Ficheros en C

Libreta:	Armando Rojas - Cursos Platzi		
Creado:	09/06/2018 11:18	**Actualizado:**	09/06/2018 11:19
Autor:	armando rojas		
URLOrigen:	http://www.it.uc3m.es/abel/as/MMC/L2/FilesDef_es.html		

1. Introducción

C ofrece un conjunto de funciones para realizar operaciones de entrada y salida (E/S) con las cuales puedes leer y escribir cualquier tipo de fichero. Antes de ver estas funciones, hay que entender algunos conceptos básicos.

1.1. Qué es un fichero

En C, un *fichero* se puede referir a un fichero en disco, a un terminal, a una impresora, etc. Dicho de otro modo, un fichero representa un dispositivo concreto con el que puedes intercambiar información. Antes de poder trabajar con un fichero, tienes que abrir ese fichero. Tras terminar el intercambio de información, debes cerrar el fichero que abriste.

1.2. Qué es un flujo o *stream*

El conjunto datos que transfieres desde un programa a un fichero, o al revés, se le conoce como *flujo* (*stream* en inglés), y consiste en una serie de bytes (o caracteres). A diferencia de un fichero, que se refiere a un dispositivo concreto de E/S, un flujo es independiente del dispositivo. Es decir, todos los flujos tienen el mismo comportamiento independientemente del dispositivo al que esté asociado. De esta manera puedes realizar operaciones E/S con tan sólo asociar un flujo a un fichero.

Hay dos tipos de flujos. Uno es el *flujo de texto*, consistente en líneas de texto. Una línea es una secuencia de caracteres terminada en el carácter de línea nueva (`'\n'` ó `'\r\n'`). El otro formato es el del *flujo binario*, que consiste en una secuencia de bytes que representan datos internos como números, estructuras o arrays. Se utiliza principalmente para datos que no son de tipo texto, donde no importa la apariencia de esos datos en el fichero (da igual que no se vean "en bonito", como podría verse en un fichero de texto).

1.3. E/S mediante buffers

Una porción de memoria que se usa temporalmente para almacenar datos antes de ser enviados a su destino se llama *buffer*. Con ayuda de los buffers, el sistema operativo puede mejorar su eficiencia reduciendo el número de accesos a efectuar en un dispositivo de E/S (es decir, en un fichero).

El acceso a disco o a otros dispositivos de E/S suele ser más lento que a memoria de acceso directo; por eso, si varias operaciones E/S se realizan en un buffer en vez de en el fichero en sí, el funcionamiento de un programa es más eficiente. Por ejemplo, si se envían varias operaciones de escritura a un buffer, las modificaciones de esos datos escritos se quedan en memoria hasta que es hora de guardarlas, que es cuando esos datos se llevan al dispositivo real (a este proceso se le conoce como *flushing*).

Por defecto, todas las operaciones de los flujos E/S en C son con buffer.

1.4. Comportamiento del modelo E/S

Ya que la unidad más pequeña que se puede representar en C es un caracter, se puede acceder a un fichero desde cualquiera de sus caracteres (o bytes, que es lo mismo). Se puede leer o escribir cualquier número de caracteres desde un punto móvil, conocido como el*indicador de posición de fichero*. Los caracteres se leen o escriben en secuencia desde ese punto, y el indicador se va moviendo de acuerdo a eso. El indicador se coloca al principio del fichero cuando éste se abre, pero luego se puede mover a cualquier punto explícitamente (no sólo porque se haya leído o escrito algo).

2. Funciones básicas

Vamos a centrarnos en cómo abrir y cerrar ficheros y en cómo interpretar los mensajes de error que pueden dar estas dos operaciones.

2.1. Punteros a FILE

La estructura FILE es la estructura que controla los ficheros y se encuentra en la cabecera stdio.h. Los flujos utilizan estos punteros a ficheros para realizar las operaciones E/S. La siguiente línea de código declara una variable de tipo puntero a fichero:

```
FILE *ptr_fich;
```

En la estructura FILE es donde se encuentra el atributo que representa el indicador de posición de fichero.

