transporta, desde o hasta la memoria principal, una cantidad fija de información, que se llama **bloque** o **registro físico** y que depende de las características físicas del citado dispositivo. En un bloque o registro físico puede haber varios registros de archivo, o puede que un registro de archivo ocupe varios bloques. Cuantos más registros de archivo quepan en cada bloque menor será el número de accesos necesarios al dispositivo de

almacenamiento físico para lograr procesar toda la información del archivo.

2. El sistema operativo también realiza la necesaria transformación de las direcciones: porque una es la posición real o efectiva donde se encuentra el registro dentro del soporte de información (dirección física o **dirección hardware**) y otra distinta es la posición relativa que ocupa el registro en nuestro archivo, tal y como es visto este archivo por el programa que lo manipula (**dirección lógica** o simplemente **dirección**).
3. **Un archivo es una estructura de datos externa al programa**. Nuestros programas acceden a los archivos para leer, modificar, añadir, o eliminar registros. El proceso de lectura o de escritura también lo gobierna el sistema operativo. Al leer un archivo desde un programa, se transfiere la información, de bloque en bloque, desde el archivo hacia una zona reservada de la memoria principal llamada **buffer**, y que está asociada a las operaciones de entrada y salida de archivo. También se actúa a través del buffer en las operaciones de escritura sobre el archivo.

Archivos y sus operaciones

Antes de abordar cómo se pueden manejar los archivos en C, será conveniente hacer una breve presentación sobre los archivos con los que vamos a trabajar: distintos modos en que se pueden organizar, y qué operaciones se pueden hacer con ellos en función de su modo de organización.

Hay diferentes modos de estructurar o de organizar un archivo. Las características del archivo y las operaciones que con él se vayan a poder realizar dependen en gran medida de qué modo de organización se adopte. Las dos principales formas de organización que vamos a ver en este manual son:

1. **Secuencial**. Los registros se encuentran en un orden secuencial, de forma consecutiva. Los registros deben ser leídos, necesariamente, según ese orden secuencial. Es posible leer o escribir un cierto número de datos comenzando siempre desde el principio del archivo. También es posible añadir datos a partir del final del archivo. El acceso secuencial es una forma de acceso sistemático a los datos poco eficiente si se quiere encontrar un elemento particular.
2. **Indexado**. Se dispone de un índice para obtener la ubicación de cada registro. Eso permite localizar cualquier registro del archivo sin tener que leer todos los que le preceden.

La decisión sobre cuál de las dos formas de organización tomar dependerá del uso que se dé al archivo.

Para poder trabajar con **archivos secuenciales**, se debe previamente asignar un nombre o identificador a una dirección de la memoria externa (a la que hemos llamado antes dirección hardware). Al crear ese identificador se define un indicador de posición de archivo que se coloca en esa dirección inicial. Al iniciar el trabajo con un archivo, el indicador se coloca en el primer elemento del archivo que coincide con la dirección hardware del archivo.

Para extraer un registro del archivo, el indicador debe previamente estar ubicado sobre él; y después de que ese elemento es leído o extraído, el indicador se desplaza automáticamente al siguiente registro de la secuencia.

Para añadir nuevos registros primero es necesario que el indicador se posicione o apunte al final del archivo. Conforme el archivo va creciendo de tamaño, a cada nuevo registro se le debe asignar nuevo espacio en esa memoria externa.

Y si el archivo está realizando acceso de lectura, entonces no permite el de escritura; y al revés: no se pueden utilizar los dos modos de acceso (lectura y escritura) de forma simultánea.

Las operaciones que se pueden aplicar sobre un archivo secuencial son:

1. **Creación** de un nuevo archivo vacío. El archivo es una secuencia vacía: ().
2. **Adición** de registros mediante buffer. La adición almacenar un registro nuevo concatenado con la secuencia actual. El archivo pasa a ser la secuencia (secuencia inicial + buffer). La información en un archivo secuencial solo es posible añadirla al final del archivo.
3. **Inicialización** para comenzar luego el proceso de extracción. Con esta operación se coloca el indicador sobre el primer elemento de la secuencia, dispuesto así para comenzar la lectura de registros. El archivo tiene entonces la siguiente estructura: Izquierda = (); Derecha = (secuencia); Buffer = primer elemento de (secuencia).
4. **Extracción** o lectura de registros. Esta operación coloca el indicador sobre el primer elemento o registro de la parte derecha del archivo y concatena luego el primer elemento de la parte derecha al final de la parte izquierda. Y eso de forma secuencial: para leer el registro n en el archivo es preciso leer previamente todos los registros desde el 1 hasta el n – 1. Durante el proceso de extracción hay que verificar, antes de cada nueva lectura, que no se ha llegado todavía al final del archivo y que, por tanto, la parte derecha aún no es la secuencia vacía.

Y no hay más operaciones. Es decir, no se puede definir ni la operación inserción de registro, ni la operación modificación de registro, ni la operación borrado de registro. Al menos diremos que no se realizan fácilmente. La operación de inserción se puede realizar creando de hecho un nuevo archivo. La modificación se podrá hacer si al realizar la modificación no se aumenta la longitud del registro. Y el borrado no es posible y, por tanto, en los archivos secuenciales se define el borrado lógico: marcar el registro de tal forma que esa marca se interprete como elemento borrado.

