6. Manejo de ficheros

6.1. Escritura en un fichero de texto

Para manejar ficheros, siempre deberemos realizar tres operaciones básicas:

- Abrir el fichero.
- Leer datos de él o escribir datos en él.
- Cerrar el fichero.

Eso sí, no siempre podremos realizar esas operaciones, así que además tendremos que comprobar los posibles errores. Por ejemplo, puede ocurrir que intentemos abrir un fichero que realmente no exista, o que queramos escribir en un dispositivo que sea sólo de lectura.

Vamos a ver un ejemplo, que cree un fichero de texto y escriba algo en él:

```
/*---------------------------*/
/*  Ejemplo en C nº 55:      */
/*  c055.c                   */
/*                           */
/*  Escritura en un fichero  */
/*  de texto                 */
/*                           */
/*  Curso de C,              */
/*    Nacho Cabanes          */
/*---------------------------*/

#include <stdio.h>

main()
{
    FILE* fichero;

    fichero = fopen("prueba.txt", "wt");
    fputs("Esto es una línea\n", fichero);
    fputs("Esto es otra", fichero);
    fputs(" y esto es continuación de la anterior\n", fichero);
    fclose(fichero);
}
```

Hay varias cosas que comentar sobre este programa:

- **FILE** es el tipo de datos asociado a un fichero. Siempre aparecerá el asterisco a su derecha, por motivos que veremos más adelante (cuando hablemos de "punteros").
- Para abrir el fichero usamos "**fopen**", que necesita dos datos: el nombre del fichero y el modo de lectura. El modo de lectura estará formado por varias letras, de las cuales por ahora nos interesan dos: "w" indicaría que queremos escribir (*write*) del fichero, y "t" avisa de que se trata de un fichero de texto (*text*). Como abrimos el fichero para escribir en él, se creará el fichero si no existía, y **se borrará** su contenido si ya existía (más adelante veremos cómo añadir a un fichero sin borrar su contenido).
- Para **escribir** en el fichero y para leer de él, tendremos órdenes muy parecidas a las que usábamos en pantalla. Por ejemplo, para escribir una cadena de texto usaremos

"fputs", que recuerda mucho a "puts" pero con la diferencia de que no avanza de línea después de cada texto (por eso hemos añadido \n al final de cada frase).

- Finalmente, **cerramos** el fichero con "fclose".

Ejercicios propuestos:

- Crea un programa que vaya leyendo las frases que el usuario teclea y las guarde en un fichero de texto llamado "registroDeUsuario.txt". Terminará cuando la frase introducida sea "fin" (esa frase no deberá guardarse en el fichero).

6.2. Lectura de un fichero de texto

Si queremos **leer de un fichero**, los pasos son muy parecidos, sólo que lo abriremos para lectura (el modo de escritura tendrá una "r", de "read", en lugar de "w"), y leeremos con "fgets":

```
/*----------------------------*/
/*  Ejemplo en C nº 56:       */
/*  c056.c                    */
/*                            */
/*  Lectura de un fichero de  */
/*  texto                     */
/*                            */
/*  Curso de C,               */
/*    Nacho Cabanes           */
/*----------------------------*/

#include <stdio.h>

main()
{
    FILE* fichero;
    char nombre[80] = "c:\\autoexec.bat";
    char linea[81];

    fichero = fopen(nombre, "rt");

    if (fichero == NULL)
    {
      printf("No existe el fichero!\n");
      exit(1);
    }
    fgets(linea, 80, fichero);
    puts(linea);
    fclose(fichero);
}
```

En este fuente hay un par de cambios:

- En el nombre del fichero, hemos indicado un nombre algo más complejo. En estos casos, hay que recordar que si aparece alguna barra invertida (\), deberemos duplicarla, porque la barra invertida se usa para indicar ciertos códigos de control. Por ejemplo, \n es el código de avance de línea y \a es un pitido. El modo de lectura en este caso es "r" para indicar que queremos leer (*read*) del fichero, y "t" avisa de que es un fichero de texto.