2.2. Abriendo un fichero

La función fopen abre un fichero y lo asocia a un flujo. Necesitas especificar como argumentos el nombre de ese fichero (o la ruta y el nombre) y el modo de apertura.

```
#include <stdio.h>
FILE *fopen(const char *filename, const char *mode);
```

Aquí filename es un puntero a char que referencia un string con el nombre del fichero. mode apunta a otro string que especifica la manera en la que se va a abrir el fichero. La función fopen devuelve un puntero de tipo FILE. Si ha ocurrido algún error al abrirse, devuelve NULL.

El parámetro mode es una combinación de los caracteres r (*read*, lectura), w (*write*, escritura), b (binario), a (*append*, añadir), y +(actualizar). Si usas el caracter a y el fichero existe, el contenido de ese fichero se mantiene y los datos nuevos se añaden justo al final, tras el último dato que ya estuviera escrito (el indicador de fichero pues no está situado al inicio del fichero, sino que en este caso está al final). Si el fichero no existe, lo crea. Con w es diferente; este modo siempre borra los datos del fichero si existe (si no, crea uno nuevo). Si al modo le añades el +, permite que el fichero se abra para escritura o lectura y puedes modificar cualquier dato que hubiera en él. Si usas r, el fichero debe existir; si no, fopen dará error y te devolverá NULL.

La siguiente lista muestra los posibles modos de abrir un fichero:

- "r" abre un fichero de texto existente para lectura.
- "w" crea un fichero de texto para escritura.
- "a" abre un fichero de texto existente para añadir datos.
- "r+" abre un fichero de texto existente para lectura o escritura.
- "w+" crea un fichero de texto para lectura o escritura.
- "a+" abre o crea un fichero de texto para añadirle datos.
- "rb" abre un fichero binario existente para lectura.

- "wb" crea un fichero binario para escritura.
- "ab" abre un fichero binario existente para añadir datos.
- "r+b" abre un fichero binario existente para lectura o escritura.
- "w+b" crea un fichero binario para lectura o escritura.
- "a+b" abre o crea un fichero binario para añadirle datos.

A veces podrás ver que el modo está escrito como "rb+" en vez de "r+b", por ejemplo, pero es equivalente.

2.3. Cerrando un fichero

Después de abrir un fichero y leerlo o escribir en él, hay que desenlazarlo del flujo de datos al que fue asociado. Esto se hace con la función fclose, cuya sintaxis es la siguiente:

```
#include <stdio.h>
int fclose(FILE *stream);
```

Si fclose cierra bien el fichero, devuelve 0. Si no, la función devuelve EOF. Normalmente sólo falla si se ha borrado el fichero antes de intentar cerrarlo. Acuérdate siempre de cerrar el fichero. Si no lo haces, se pueden perder los datos almacenados en él. Además, si no lo cierras, otros programas pueden tener problemas si lo quieren abrir más tarde.

El siguiente programa abre y cierra un fichero de texto, comprobando los posibles errores que puedan surgir:

```
 1  #include <stdio.h>
 2
 3  enum {EXITO, FALLO};
 4  int main(void)
 5  {
 6  FILE *ptr_fichero;
 7  char nombre_fichero[] = "resumen.txt";
 8  int resultado = EXITO;
 9
10  if ( (ptr_fichero = fopen(nombre_fichero, "r") ) == NULL)
11  {
12  printf("No se ha podido abrir el fichero %s.\n",
13  nombre_fichero);
14  resultado = FALLO;
15  }
16  else
17  {
18  printf("Abierto fichero; listos para cerrarlo.\n");
19  if (fclose(ptr_fichero)!=0)
20  {
21  printf("No se ha podido cerrar el fichero %s.\n",
22  nombre_fichero);
23  resultado = FALLO;
```

```
24      }
25      }
        return resultado;
        }
```

2.4. Abriendo un fichero con un descriptor

Para poder abrir un fichero dado su descriptor de fichero (*file descriptor (fd)* en inglés) se ha de usar la función `fdopen`. Esta función se comporta como la función `fopen`, pero en vez de abrir un fichero dado su nombre, usa el identificador de fichero para abrirlo:

```
#include <stdio.h>
int fdopen(int fildes, const char *mode);
```

Una forma de conseguir un descriptor para un fichero es con la función `mkstemp`:

```
#include <stdio.h>
int mkstemp(char *template);
```

Esta función genera un nombre de fichero temporal único a partir del string `template`. Los últimos seis caracteres de `template` deben ser "`XXXXXX`", que son reemplazados luego por `mkstemp` con una cadena que hace que el nombre no esté repetido. Ya que será modificada, `template` no debe ser definida como constante. La función `mkstemp` devuelve el descriptor de fichero del fichero temporal creado o `-1` en caso de error.