Archivos de texto y binarios

Decíamos antes que un archivo es un conjunto de bytes secuenciales, terminados por el carácter especial **EOF**.

Si nuestro archivo es de texto, esos bytes serán interpretados como caracteres. Toda la información que se puede guardar en un archivo de texto son caracteres. Esa información podrá por tanto ser visualizada por un editor de texto.

Si se desean almacenar los datos de una forma más eficiente, se puede trabajar con archivos binarios. Los números, por ejemplo, no se almacenan como cadenas de caracteres, sino según la codificación interna que use el ordenador. Esos archivos binarios no pueden visualizarse mediante un editor de texto.

Si lo que se desea es que nuestro archivo almacene una información generada por nuestro programa y que luego esa información pueda ser, por ejemplo, editada, entonces se deberá trabajar con ficheros de caracteres o de texto. Si lo que se desea es almacenar una información que pueda luego ser procesada por el mismo u otro programa, entonces es mejor trabajar con ficheros binarios.

Tratamiento de archivos en el lenguaje C

Ya hemos visto todas las palabras reservadas de C. Ninguna de ellas hace referencia a operación alguna de entrada o salida de datos. Todas las operaciones de entrada y salida están definidas mediante funciones de biblioteca estándar.

Para trabajar con archivos con buffer, las funciones, tipos de dato predefinidos y constantes están recogidos en la biblioteca **stdio.h**. Para trabajar en entrada y salida de archivos sin buffer están las funciones definidas en **io.h**.

Todas las funciones de **stdio.h** de acceso a archivo trabajan mediante una interfaz que está localizada por un puntero. Al crear un archivo, o al trabajar con él, deben seguirse las normas que dicta el sistema operativo. De trabajar así se encargan las funciones ya definidas, y esa gestión es transparente para el programador.

Esa interfaz permite que el trabajo de acceso al archivo sea independiente del dispositivo final físico donde se realizan las operaciones de entrada o salida. Una vez el archivo ha quedado abierto, se puede intercambiar información entre ese archivo y el programa. El modo en que la interfaz gestiona y realiza ese tráfico es algo que no afecta para nada a la programación.

Al abrir, mediante una función, un archivo que se desee usar, se indica, mediante un nombre, a qué archivo se quiere acceder; y esa función de apertura devuelve al programa una dirección que deberá emplearse en las operaciones que se realicen con ese archivo desde el programa. Esa dirección se recoge en un puntero, llamado **puntero de archivo**. Es un puntero a una estructura que mantiene información sobre el archivo: la dirección del buffer, el código de la operación que se va a realizar, etc. De nuevo el programador no se debe preocupar de esos detalles: simplemente debe declarar en su programa un puntero a archivo, como ya veremos más adelante.

El modo en que las funciones estándar de ANSI C gestionan todo el acceso a disco es algo transparente al programador. Cómo trabaja realmente el sistema operativo con el archivo sigue siendo algo que no afecta al programador. Pero es necesario que de la misma manera que una función de ANSI C ha negociado con el sistema operativo la apertura del archivo y ha facilitado al programador una dirección de memoria, también sea una función de ANSI C quien cierre al final del proceso los archivos abiertos, de forma también transparente para el programador. Si se interrumpe inesperadamente la ejecución de un programa, o éste termina sin haber cerrado los archivos que tiene

abiertos, se puede sufrir un daño irreparable sobre esos archivos, y perderlos o perder parte de su información.

También es transparente al programador el modo en que se accede de hecho a la información del archivo. El programa no accede nunca al archivo físico, sino que actúa siempre y únicamente sobre la memoria intermedia o buffer, que es el lugar de almacenamiento temporal de datos. Únicamente se almacenan los datos en el archivo físico cuando la información se transfiere desde el buffer hasta el disco. Y esa transferencia no necesariamente coincide con la orden de escritura o lectura que da el programador. De nuevo, por tanto, es muy importante terminar los procesos de acceso a disco de forma regular y normalizada, pues de lo contrario, si la terminación del programa se realiza de forma anormal, es muy fácil que se pierdan al menos los datos que estaban almacenados en el buffer y que aún no habían sido, de hecho, transferidos a disco.

Archivos secuenciales con buffer.

Antes de utilizar un archivo, la primera operación, previa a cualquier otra, es la de apertura.

Ya hemos dicho que cuando abrimos un archivo, la función de apertura asignará una dirección para ese archivo. Debe por tanto crearse un puntero para recoger esa dirección.

En la biblioteca **stdio.h** está definido el tipo de dato ***FILE***, que es tipo de dato puntero a archivo. Este puntero nos permite distinguir entre los diferentes ficheros abiertos en el programa. Crea la secuencia o interfaz que nos permite la transferencia de información con el archivo apuntado.

La sintaxis para la declaración de un puntero a archivo es la siguiente:

FILE *puntero_a_archivo;

Vamos ahora a ir viendo diferentes funciones definidas en **stdio.h** para la manipulación de archivos.

- **Apertura de archivo.**

  La función ***fopen*** abre un archivo y devuelve un puntero asociado al mismo, que puede ser utilizado para que el resto de funciones de manipulación de archivos accedan a este archivo abierto.