- Para **leer** del fichero y usaremos "fgets", que se parece mucho a "gets", pero podemos limitar la longitud del texto que leemos (en este ejemplo, a 80 caracteres) desde el fichero. Esta cadena de texto **conservará** los caracteres de avance de línea.
- Si **no se consigue** abrir el fichero, se nos devolverá un valor especial llamado NULL (que también veremos con mayor detalle más adelante, cuando hablemos de punteros).
- La orden "**exit**" es la que nos permite abandonar el programa en un punto. La veremos con más detalle un poco más adelante.

Ejercicios propuestos:

- Crea un programa que lea las tres primeras líneas del fichero creado en el apartado anterior y las muestre en pantalla. Si el fichero no existe, se deberá mostrar un aviso.

6.3. Lectura hasta el final del fichero

Normalmente no querremos leer sólo una frase del fichero, sino procesar todo su contenido. Para ayudarnos, tenemos una orden que nos permite saber si ya hemos llegado al final del fichero. Es "feof" (EOF es la abreviatura de End Of File, fin de fichero).

Por tanto, nuestro programa deberá repetirse **mientras que no se acabe** el fichero, así:

```
/*---------------------------*/
/*  Ejemplo en C nº 57:      */
/*  c057.c                   */
/*                           */
/*  Lectura hasta el final   */
/*  de un fichero de texto   */
/*                           */
/*  Curso de C,              */
/*    Nacho Cabanes          */
/*---------------------------*/

#include <stdio.h>

main()
{
    FILE* fichero;
    char nombre[80] = "c:\\autoexec.bat";
    char linea[81];

    fichero = fopen(nombre, "rt");

    if (fichero == NULL)
    {
      printf("No existe el fichero!\n");
      exit(1);
    }
    while (! feof(fichero)) {
      fgets(linea, 80, fichero);
      puts(linea);
    }
    fclose(fichero);
}
```

Esa será la estructura básica de casi cualquier programa que deba leer un fichero completo, de principio a fin: abrir, comprobar que se ha podido acceder correctamente, leer con "while !(feof(…))" y cerrar.

Ejercicios propuestos:

- Un programa que pida al usuario que teclee frases, y las almacene en el fichero "frases.txt". Acabará cuando el usuario pulse Intro sin teclear nada. Después deberá mostrar el contenido del fichero.
- Un programa que pregunte un nombre de fichero y muestre en pantalla el contenido de ese fichero, haciendo una pausa después de cada 25 líneas, para que dé tiempo a leerlo. Cuando el usuario pulse intro, se mostrarán las siguientes 25 líneas, y así hasta que termine el fichero. (Pista: puedes usar un contador, volverlo a poner a cero tras cada 25 líneas o bien comprobar si has avanzado otras 25 líneas usando la operación "resto de la división", y hacer la pausa con "getchar()").

6.4. Ficheros con tipo

Es frecuente que los ficheros que queramos manejar no sean de texto, pero que aun así tengan un formato bastante definido. Por ejemplo, podemos querer crear una agenda, en la que los datos de cada persona estén guardados en un "struct". En este caso, podríamos guardar los datos usando "fprintf" y "fscanf", análogos a "printf" y "scanf" que ya conocemos.

```
fprintf( fichero, "%40s%5d\n", persona.nombre, persona.numero);
fscanf( fichero, "%40s%5d\n", &persona.nombre, &persona.numero);
```

Como se puede ver en este ejemplo, suele ser recomendable indicar la anchura que debe tener cada dato cuando guardamos con "fprintf", para que se pueda recuperar después de la misma forma con "fscanf".

Aun así, "fscanf" tiene el mismo problema que "scanf": si leemos una cadena de texto, la considera terminada después del primer espacio en blanco, y lo que haya a continuación lo asignará a la siguiente cadena. Por eso, cuando manejemos textos con espacios, será preferible usar "fgets" o bien otras dos órdenes para manejo de ficheros que veremos un poco más adelante.