3. Lectura y escritura de ficheros

En C puedes realizar las operaciones de lectura y escritura de varias maneras:

- Leer y escribir carácter a carácter (o lo que es lo mismo, byte a byte), con funciones como `fgetc` y `fputc`.
- Leer y escribir línea a línea, con funciones como `fgets` y `fputs`.
- Leer y escribir un bloque de caracteres (o bytes) cada vez, con `fread` y `fwrite`.

Para este curso nos vamos a centrar en la última, la lectura por bloques, que nos será útil tanto para ficheros de texto como para binarios.

3.1. Lectura/Escritura por bloques

En C contamos con las funciones `fread` y `fwrite` para realizar operaciones E/S por bloques.

La sintaxis de `fread` es la siguiente:

```
#include <stdio.h>
size_t fread(void *ptr, size_t size, size_t n, FILE *stream);
```

Aquí `ptr` es un array donde se van a almacenar los datos que se van a leer. `size` indica el tamaño de cada elemento del array. `n`especifica el número de elementos que se van a leer. `stream` es un puntero a un fichero que ha tenido que haber sido abierto previamente. `size_t` es un tipo que está definido también en la cabecera `stdio.h`. La función devuelve el número de elementos que realmente se han leído (que puede ser menor que `n`); si todo ha ido bien o aún no se ha llegado al final del fichero, ese número devuelto debería ser mayor que 0 e igual o menor (en caso de que no sepamos la longitud de lo que vamos a leer) que el tercer argumento, `n`.

Si obtenemos un error, habrá que distinguir si se produjo porque llegamos al final del fichero o por alguna otra causa; para ello contamos con las funciones `feof` y `ferror`, que veremos a continuación.

La sintaxis de `fwrite` es la siguiente:

```
#include <stdio.h>
size_t fwrite(const void *ptr, size_t size, size_t n, FILE *stream);
```

`ptr` hace referencia a un array con los datos que se van a escribir al fichero abierto apuntado por `stream`. `size` indica el tamaño de cada elemento del array. `n` especifica el número de elementos que se van a escribir. `stream` es un puntero a un fichero que ha tenido que haber sido abierto previamente. La función devuelve el número de elementos que se han escrito; luego si no ha habido ningún error, ese número devuelto debería ser igual que el tercer argumento, `n`. El valor devuelto puede ser menos que este `n` si ha habido algún error.

Como se ha comentado anteriormente, tenemos la función `feof`, que se usa para saber cuándo hemos llegado al final del fichero:

```
#include <stdio.h>
int feof(FILE *stream);
```

Esta función devuelve `0` si aún no se ha llegado al final del fichero. Si se ha llegado, devuelve un entero distinto de cero.

Finalmente, contamos con la función `ferror` para saber si ha ocurrido algún error en la lectura o escritura, que devolverá `0` si no ha ocurrido ninguno y un valor distinto de cero en caso contrario:

```
#include <stdio.h>
int ferror(FILE *stream);
```