  Su prototipo es:

  FILE *fopen(const char*nombre_archivo, const char *modo_apertura);

  Donde ***nombre_archivo*** es el nombre del archivo que se desea abrir. Debe ir entre comillas dobles, como toda cadena de caracteres. El nombre debe estar consignado de tal manera que el sistema operativo sepa identificar el archivo de qué se trata.

  Y donde ***modo_apertura*** es el modo de acceso para el que se abre el archivo. Debe ir en comillas dobles. Los posibles modos de apertura de un archivo secuencial con buffer son:

  "r" Abre un archivo de texto para lectura. El archivo debe existir.

  "w" Abre un archivo de texto para escritura. Si existe ese archivo, lo borra y lo crea de nuevo. Los datos nuevos se escriben desde el principio.

  "a" Abre un archivo de texto para escritura. Los datos nuevos se añaden al final del archivo. Si ese archivo no existe, lo crea.

  "r+" Abre un archivo de texto para lectura/escritura. Los datos se escriben desde el principio. El fichero debe existir.

  "w+" Abre un archivo de texto para lectura/escritura. Los datos se escriben desde el principio. Si el fichero no existe, lo crea.

"rb" Abre un archivo binario para lectura. El archivo debe existir.

"wb" Abre un archivo binario para escritura. Si existe ese archivo, lo borra y lo crea de nuevo. Los datos nuevos se escriben desde el principio.

"ab" Abre un archivo binario para escritura. Los datos nuevos se añaden al final del archivo. Si ese archivo no existe, lo crea.

"r+b" Abre un archivo binario para lectura/escritura. Los datos se escriben desde el principio. El fichero debe existir.

"w+b" Abre un archivo binario para lectura/escritura. Los datos se escriben desde el principio. Si el fichero no existe, lo crea.

Ya vemos que hay muy diferentes formas de abrir un archivo. Queda claro que de todas ellas destacan dos bloques: aquellas que abren el archivo para manipular una información almacenada en binario, y otras que abren el archivo para poder manipularlo en formato texto. Ya iremos viendo ambas formas de trabajar la información a medida que vayamos presentando las distintas funciones.

La función ***fopen*** devuelve un puntero a una estructura que recoge las características del archivo abierto. Si se produce algún error en la apertura del archivo, entonces la función ***fopen*** devuelve un puntero nulo.

Ejemplos simples de esta función serían:

```
FILE *fichero;
fichero = fopen("datos.dat","w");
```

Que deja abierto el archivo *datos.dat* para escritura. Si ese archivo ya existía, queda eliminado y se crea otro nuevo y vacío.

El nombre del archivo puede introducirse mediante variable:

```
char nombre_archivo[80];
printf("Indique el nombre del archivo ... ");
```

```
gets(nombre_archivo);
fopen(nombre_archivo, "w");
```

Y ya hemos dicho que si la función ***fopen*** no logra abrir el archivo, entonces devuelve un puntero nulo. Es muy conveniente verificar siempre que el fichero ha sido realmente abierto y que no ha habido problemas:

```
FILE *archivo;
if(archivo = fopen("datos.dat", "w") == NULL)
      printf("No se puede abrir el archivo\n");
```

Dependiendo del compilador se podrán tener más o menos archivos abiertos a la vez. En todo caso, siempre se podrán tener, al menos ocho archivos abiertos simultáneamente.

- **Cierre del archivo abierto.**

  La función ***fclose*** cierra el archivo que ha sido abierto mediante *fopen*. Su prototipo es el siguiente:

  int fclose(FILE *nombre_archivo);

  La función devuelve el valor cero si ha cerrado el archivo correctamente. Un error en el cierre de un archivo puede ser fatal y puede generar todo tipo de problemas. El más grave de ellos es el de la pérdida parcial o total de la información del archivo.

  Cuando una función termina normalmente su ejecución, cierra de forma automática todos sus archivos abiertos. De todas formas es conveniente cerrar los archivos cuando ya no se utilicen dentro de la función, y no mantenerlos abiertos en espera de que se finalice su ejecución.

- **Escritura de un carácter en un archivo.**

  Existen dos funciones definidas en **stdio.h** para escribir un carácter en el archivo. Ambas realizan la misma función y ambas se utilizan indistintamente. La duplicidad de definición es necesaria para preservar la compatibilidad con versiones antiguas de C.

  Los prototipos de ambas funciones son:

int putc(int c, FILE *nombre_archivo);

int fputc(int c, FILE * nombre_archivo);

Donde ***nombre_archivo*** recoge la dirección que ha devuelto la función *fopen*. El archivo debe haber sido abierto para escritura y en formato texto. Y donde la variable *c* es el carácter que se va a escribir. Por razones históricas, ese carácter se define como un entero, pero de esos dos o cuatro bytes (dependiendo de la longitud de la palabra) sólo se toma en consideración el menos significativo.

Si la operación de escritura se realiza con éxito, la función devuelve el mismo carácter escrito.