Ejercicios propuestos:

- Crear un "struct" que almacene los siguientes datos de una persona: nombre, edad, ciudad de residencia. Pedir al usuario esos datos de una persona y guardarlos en un fichero llamado "gente.dat". Cerrar el fichero, volverlo a abrir para lectura y mostrar los datos que se habían guardado.
- Ampliar el programa anterior para que use un "array de structs", de forma que se puedan tener datos de 10 personas. Se deberá pedir al usuario los datos de las 10 personas y guardarlos en el fichero. Después se pedirá al usuario un número del 1 al 10 y se mostrarán los datos de la persona indicada por ese número, que se deberán leer de fichero (1 será la primera ficha, y 10 será la última). Por ejemplo, si el usuario indica que quiere ver los datos de la persona 3 (tercera), se deberá leer las dos primeras, ignorando su contenido, y después leer la tercera, que sí se deberá mostrar.
- Una agenda que maneje los siguientes datos: nombre, dirección, tlf móvil, email, y día, mes y año de nacimiento (estos tres últimos datos deberán ser números enteros

cortos). Deberá tener capacidad para 100 fichas. Se deberá poder añadir un dato nuevo, visualizar los nombres de las fichas existentes, o mostrar todos los datos de una persona (se preguntará al usuario cual es el nombre de esa persona que quiere visualizar). Al empezar el programa, leerá los datos de un fichero llamado "agenda.dat" (si existe). Al terminar, guardará todos los datos en ese fichero.

6.5 Leer y escribir letra a letra

Si queremos leer o escribir **sólo una letra**, tenemos las órdenes "fgetc" y "fputc", que se usan:

```
letra = fgetc( fichero );
fputc (letra, fichero);
```

6.6 Modos de apertura

Antes de seguir, vamos a ver las letras que pueden aparecer en el **modo de apertura** del fichero, para poder añadir datos a un fichero ya existente:

Tipo	Significado
r	Abrir sólo para lectura.
w	Crear para escribir. Sobreescribe el fichero si existiera ya (borrando el original).
a	Añade al final del fichero si existe, o lo crea si no existe.
+	Se escribe a continuación de los modos anteriores para indicar que también queremos modificar. Por ejemplo: r+ permite leer y modificar el fichero.
t	Abrir en modo de texto.
b	Abrir en modo binario.

Ejercicios propuestos:

- Un programa que pida al usuario que teclee frases, y las almacene en el fichero "registro.txt", que puede existir anteriormente (y que no deberá borrarse, sino añadir al final de su contenido). Cada sesión acabará cuando el usuario pulse Intro sin teclear nada.
- Crear un programa que pida al usuario pares de números enteros y escriba su suma (con el formato "20 + 3 = 23") en pantalla y en un fichero llamado "sumas.txt", que se encontrará en un subdirectorio llamado "resultados". Cada vez que se ejecute el programa, deberá añadir los nuevos resultados a continuación de los resultados de las ejecuciones anteriores.

6.7 Ficheros binarios

Hasta ahora nos hemos centrado en los ficheros de texto, que son sencillos de crear y de leer. Pero también podemos manejar ficheros que contengan información de cualquier tipo.

En este caso, utilizamos "fread" para leer un bloque de datos y "fwrite" para guardar un bloque de datos. Estos datos que leamos se guardan en un **buffer** (una zona intermedia de memoria). En el momento en que se lean menos bytes de los que hemos pedido, quiere decir que hemos llegado al final del fichero.

En general, el manejo de "fread" es el siguiente:

```
cantidadLeida = fread(donde, tamañoDeCadaDato, cuantosDatos, fichero);
```

Por ejemplo, para leer 10 números enteros de un fichero (cada uno de los cuales ocuparía 4 bytes, si estamos en un sistema operativo de 32 bits), haríamos

```
int datos[10];
resultado = fread(&datos, 4, 10, fichero);
if (resultado < 10)
  printf("Había menos de 10 datos!");
```