El siguiente programa va leyendo un fichero de bloque en bloque y escribe dichos bloques a un fichero de salida.

```
1  #include <stdio.h>
```

```
2
3   enum {EXITO, FALLO, LONGITUD_MAXIMA = 80};
4
5   int error_msg(char *nombre_fichero)
6   {
7   printf("Falló algo con el fichero %s.\n", nombre_fichero);
8   return FALLO;
9   }
10
11  int main(void)
12  {
13  FILE *ptr_fich1, *fich_ptr2;
14  char nombre_fich1[] = "copia_resumen.txt";
15  char nombre_fich2[] = "resumen.txt";
16  int resultado = EXITO;
17
18  if ( (ptr_fich1 = fopen(nombre_fich1, "w")) == NULL )
19  {
20  resultado = error_msg(nombre_fich1);
21  }
22  else
23  {
24  if ( (ptr_fich2 = fopen(nombre_fich2, "r")) == NULL )
25  {
26  resultado = error_msg(nombre_fich2);
27  }
28  else
29  {
30  int num;
31  char buffer[LONGITUD_MAXIMA + 1];
32  while(!feof(ptr_fich2))
33  {
34  num = fread(buffer,sizeof(char), LONGITUD_MAXIMA, ptr_fich2);
35  buffer[num*sizeof(char)] = '\0';
36  printf("%s.",buffer);
37  if (fwrite(buffer, sizeof(char), num, ptr_fich1) != num)
38  {
39  resultado = error_msg(nombre_fich1);
40  break;
41  }
42  } //end of while
43  printf("\n");
44  if (fclose(ptr_fich2)!=0)
45  {
46  resultado = error_msg(nombre_fich2);
47  }
48  }
49  if (fclose(ptr_fich2)!=0)
50  {
51  resultado = error_msg(nombre_fich1);
52  }
53  }
54  return resultado;
55  }
```

El buffer declarado en la línea 31 tiene `LONGITUD_MAXIMA + 1` elementos, aunque sólo leemos `LONGITUD_MAXIMA` elementos por bloque. Esto es así porque necesitamos añadir en la línea 35 el carácter de terminación, para que `buffer` sea considerado como un string y la función `printf` pueda imprimirlo correctamente, sin lugar a errores.

4. Acceso aleatorio a ficheros

En la sección anterior hemos visto cómo leer y escribir datos de manera secuencial. En algunos casos, sin embargo, se necesita acceder a una parte específica en mitad de un fichero. Para eso contamos con las funciones, `fseek`, `ftell` y `rewind`, que permiten realizar accesos aleatorios.

La función `fseek` permite desplazar el indicador de posición de fichero al sitio desde donde el cual quieres acceder al fichero. La sintaxis es:

```
#include <stdio.h>
int fseek(FILE *stream, long offset, int whence);
```

`stream` es el puntero asociado con el fichero abierto. `offset` indica el número de bytes que se va a desplazar el indicador desde una posición especificada por `whence`, que puede tener uno de los siguientes valores: `SEEK_SET`, `SEEK_CUR` y `SEEK_END`. Si todo va bien, la función devuelve `0`; si no, devuelve un valor distinto de cero.

Si se utiliza `SEEK_SET`, el desplazamiento se realizará desde el principio del fichero, y el tamaño de `offset` deberá ser mayor o igual que cero. Si se usa `SEEK_END`, el desplazamiento se realizará desde el final del fichero, y el valor de `offset` tendrá que ser negativo. Si se usa `SEEK_CUR`, el desplazamiento se calcula desde la posición actual del indicador de posición de fichero.

Puedes obtener el valor actual del indicador de posición de fichero llamando a `ftell`:

```
#include <stdio.h>
int ftell(FILE *stream);
```

El valor que devuelve `ftell` representa el número de bytes desde el principio del fichero hasta donde nos encontramos en ese momento (sitio indicado por el indicador de posición). Si la función falla, devuelve `-1L` (es decir, el valor largo (`long`) de menos 1).

A veces te interesará reiniciar el indicador de posición de fichero y llevarlo al inicio del mismo. Para esto cuentas con la función `rewind`:

```
#include <stdio.h>
void rewind(FILE *stream);
```

De esta manera, la siguiente línea de código:

```
rewind(ptr_fich);
```

es equivalente a esta otra:

```
fseek(ptr_fich,0L,SEEK_SET);
```

5. Manipulación directa de ficheros

Existen otra serie de funciones adicionales para manejar ficheros que puede que te sean útiles.

La función `remove` que, dado el nombre de un fichero, lo borra del sistema de ficheros:

```
int remove(char *filename);
```

Esta función devuelve `0` si el fichero pudo eliminarse correctamente; si no, devuelve otro número.

La función `rename`, que renombra un fichero, teniendo además la posibilidad de poder moverlo entre directorios si en el argumento `newname` ponemos una ruta en vez de sólo un nombre de fichero:

```
int rename(const char *oldname, const char *newname);
```

Esta función devuelve `0` si el fichero pudo renombrarse correctamente; si no, devuelve otro número. Si falla por cualquier razón, el fichero original no se ve afectado.