Vamos a hacer un programa que solicite al usuario su nombre y entonces guarde ese dato en un archivo que llamaremos nombre.dat.

```
#include <stdio.h>
#include <stdlib.h>
void main(void)
{
	char nombre[80];
	short int i;
	FILE *archivo;

	printf("Su nombre ... ");
	gets(nombre);

	archivo = fopen("nombre.dat", "w");
	if(archivo == NULL)
	{
		printf("No se ha podido abrir el archivo.\n");
		printf("Pulse una tecla para finalizar... ");
		getchar();
		exit(1);
	}
	i = 0;
	while(nombre[i] != NULL)
	{
		fputc(nombre[i],archivo);
		i++;
	}
	fclose(archivo);
}
```

Una vez ejecutado el programa, y si todo ha ido correctamente, se podrá abrir el archivo *nombre.dat* con un editor de texto y comprobar que realmente se ha guardado el nombre en ese archivo.

- **Lectura de un carácter desde un archivo.**

  De manera análoga a las funciones de escritura, existen también funciones de lectura de caracteres desde un archivo. De nuevo hay dos funciones equivalentes, cuyos prototipos son:

  int fgetc(FILE *nombre_archivo);

  int getc(FILE * nombre_archivo);

  Que reciben como parámetro el puntero devuelto por la función *fopen* al abrir el archivo y devuelven el carácter, de nuevo como un entero. El archivo debe haber sido abierto para lectura y en formato texto. Cuando ha llegado al final del archivo, la función ***fgetc***, o ***getc***, devuelve una marca de fin de archivo que se codifica como **EOF**.

  El código para leer el nombre desde el archivo donde ha quedado almacenado en el programa anterior sería:

```
#include <stdio.h>
#include <stdlib.h>
void main(void)
{
      char nombre[80];
      short int i;
      FILE *archivo;

      archivo = fopen("nombre.dat", "r");
      if(archivo == NULL)
      {
            printf("No se ha podido abrir el archivo.\n");
            printf("Pulse una tecla para finalizar... ");
            getchar();
            exit(1);
      }
      i = 0;
      while((nombre[i++] = fgetc(archivo)) != EOF);
      /* El último elemento de la cadena ha quedado igual
         a EOF. Se cambia al carácter fin de cadena, NULL */
      nombre[--i] = NULL;
      fclose(archivo);
```

```
      printf("Su nombre ... %s", nombre);
}
```

Este código mostrará por pantalla el nombre almacenado en el archivo *nombre.dat*.

- **Lectura y escritura de una cadena de caracteres.**

  Las funciones ***fputs*** y ***fgets*** escriben y leen, respectivamente, cadenas de caracteres sobre archivos de disco.

  Sus prototipos son:

  int fputs(const char *s, FILE *nombre_archivo);

  char *fgets(char *s, int n, FILE * nombre_archivo);

  La función ***fputs*** escribe la cadena ***s*** en el archivo indicado por el puntero *nombre_archivo*. Si la operación ha sido correcta, devuelve un valor no negativo. El archivo debe haber sido abierto en formato texto y para escritura o para lectura, dependiendo de la función que se emplee.

  La función ***fgets*** lee del archivo indicado por el puntero *nombre_archivo* una cadena de caracteres. Lee los caracteres desde el inicio hasta un total de ***n***, que es el valor que recibe como segundo parámetro. Si antes del carácter n-ésimo ha terminado la cadena, también termina la lectura y cierra la cadena con un carácter nulo.

  En el programa que vimos para la función *fputc* podríamos eliminar la variable *i* y cambiar la estructura ***while*** por la sentencia:

  ```
  fputs(nombre,archivo);
  ```

  Y en el programa que vimos para la función *fgetc*, la sentencia podría quedar sencillamente:

  ```
  fgets(nombre, 80, archivo);
  ```

- **Lectura y escritura formateada.**

Las funciones ***fprintf*** y ***fscanf*** de entrada y salida de datos por disco tienen un uso semejante a las funciones *printf* y *scanf*, de entrada y salida por consola.

Sus prototipos son:

int fprintf(FILE *nombre_archivo, const char *cadena_formato [, argumento, ...]);

int fscanf(FILE *nombre_archivo, const char *cadena_formato [, dirección, ...]);

Donde ***nombre_archivo*** es el puntero a archivo que devuelve la función *fopen*. Los demás argumentos de estas dos funciones ya los conocemos, pues son los mismos que las funciones de entrada y salida por consola. La función ***fscanf*** devuelve el carácter EOF si ha llegado al final del archivo. El archivo debe haber sido abierto en formato texto y para escritura o para lectura, dependiendo de la función que se emplee.

Veamos un ejemplo de estas dos funciones. Hagamos un programa que guarde en un archivo (que llamaremos *numeros.dat*") los valores que previamente se han asignado de forma aleatoria a un vector de variables ***float***. Esos valores se almacenan dentro de una cadena de texto. Y luego, el programa vuelve a abrir el archivo para leer los datos y cargarlos en otro vector y los muestra en pantalla.

```
#include <stdio.h>
#include <stdlib.h>
#define TAM 10
void main(void)
{
      float or[TAM], cp[TAM];
      short i;
      FILE *ARCH;
      char c[100];

      randomize();
      for(i = 0 ;  i < TAM ; i++)
            or[i] = (float)random(1000) / random(100);

      ARCH = fopen("numeros.dat", "w");
      if(ARCH == NULL)
```

```
      {
            printf("No se ha podido abrir el archivo.\n");
            printf("Pulse una tecla para finalizar... ");
            getchar();
            exit(1);
      }
      for(i = 0 ; i < TAM ;  i++)
            fprintf(ARCH,"Valor %04hi-->%12.4f\n",i,or[i]);

      fclose(ARCH);

      ARCH = fopen("numeros.dat", "r");
      if(ARCH == NULL)
      {
            printf("No se ha podido abrir el archivo.\n");
            printf("Pulse una tecla para finalizar... ");
            getchar();
            exit(1);
      }

      printf("Los valores guardados en el archivo son:\n");
      i = 0;
      while(fscanf(ARCH,"%s%s%s%f",c,c,c,cp + i++)!= EOF);
      for(i = 0 ; i < TAM ; i++)
            printf("Valor %04hd --> %12.4f\n",i,cp[i]);
      fclose(ARCH);
}
```