Al igual que ocurría con "scanf", la variable en la que guardemos los datos se deberá indicar precedida del símbolo &, por motivos con detalle que veremos cuando hablemos sobre punteros. También al igual que pasaba con "scanf", si se trata de una cadena de caracteres (bien porque vayamos a leer una cadena de texto, o bien porque queramos leer datos de cualquier tipo pero con la intención de manejarlos byte a byte), como `char dato[500]` no será necesario indicar ese símbolo &, como en este ejemplo:

```
char cabecera [40];
resultado = fread(cabecera, 1, 40, fichero);
if (resultado < 40)
  printf("Formato de fichero incorrecto, no está toda la cabecera!");
else
 printf("El byte en la posición 5 es un %d", cabecera[4]);
```

6.8 Ejemplo: copiador de ficheros

Vamos a ver un ejemplo, que duplique un fichero de cualquier tipo (no necesariamente de texto), y después veremos las novedades:

```
/*---------------------------*/
/*  Ejemplo en C nº 58:      */
/*  c058.c                   */
/*                           */
/*  Copiador de ficheros     */
/*                           */
/*  Curso de C,              */
/*    Nacho Cabanes          */
/*---------------------------*/

#include <stdio.h>

FILE *fichOrg, *fichDest;        /* Los dos ficheros */
char buffer[2048];               /* El buffer para guardar lo que leo */
char nombreOrg[80],              /* Los nombres de los ficheros */
```

```
nombreDest[80];
int cantidad;                    /* El número de bytes leídos */

main()
{
    /* Accedo al fichero de origen */
    printf("Introduzca el nombre del fichero Origen: ");
    scanf("%s",nombreOrg);
    if ((fichOrg = fopen(nombreOrg, "rb")) == NULL)
    {
      printf("No existe el fichero origen!\n");
      exit(1);

    }
    /* Y ahora al de destino */
    printf("Introduzca el nombre del fichero Destino: ");
    scanf("%s",nombreDest);
    if ((fichDest = fopen(nombreDest, "wb")) == NULL)
    {
      printf("No se ha podido crear el fichero destino!\n");
      exit(1);
    }
    /* Mientras quede algo que leer */
    while (! feof(fichOrg) )
    {
      /* Leo datos: cada uno de 1 byte, todos los que me caben */
      cantidad = fread( buffer, 1, sizeof(buffer), fichOrg);
      /* Escribo tantos como haya leído */
      fwrite(buffer, 1, cantidad, fichDest);
    }
    /* Cierro los ficheros */
    fclose(fichOrg);
    fclose(fichDest);
}
```

Los cambios con relación a lo que conocíamos de ficheros de texto son los siguientes:

- Los ficheros pueden no ser de texto, de modo que leemos uno como fichero binario (con "rb") y escribimos el otro también como fichero binario (con "wb").

- Definimos un buffer de 2048 bytes (2 K), para ir leyendo la información por bloques (y guardando después cada bloque en el otro fichero).

- En la misma línea intentamos abrir el fichero y comprobamos si todo ha sido correcto, con

  ```
  if ((fichOrg = fopen(nombreOrg, "rb")) == NULL)
  ```

  Esto puede resultar menos legible que hacerlo en dos líneas separadas, como hemos hecho hasta ahora, pero es más compacto, y, sobre todo, muy frecuente encontrarlo en "fuentes ajenos" más avanzados, como los que se puedan encontrar en Internet o cuando se programe en grupo con otras personas, de modo que he considerado adecuado incluirlo.

- A "fread" le decimos que queremos leer 2048 datos de 1 byte cada uno, y él nos devuelve la cantidad de bytes que ha leído realmente. Para que el fuente sea más fácil

de aplicar a otros casos en los que no sean bloques de 2048 bytes exactamente, suele ser preferible indicar que queremos leer el tamaño del bloque, usando "sizeof":

```
cantidad = fread( buffer, 1, sizeof(buffer), fichOrg);
```

Cuando la cantidad leida sea menos de 2048 bytes, es que el fichero se ha acabado (lo podemos comprobar mirando esta cantidad o con "feof").

- "fwrite" se maneja igual que fread: se le indica dónde están los datos, el tamaño de cada dato, cuantos datos hay que escribir y en qué fichero almacenarlos. En nuestro ejemplo, el número de bytes que debe escribir será el que haya leido:

```
fwrite(buffer, 1, cantidad, fichDest);
```

Ejercicios propuestos:

- Mejorar la agenda anterior, para guardar y leer cada "ficha" (struct) de una vez, usando fwrite/fread y sizeof, como en el último ejemplo.
- Crear un "struct" que almacene los siguientes datos de una persona: nombre, edad, ciudad de residencia. Pedir al usuario esos datos de una persona y guardarlos en un fichero llamado "gente.dat", usando "fwrite". Cerrar el fichero, volverlo a abrir para lectura y mostrar los datos que se habían guardado, que se deben leer con "fread".
- Ampliar el programa anterior para que use un "array de structs", de forma que se puedan tener datos de 10 personas. Se deberá pedir al usuario los datos de las 10 personas y guardarlos en el fichero, usando "fwrite". Después se pedirá al usuario un número del 1 al 10 y se mostrarán los datos de la persona indicada por ese número, que se deberán leer de fichero (1 será la primera ficha, y 10 será la última). Por ejemplo, si el usuario indica que quiere ver los datos de la persona 3 (tercera), se deberá leer las dos primeras (con "fread"), ignorando su contenido, y después leer la tercera, que sí se deberá mostrar.

6.9 Acceder a cualquier posición de un fichero

Cuando trabajamos con un fichero, es posible que necesitemos **acceder** directamente a una cierta posición del mismo. Para ello usamos "**fseek**", que tiene el formato:

```
int fseek(FILE *fichero, long posicion, int desde);
```

Como siempre, comentemos qué es cada cosa:

- Es de tipo "int", lo que quiere decir que nos va a devolver un valor, para que comprobemos si realmente se ha podido saltar a la dirección que nosotros le hemos pedido: si el valor es 0, todo ha ido bien; si es otro, indicará un error (normalmente, que no hemos abierto el fichero).
- "fichero" indica el fichero dentro de el que queremos saltar. Este fichero debe estar abierto previamente (con fopen).
- "posición" nos permite decir a qué posición queremos saltar (por ejemplo, a la 5010).
- "desde" es para poder afinar más: la dirección que hemos indicado con posic puede estar referida al comienzo del fichero, a la posición en la que nos encontramos

actualmente, o al final del fichero (entonces posic deberá ser negativo). Para no tener que recordar que un 0 quiere decir que nos referimos al principio, un 1 a la posición actual y un 2 a la final, tenemos definidas las siguientes **constantes**:

```
SEEK_SET (0):  Principio
SEEK_CUR (1):  Actual
SEEK_END (2):  Final
```

Vamos a ver tres ejemplos de su uso:

- Ir a la posición 10 del fichero: `fseek(miFichero, 10, SEEK_SET);`
- Avanzar 5 posiciones a partir de la actual: `fseek(miFichero, 5, SEEK_CUR);`
- Ir a la posición 8 antes del final del fichero: `fseek(miFichero, -8, SEEK_END);`

Finalmente, si queremos saber en **qué posición** de un fichero nos encontramos, podemos usar "**ftell**(fichero)".

Esta orden nos permite saber también la **longitud** de un fichero: nos posicionamos primero al final con "fseek" y luego comprobamos con "ftell" en qué posición estamos:

```
fseek(fichero, 0, SEEK_END);
longitud = ftell(fichero);
```

Ejercicios propuestos:

- Ampliar el programa anterior (el "array de structs" con 10 personas) para que el dato que indique el usuario se lea sin leer y descartar antes los que le preceden, sino que se salte directamente a la ficha deseada usando "fseek".

6.10 Ejemplo: leer información de un fichero BMP

Ahora vamos a ver un ejemplo un poco más sofisticado: vamos a abrir un fichero que sea una imagen en formato BMP y a mostrar en pantalla si está comprimido o no.

Para eso necesitamos antes saber cómo se guarda la información en un fichero BMP, pero esto es algo fácil de localizar:

FICHEROS .BMP

Un fichero BMP está compuesto por las siguientes partes: una cabecera de fichero, una cabecera del bitmap, una tabla de colores y los bytes que definirán la imagen.