El archivo contiene (en una ejecución cualquiera: los valores son aleatorios, y en cada ejecución llegaremos a valores diferentes) la siguiente información:

```
Valor 0000 -->       9.4667
Valor 0001 -->      30.4444
Valor 0002 -->      12.5821
Valor 0003 -->       0.2063
Valor 0004 -->      16.4545
Valor 0005 -->      28.7308
Valor 0006 -->       9.9574
Valor 0007 -->       0.1039
Valor 0008 -->      18.0000
Valor 0009 -->       4.7018
```

Hemos definido la variable *c* para que vaya cargando desde el archivo los tramos de cadena de caracteres que no nos interesan para la obtención, mediante la función *fscanf*, de los sucesivos valores ***float*** generados. Con esas tres lecturas de cadena la variable *c* va leyendo las cadenas "Valor"; la cadena de caracteres que recoge el índice *i*; la

cadena"-->". La salida por pantalla tendrá la misma apariencia que la obtenida en el archivo.

Desde luego, con la función *fscanf* es mejor codificar bien la información del archivo, porque de lo contrario la lectura de datos desde el archivo puede llegar a hacerse muy incómoda.

- **Lectura y escritura en archivos binarios.**

  Ya hemos visto las funciones para acceder a los archivos secuenciales de tipo texto. Vamos a ver ahora las funciones de lectura y escritura en forma binaria.

  Si en todas las funciones anteriores hemos requerido que la apertura del fichero o archivo se hiciera en formato texto, ahora desde luego, para hacer uso de las funciones de escritura y lectura en archivos binarios, el archivo debe hacer sido abierto en formato binario.

  Las funciones que vamos a ver ahora permiten la lectura o escritura de cualquier tipo de dato.

  Los prototipos son los siguientes:

  size_t fread(void *buffer, size_t n_bytes, size_t contador, FILE *nombre_archivo);

  size_t fwrite(const void *buffer, size_t n_bytes, size_t contador, FILE *nombre_archivo);

  Donde ***buffer*** es un puntero a la región de memoria donde se van a escribir los datos leídos en el archivo, o el lugar donde están los datos que se desean escribir en el archivo. Habitualmente será la dirección de una variable. ***n_bytes*** es el número de bytes que ocupa cada dato que se va a leer o grabar, y ***contador*** indica el número de datos de ese tamaño que se van a leer o grabar. El último parámetro es el de la dirección que devuelve la función *fopen* cuando se abre el archivo.

Ambas funciones devuelven el número de elementos escritos o leídos. Ese valor debe ser el mismo que el de la variable *contador*, a menos que haya ocurrido un error.

Estas dos funciones son útiles para leer y escribir cualquier tipo de información. Es habitual emplearla junto con el operador ***sizeof***, para determinar así la longitud (*n_bytes*) de cada elemento a leer o escribir.

El ejemplo anterior puede servir para ejemplificar ahora el uso de esas dos funciones. El archivo "*numeros.dat*" será ahora de tipo binario. El programa cargará en forma binaria esos valores y luego los leerá para calcular el valor medio de todos ellos y mostrarlos por pantalla:

```
#include <stdio.h>
#include <stdlib.h>
#define TAM 10
void main(void)
{
      float or[TAM], cp[TAM];
      double suma = 0;
      short i;
      FILE *ARCH;

      randomize();
      for(i = 0 ;  i < TAM ; i++)
            or[i] = (float)random(1000) / random(100);

      ARCH = fopen("numeros.dat", "wb");
      if(ARCH == NULL)
      {
            printf("No se ha podido abrir el archivo.\n");
            printf("Pulse una tecla para finalizar... ");
            getchar();
            exit(1);
      }
      fwrite(or,sizeof(float),TAM,ARCH);
      fclose(ARCH);

      ARCH = fopen("numeros.dat", "rb");
      if(ARCH == NULL)
      {
            printf("No se ha podido abrir el archivo.\n");
            printf("Pulse una tecla para finalizar... ");
            getchar();
            exit(1);
      }
      fread(cp,sizeof(float),TAM,ARCH);
```

```
        fclose(ARCH);

        for(i = 0 ; i < TAM ; i++)
        {
              printf("Valor %04hd --> %12.4f\n",i,cp[i]);
              suma += *(cp + i);
        }
        printf("\n\nLa media es ... %lf", suma / TAM);
}
```

- Otras funciones útiles en el acceso a archivo

  Función ***feof***: Esta función (en realidad es una macro) determina el final de archivo. Es conveniente usarla cuando se trabaja con archivos binarios, donde se puede inducir a error y tomar como carácter EOF un valor entero codificado.