En concreto, los datos que forman la cabecera de fiochero y la cabecera de bitmap son los siguientes:

TIPO DE INFORMACIÓN	POSICIÓN EN EL ARCHIVO
Tipo de fichero (letras BM)	0-1
Tamaño del archivo	2-5
Reservado	6-7

Reservado	8-9
Inicio de los datos de la imagen	10-13
Tamaño de la cabecera de bitmap	14-17
Anchura (píxeles)	18-21
Altura (píxeles)	22-25
Número de planos	26-27
Tamaño de cada punto	28-29
Compresión (0=no comprimido)	30-33
Tamaño de la imagen	34-37
Resolución horizontal	38-41
Resolución vertical	42-45
Tamaño de la tabla de color	46-49
Contador de colores importantes	50-53

Con esta información nos basta para nuestro propósito: la compresión se indica en la posición 30 del fichero, ocupa 4 bytes (lo mismo que un "int" en los sistemas operativos de 32 bits), y si es un 0 querrá decir que la imagen no está comprimida.

Entonces lo podríamos comprobar así:

```
/*----------------------------*/
/*  Ejemplo en C nº 59:       */
/*  c059.c                    */
/*                            */
/*  Información sobre un      */
/*  fichero BMP (1)           */
/*                            */
/*  Curso de C,               */
/*    Nacho Cabanes           */
/*----------------------------*/

#include <stdio.h>

main(){
  char nombre[60];
  FILE* fichero;
  int compresion;

  puts("Comprobador de imágenes BMP\n");
  printf("Dime el nombre del fichero: ");
  gets(nombre);
  fichero = fopen(nombre, "rb");
  if (fichero==NULL)
    puts("No encontrado\n");
  else {
    fseek(fichero, 30, SEEK_SET);
    fread(&compresion, 1, 4, fichero);
    fclose(fichero);
    if (compression == 0)
      puts("Sin compresión");
   else
      puts ("BMP Comprimido ");
  }
}
```

Ya que estamos, podemos mejorarlo un poco para que además nos muestre el ancho y el alto de la imagen, y que compruebe antes si realmente se trata de un fichero BMP:

```
/*---------------------------*/
/*  Ejemplo en C nº 60:      */
/*  c060.c                   */
/*                           */
/*  Información sobre un     */
/*  fichero BMP (2)          */
/*                           */
/*  Curso de C,              */
/*    Nacho Cabanes          */
/*---------------------------*/

#include <stdio.h>

main(){
  char nombre[60];
  FILE* fichero;
  int compresion, ancho, alto;
  char marca1, marca2;

  puts("Comprobador de imágenes BMP\n");
  printf("Dime el nombre del fichero: ");
  gets(nombre);
  fichero = fopen(nombre, "rb");
  if (fichero==NULL)
    puts("No encontrado\n");
  else {
    marca1 = fgetc(fichero);  /* Leo los dos primeros bytes */
    marca2 = fgetc(fichero);
    if ((marca1 =='B') && (marca2 =='M')) { /* Si son BM */
        printf("Marca del fichero: %c%c\n",
          marca1, marca2);
        fseek(fichero, 18, SEEK_SET);    /* Posición 18: ancho */
        fread(&ancho, 1, 4, fichero);
        printf("Ancho: %d\n", ancho);
        fread(&alto, 1, 4, fichero);     /* Siguiente dato: alto */
        printf("Alto: %d\n", alto);
        fseek(fichero, 4, SEEK_CUR);     /* 4 bytes después: compresión */
        fread(&compresion, 1, 4, fichero);
        fclose(fichero);
        switch (compresion) {
          case 0: puts("Sin compresión"); break;
          case 1: puts("Compresión RLE 8 bits"); break;
          case 2: puts("Compresión RLE 4 bits"); break;
          }
    } else
      printf("No parece un fichero BMP\n");  /* Si la marca no es BM */
  }
}
```

Ejercicios propuestos:

- Mejorar la última versión de la agenda anterior (la que usa fwrite, fread y sizeof) para que no lea todas las fichas a la vez, sino que lea una única ficha del disco cada vez que lo necesite, saltando a la posición en que se encuentra dicha ficha con "fseek".
- Hacer un programa que muestre información sobre una imagen en formato GIF (se deberá localizar en Internet los detalles sobre dicho formato): versión, ancho de la imagen (en píxeles), alto de la imagen y cantidad de colores.
- Hacer un programa que muestre información sobre una imagen en formato PCX: ancho de la imagen (en píxeles), alto de la imagen y cantidad de colores.

- Mejorar la base de datos de ficheros (ejemplo 53) para que los datos se guarden en disco al terminar la sesión de uso, y se lean de disco cuando comienza una nueva sesión.
- Mejorar la base de datos de ficheros (ejemplo 53) para que cada dato introducido se guarde inmediatamente en disco, sin esperar a que termine la sesión de uso. En vez de emplear un "array de structs", debe existir un solo "struct" en memoria cada vez, y para las búsquedas se recorra todo el contenido del fichero.
- Mejorar el ejercicio anterior (ejemplo 53 ampliado con ficheros, que se manejan ficha a ficha) para que se pueda modificar un cierto dato a partir de su número (por ejemplo, el dato número 3). En esa modificación, se deberá permitir al usuario pulsar Intro sin teclear nada, para indicar que no desea modificar un cierto dato, en vez de reemplazarlo por una cadena vacía.

6.11. Ficheros especiales 1: la impresora

Mandar algo a impresora desde C no es difícil (al menos en principio): en muchos sistemas operativos, la impresora es un dispositivo al que se puede acceder a través como si se tratara de un fichero.

Por ejemplo, en MsDos, se puede mostrar un fichero de texto en pantalla usando

```
TYPE DATOS.TXT
```

y lo mandaríamos a impresora si redirigimos la salida hacia el dispositivo llamado PRN:

```
TYPE DATOS.TXT > PRN:
```

De igual manera, desde C podríamos crear un programa que mandara información al fichero ficticio PRN: para escribir en impresora, así:

```
/*---------------------------*/
/*  Ejemplo en C nº 61:      */
/*  c061.c                   */
/*                           */
/*  Escritura en impresora   */
/*  (con MsDos)              */
/*                           */
/*  Curso de C,              */
/*    Nacho Cabanes          */
/*---------------------------*/

#include <stdio.h>

main()
{
    FILE* impresora;

    impresora = fopen("prn:", "wt");
    fputs("Esto va a la impresora\n", impresora;);
    fclose(impresora);
}
```

(este mismo ejemplo debería funcionar desde muchas versiones de Windows, con bastante independencia de la impresora que tengamos instalada).

En Linux la idea sería la misma, pero el nombre de dispositivo sería "/dev/lp". Como inconveniente, normalmente sólo puede escribir en este dispositivo el administrador y los usuarios que pertenezcan a su grupo. Si pertenecemos a ese grupo, haríamos:

```
impresora = fopen("/dev/lp", "wt");
```

6.12. Ficheros especiales 2: salida de errores

Hemos comentado que en muchos sistemas operativos se puede usar el símbolo ">" para redirigir hacia "otro sitio" (la impresora o un fichero de texto, por ejemplo) la información que iba destinada originalmente a la pantalla. Esto funciona, entre otros, en Windows, MsDos y toda la familia de sistemas operativos Unix (incluido Linux).

Pero en el caso de Linux (y los Unix en general) podemos redirigir además los mensajes de error hacia otro sitio distinto del resto de mensajes (que iban destinados a pantalla). Esto se consigue con el símbolo "2>" :

```
calculaResultados  > valores.txt  2> errores.txt
```

Esta orden pone en marcha un programa llamado "calculaResultados", guarda en el fichero "valores.txt" los mensajes que normalmente aparecerían en pantalla, y guarda en el fichero "errores.txt" los mensajes de error.

Esta política de separar los mensajes de información y los mensajes de error es fácil de llevar a nuestros programas. Basta con que los mensajes de error no los mandemos a pantalla con órdenes como "printf", sino que los mandemos a un fichero especial llamado "stderr" (salida estándar de errores).