  Su prototipo es:

  int feof(FILE *nombre_archivo);

  que devuelve un valor diferente de cero si en la última operación de lectura se ha detectado el valor EOF. en caso contrario devuelve el valor cero.

  Función ***ferror***: Esta función (en realidad es una macro) determina si se ha producido un error en la última operación sobre el archivo. Su prototipo es:

  int ferror(FILE * nombre_archivo);

  Si el valor devuelto es diferente de cero, entonces se ha producido un error; si es igual a cero, entonces no se ha producido error alguno.

  Si deseamos hacer un programa que controle perfectamente todos los accesos a disco, entonces convendrá ejecutar esta función después de cada operación de lectura o escritura.

  Función ***remove***: Esta función elimina un archivo. El archivo será cerrado si estaba abierto y luego será eliminado. Quiere esto decir que el archivo quedará destruido, que no es lo mismo que quedarse vacío.

  Su prototipo es:

int remove(const char * nombre_archivo);

Donde *nombre_archivo* es el nombre del archivo que se desea borrar. En ese nombre, como siempre, debe ir bien consignada la ruta completa del archivo. Un archivo así eliminado no es recuperable.

Por ejemplo, en nuestro ejemplos anteriores, después de haber hecho la transferencia de datos al vector de ***float***, podríamos ya eliminar el archivo de nuestro disco. Hubiera abastado poner la sentencia:

```
remove("numeros.dat");
```

Si el archivo no ha podido ser eliminado (por denegación de permiso o porque el archivo no existe en la ruta y nombre que ha dado el programa) entonces la función devuelve el valor -1. Si la operación de eliminación del archivo ha sido correcta, entonces devuelve un cero.

En realidad, la macro *remove* lo único que hace es invocar a la función de borrado definida en **io.h**: la función ***unlink***, cuyo prototipo es:

int unlink(const char *filename);

Y cuyo comportamiento es idéntico al explicado para la macro *remove*.

Entrada y salida sobre archivos de acceso aleatorio

Disponemos de algunas funciones que permiten acceder de forma aleatoria a una u otra posición del archivo.

Ya dijimos que un archivo, desde e punto de vista del programador es simplemente un puntero a la posición del archivo (en realidad al buffer) donde va a tener lugar el próximo acceso al archivo. Cuando se abre el archivo ese puntero recoge la dirección de la posición cero del archivo, es decir, al principio. Cada vez que el programa indica escritura de datos, el puntero termina ubicado al final del archivo.

Pero también podemos, gracias a algunas funciones definidas en **io.h**, hacer algunos accesos aleatorios. En realidad, el único elemento nuevo

que se incorpora al hablar de acceso aleatorio es una función capaz de posicionar el puntero del archivo devuelto por la función *fopen* en distintas partes del fichero y poder así acceder a datos intermedios.

La función ***fseek*** puede modificar el valor de ese puntero, llevándolo hasta cualquier byte del archivo y logrando así un acceso aleatorio. Es decir, que las funciones estándares de ANSI C logran hacer accesos aleatorios únicamente mediante una función que se añade a todas las que ya hemos visto para los accesos secuenciales.

El prototipo de la función, definida en la biblioteca **stdio.h** es el siguiente:

int fseek(FILE *nomnre_archivo, long despl, int modo);

Donde ***nombre_archivo*** es el puntero que ha devuelto la función *fopen* al abrir el archivo; donde **despl** es el desplazamiento, en bytes, a efectuar; y donde ***modo*** es el punto de referencia que se toma para efectuar el desplazamiento. Para esa definición de modo, **stdio.h** define tres constantes diferentes:

SEEK_SET, que es valor 0.

SEEK_CUR, que es valor 1,

SEEK_END, que es valor 2.

El modo de la función ***fseek*** puede tomar como valor cualquiera de las tres constantes. Si tiene la primera (SEEK_SET), el desplazamiento se hará a partir del inicio del fichero; si tiene la segunda (SEEK_CUR), el desplazamiento se hará a partir de la posición actual del puntero; si tiene la tercera (SEEK_END), el desplazamiento se hará a partir del final del fichero.

Para la lectura del archivo que habíamos visto para ejemplificar la función *fscanf*, las sentencias de lectura quedarían mejor si se hiciera así:

```
printf("Los valores guardados en el archivo son:\n");
i = 0;
```

```
while(!feof(ARCH))
{
      fseek(ARCH,16,SEEK_CUR);
      fscanf(ARCH,"%f",cp + i++);
}
```

Donde hemos indicado 16 en el desplazamiento en bytes, porque 16 son los caracteres que no deseamos que se lean en cada línea.

Los desplazamientos en la función *fseek* pueden ser positivos o negativos. Desde luego, si los hacemos desde el principio lo razonable es hacerlos positivos, y si los hacemos desde el final hacerlos negativos. La función acepta cualquier desplazamiento y no produce nunca un error. Luego, si el desplazamiento ha sido erróneo, y nos hemos posicionado en medio de ninguna parte o en un byte a mitad de dato, entonces la lectura que pueda hacer la función que utilicemos será imprevisible.

Una última función que presentamos en este capítulo es la llamada ***rewind***, cuyo prototipo es:

void rewind(FILE *nombre_archivo);

Que "rebobina" el archivo, devolviendo el puntero a su posición inicial, al principio del archivo.