Por ejemplo, a la hora de intentar abrir un fichero podríamos hacer:

```
fichero = fopen("ejemplo.txt", "rt");
if (fichero == NULL)
  fprintf(stderr, "Fichero no encontrado!\n");
else
  printf("Accediendo al fichero...\n");
```

Si el usuario de nuestro programa no usa "2>", los mensajes de error le aparecerían en pantalla junto con cualquier otro mensaje, pero si se trata de un usuario avanzado, le estamos dando la posibilidad de analizar los errores cómodamente.

6.13. Un ejemplo de lectura y escritura: TAG de un MP3

Los ficheros de sonido en formato MP3 pueden contener información sobre el autor, el título, etc. Si la contienen, se encontraría a 128 bytes del final del fichero. Los primeros 3 bytes de esos 128 deberían ser las letras TAG. A continuación, tendríamos otros 30 bytes que serían el

título de la canción, y otros 30 bytes que serían el nombre del autor. Con esto ya podríamos crear un programa que lea esa información de un fichero MP3 (si la contiene) e incluso que la modifique.

Estos textos (título, autor y otros) deberían estar rellenos con caracteres nulos al final, pero es algo de lo que no tenemos la certeza, porque algunas aplicaciones lo rellenan con espacios (es el caso de alguna versión de WinAmp). Por eso, leeremos los datos con "fread" y añadiremos un carácter nulo al final de cada uno.

Además, haremos que el programa nos muestre la información de varios ficheros: nos pedirá un nombre, y luego otro, y así sucesivamente hasta que pulsemos Intro sin teclear nada más.

```
/*---------------------------*/
/*  Ejemplo en C nº 62:      */
/*  c062.c                   */
/*                           */
/*  Lectura y escritura en   */
/*  un fichero               */
/*                           */
/*  Curso de C,              */
/*    Nacho Cabanes          */
/*---------------------------*/

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

main() {
    FILE* fich;
    char temp[31];
    int i;

    do {
    /* Pido el nombre del fichero */
    printf("\nEscribe el nombre del fichero MP3 a comprobar: ");
    gets(temp);
    /* Si no teclea nada, terminaré */
    if (strcmp(temp,"")==0)
        puts("\nAplicacion finalizada.");
    /* Si existe nombre, intento abrir */
    else if ( (fich=fopen(temp,"r+b"))!=NULL ){
        /* Si he podido abrir, muestro el nombre */
        printf("Nombre del fichero: %s\n",temp);
        /* Miro el tamaño del fichero */
        fseek(fich,0,SEEK_END);
        printf("Tamaño: %d\n",ftell(fich));
        /* A 128 bytes está la marca "TAG" */
        fseek(fich,-128,SEEK_END);
        fread(temp,3,1,fich);
        /* Son 3 letras, añado caracter nulo al final */
        temp[3]='\0';
        if (strcmp(temp,"TAG")!=0)
            puts("No se encontró información válida.");
        else {
            /* Si existe la marca, leo los datos */
            /* Primero, 30 letras de titulo */
            fread(temp,30,1,fich);
            temp[strlen(temp)]='\0';
            printf("Titulo: %s\n",temp);
            /* Luego 30 letras de autor */
```

```
            fread(temp,30,1,fich);
            temp[strlen(temp)]='\0';
            printf("Artista: %s\n",temp);
            /* Ahora vamos a modificar el titulo */
            printf("\nIntroduce el nuevo titulo: ");
            gets(temp);
            /* Lo rellenamos con ceros, para seguir el estándar */
            for (i=strlen(temp); i<=29; i++)
                temp[i]='\0';
            /* Y lo guardamos en su posición */
            fseek(fich,-125,SEEK_END);
            fwrite(&temp, 30, 1, fich);
            printf("Titulo actualizado.\n");
            fclose(fich);
            } /* Fin del "else" de MP3 con informacion */
    } /* Fin del "else" de fichero existente */
    else puts("No se pudo abrir el fichero\n");
    } /* Fin del "do..while" que repite hasta que no se escriba nombre */
    while (strcmp(temp,"")!=0);
}
```