Ejercicios

69. ***Descargar desde Internet un archivo con el texto completo de El Quijote. Almacenarlo en formato texto. Darle a este archivo el nombre quijote.txt. Y hacer entonces un programa que vaya leyendo uno a uno los caracteres del archivo y vaya contando cuántas veces aparece cada una de las letras del abecedario. Mostrar al final en pantalla las veces que han aparecido cada una de las letras y también el porcentaje de aparición respecto***

al total de todas las letras aparecidas.

Vamos a ofrecer dos soluciones a este programa. La primera es la más trivial:

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
void main(void)
{
      long letra[27];
      short caracter;
      long suma = 0;
      short int i;
      FILE *archivo;

      for(i = 0 ; i < 26 ; i++) letra[i] = 0;

      archivo = fopen("quijote.txt", "r");
      if(archivo == NULL)
      {
            printf("No se ha podido abrir el archivo.\n");
            printf("Pulse una tecla para finalizar... ");
            getchar();
            exit(1);
      }
      while((caracter = fgetc(archivo)) != EOF)
      {
            if(isalpha(caracter))
            {
                  i = (short)tolower(caracter) - (short)'a';
                  if(i >= 0 && i < 26) letra[i]++;
            }
      }

      fclose(archivo);
      for(i = 0 ; i < 26 ; i++) suma += letra[i];
      for(i = 0 ; i < 26 ; i++)
      {
            printf("[ %c ]= %10ld\t",(char)(i+'A'),letra[i]);
            printf("%7.2lf\n",((float)letra[i]/suma)*100);
      }

      printf("\n\nTotal letras ... %ld",suma);
}
```

Esta es la solución primera y sencilla. El vector letras tiene 26 elementos: tantos como letras tiene el abecedario ASCII. Pasamos siempre la letra a minúscula porque así no hemos de verificar que nos venga el carácter en mayúscula, y nos ahorramos muchas comparaciones. El vector letra se indexa siempre por medio de la variable *i*.

La pega es que con este código no sumamos las veces que aparecen las vocales con acento, o la letra 'u' con diéresis. Y, desde luego, no calculamos cuántas veces aparece la letra 'Ñ' ó la letra 'ñ'. Para poder hacer esos cálculos, deberemos modificar el programa añadiendo algunas instrucciones:

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
void main(void)
{
	long letra[27];
	short caracter;
	long suma = 0;
	short int i;
	FILE *archivo;

	for(i = 0 ; i < 27 ; i++) letra[i] = 0;

	archivo = fopen("quijote.txt", "r");
	if(archivo == NULL)
	{
		printf("No se ha podido abrir el archivo.\n");
		printf("Pulse una tecla para finalizar... ");
		getchar();
		exit(1);
	}
	while((caracter = fgetc(archivo)) != EOF)
	{
		if(caracter == 209 || caracter == 241)
			letra[26]++; // letras ñ y Ñ
		else if(caracter == 225 || caracter == 193)
			letra['a' - 'a']++; // letras á y Á
		else if(caracter == 233 || caracter == 201)
			letra['e' - 'a']++; // letras é y É
		else if(caracter == 237 || caracter == 205)
			letra['i' - 'a']++; // letras í e Í
		else if(caracter == 243 || caracter == 211)
			letra['o' - 'a']++; // letras ó y Ó
```

```
            else if(caracter == 250 || caracter == 218)
                  letra['u' - 'a']++; // letras ú y  Ú
            else if(caracter == 252 || caracter == 220)
                  letra['u' - 'a']++; // letras ü y Ü
            else if(isalpha(caracter))
            {
                  i = (short)tolower(caracter) - (short)'a';
                  if(i >= 0 && i < 26) letra[i]++;
            }
      }

      fclose(archivo);
      for(i = 0 ; i < 27 ; i++) suma += letra[i];
      for(i = 0 ; i < 26 ; i++)
      {
            printf("[ %c ]= %10ld\t",(char)(i+'A'),letra[i]);
            printf("%7.2lf\n",((float)letra[i]/suma)*100);
      }
      printf("[ %c ] = %10ld\t", 165,letra[26]);
      printf("%7.2lf\n",((float)letra[26] / suma) * 100);

      printf("\n\nTotal letras ... %ld",suma);
}
```

70. ***Implementar una base de datos de asignaturas. El programa será muy sencillo, y simplemente debe definir una estructura como la que ya estaba definida en un tema anterior. El programa almacenará en disco y añadirá al final de archivo cada una de las nuevas asignaturas que se añadan. La información se guardará en binario. Se ofrecerá la posibilidad de realizar un listado de todas las asignaturas por pantalla o grabando ese listado en disco, creando un documento que se pueda luego tratar con un programa editor de texto.***

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

typedef struct
{
      unsigned long clave;
```

```
      char descr[50];
      double cred;
}asignatura;

short mostrar_opciones(void);
void error(void);
short anyadir(char*);
short pantalla(char*);
short impresora(char*);

void main(void)
{
      char nombre_archivo[80];
      short opcion;
      short oK;
      printf("Nombre del archivo de asignaturas ... ");
      gets(nombre_archivo);
      do
      {
            opcion = mostrar_opciones();
            switch(opcion)
            {
            case '1':   oK = anyadir(nombre_archivo);
                        if(oK) error();
                        break;
            case '2':   oK = pantalla(nombre_archivo);
                        if(oK) error();
                        break;
            case '3':   oK = impresora(nombre_archivo);
                        if(oK) error();
                        break;
            case '4':   exit(1);
            }
      }while(1);
}

short mostrar_opciones(void)
{
      char opcion;
      clrscr();
      printf("\n\n\t\tOpciones y Tareas");
      printf("\n\n\t1. Añadir nueva asignatura.");
      printf("\n\t2. Mostrar listado por pantalla.");
      printf("\n\t3. Mostrar listado en archivo.");
      printf("\n\t4. Salir del programa.");
      printf("\n\n\t\t\tElegir opcion ... ");
      do opcion = getchar(); while(opcion <'0'&&opcion >'4');
      return opcion;
}

void error(void)
```

```
{
      printf("Error en la operacion de acceso disco.\n");
      printf("Pulse una tecla para terminar ... \n");
      getchar();
      exit(1);
}

short anyadir(char archivo[])
{
      FILE *ARCH;
      asignatura asig;
      printf("\n\n\nDATOS DE LA NUEVA ASIGNATURA.\n\n");
      printf("clave de la asignatura ... ");
      scanf("%lu",&asig.clave);
      printf("\nDescripcion ... ");
      flushall();
      gets(asig.descr);
      printf("\nCreditos ..... ");
      scanf("%lf",&asig.cred);
      ARCH = fopen(archivo,"ab");
      fwrite(&asig,sizeof(asig),1,ARCH);
      printf("\n\n\tPulsar una tecla para continuar ... ");
      getchar();
      if(ferror(ARCH)) return 1;
      fclose(ARCH);
      return 0;
}

short pantalla(char archivo[])
{
      FILE *ARCH;
      asignatura asig;

      ARCH = fopen(archivo,"a+b");
      rewind(ARCH);
      while(fread(&asig,sizeof(asig),1,ARCH) == 1)
      {
            printf("\n\nClave ......... %lu",asig.clave);
            printf("\nDescripcion ... %s",asig.descr);
            printf("\nCreditos ..... %6.1lf",asig.cred);
      }
      printf("\n\n\tPulsar una tecla para continuar ... ");
      getchar();
      if(ferror(ARCH)) return 1;
      fclose(ARCH);
      return 0;
}

short impresora(char archivo[])
{
      FILE *ARCH1, *ARCH2;
```

```
        asignatura asig;

        ARCH1 = fopen(archivo,"rb");
        ARCH2 = fopen("impresora","w");
        while(fread(&asig,sizeof(asig),1,ARCH1) == 1)
        {
              fprintf(ARCH2,"\n\nClave\t%lu", asig.clave);
              fprintf(ARCH2,"\nDescripcion \t%s", asig.descr);
              fprintf(ARCH2,"\nCreditos\t%6.1lf", asig.cred);
        }
        printf("\n\n\tPulsar una tecla para continuar ... ");
        getchar();
        if(ferror(ARCH1)) return 1;
        fclose(ARCH1);
        fclose(ARCH2);
        return 0;
}
```

La función principal presenta únicamente una estructura ***switch*** que gestiona cuatro posibles valores para la variable *opcion*. Esos valores se muestran en la primera de las funciones, la función ***mostrar_opciones***, que imprime en pantalla las cuatro posibles opciones del programa, (añadir registros, mostrarlos por pantalla, crear un archivo de impresión, y salir del programa) y devuelve a la función principal el valor de la opción elegida.

La función ***anyadir*** recoge los valores de una nueva asignatura y guarda la información, mediante la función *fwrite*, en el archivo que ha indicado el usuario al comenzar la ejecución del programa. El archivo se abre para añadir y para codificación binaria: "ab". En esta función se invoca a otra, llamada ***flushall***. Esta función, de la biblioteca **stdio.h**, vacía todos los buffers de entrada. La ejecutamos antes de la función *gets* para variar el buffer de teclado. A veces ese buffer contiene algún carácter, o el carácter intro pulsado desde la última entrada de datos por teclado, y el sistema operativo lo toma como entrada de a función *gets*, que queda ejecutada sin intervención del usuario.

La función ***pantalla*** muestra por pantalla un listado de las asignaturas introducidas hasta el momento y guardadas en el archivo. Abre el archivo para lectura en formato binario: "a+b". No lo hemos abierto

como "rb" para evitar el error en caso de que el usuario quiera leer un archivo inexistente.

La función ***impresora*** hace lo mismo que ***pantalla***, pero en lugar de mostrar los datos por la consola los graba en un archivo de texto. Por eso esa función abre dos archivos y va grabando el texto en el archivo abierto como "w". Si el archivo "*impresora*" ya existe, entonces es eliminado y crea otro en su lugar.

Se puede completar el programa con nuevas opciones. Se podría modificar la función ***mostrar_opciones*** y la función ***main*** incorporando esas opciones nuevas. Y se crearían las funciones necesarias para esas nuevas tareas. Por ejemplo: eliminar el archivo de asignaturas; hacer una copia de seguridad del archivo; buscar una asignatura en el archivo cuya clave sea la que indique el usuario y, si la encuentra, entonces muestre por pantalla la descripción y el número de créditos; etc.