Manejo de Archivos en C.

Guía para los estudiantes de programación básica

Definición

En los programas que hasta el momento se han desarrollado en el curso, los datos que hemos trabajado hasta el momento han residido en la memoria principal. Sin embargo, las grandes cantidades de datos se almacenan normalmente en un dispositivo de memoria secundaria (ya sean magnéticos como los discos duros o HDD, discos flexibles o disquetes o FDD, discos ópticos en sus diversos formatos o CD, DVD o BlueRay, o discos de estado sólido como SDD o memorias flash como SD o USB). Estas colecciones de datos se conocen como archivos (antes se les denominaba ficheros).

Un archivo es un conjunto de datos estructurados y los hay de dos tipos: archivos de texto y archivos binarios. Un **archivo de texto** es una secuencia de caracteres organizadas en líneas terminadas por un carácter de nueva línea. En estos archivos se pueden almacenar letras de canciones, código fuente de programas, base de datos simples, etc. Los archivos de texto se caracterizan por ser planos, es decir, todas las letras tienen el mismo formato y no hay palabras subrayadas, en negrita, o letras de distinto tamaño o ancho, simplemente son caracteres.

Un **archivo binario** es una secuencia de bytes que tienen una correspondencia uno a uno con un dispositivo externo. Así que no tendrá lugar ninguna traducción de caracteres. Además, el número de bytes escritos (leídos) será el mismo que los encontrados en el dispositivo externo. Ejemplos de estos archivos son sonidos, videos, imágenes, texto con formatos, archivos ejecutables (aplicaciones), etc.

El estándar de C contiene varias funciones para la edición de archivos, éstas están definidas en la librería *stdio.h* y por lo general empiezan con la letra f, haciendo referencia a file. Adicionalmente se agrega un tipo **FILE**, el cual se usará como *apuntador a la información del archivo*. La secuencia que usaremos para realizar operaciones será la siguiente:

- Crear un apuntador del tipo **FILE ***
- Abrir el archivo utilizando la función **fopen** y asignándole el resultado de la llamada a nuestro apuntador.
- Hacer las diversas operaciones (lectura, escritura, etc).
- Cerrar el archivo utilizando la función **fclose**.

Hay diversas funciones que operan sobre los archivos, que están definidas en esta librería como:

Nombre	Función
fopen()	Abre un archivo.
fclose()	Cierra un archivo.
fgets()	Lee una cadena de un archivo.
fputs()	Escribe una cadena en un archivo
fseek()	Busca un byte específico de un archivo.
fprintf()	Escribe una salida con formato en el archivo.
fscanf()	Lee una entrada con formato desde el archivo.
feof()	Devuelve cierto si se llega al final del archivo.
ferror()	Devuelve cierto si se produce un error.
rewind()	Coloca el localizador de posición del archivo al principio del mismo.
remove()	Borra un archivo.
fflush()	Vacía un archivo.

Apertura

El puntero a un archivo.

El puntero a un archivo es el hilo común que unifica el sistema de E/S con la memoria (buffer). Un puntero a un archivo es un puntero a una información que define varias cosas sobre el archivo, incluyendo el nombre, el estado y la posición actual del archivo. En esencia identifica un archivo específico y utiliza la secuencia asociada para dirigir el funcionamiento de las funciones de E/S con el buffer. Un puntero a un archivo es una variable de tipo puntero al tipo **FILE** que se define en la librería STDIO.H. Un programa necesita utilizar punteros a archivos para leer o escribir en los mismos. Para obtener una variable de este tipo se utiliza una secuencia como esta: **FILE *archivo;**

La función fopen()

Abre una secuencia para que pueda ser utilizada y la asocia a un archivo. Su prototipo es:

```
FILE* fopen (const char *nombreArchivo, const char *modo);
```

Donde `nombreArchivo` es un puntero a una cadena de caracteres que representan un nombre valido del archivo y puede incluir una especificación del directorio. La cadena a la que apunta modo determina como se abre el archivo. La siguiente tabla muestra los valores permitidos para modo.

Modo	Significado
r	Abre un archivo de texto para lectura. El archivo debe existir
w	Crea un archivo de texto para escritura. Se crea si no existe o se sobreescribe si existe.
a	Abre un archivo de texto para añadir al final del contenido. si no existe se crea
r+	Abre un archivo de texto para lectura / escritura. Debe existir.
w+	Crea un archivo de texto para lectura / escritura. Se crea si no existe o se sobreescribe si existe.
a+	Añade o crea un archivo de texto para lectura / escritura.
r+b	Abre un archivo binario para lectura / escritura.
w+b	Crea un archivo binario para lectura / escritura.
a+b	Añade o crea un archivo binario para lectura / escritura.

La función **fopen()** devuelve un puntero a archivo. Un programa nunca debe alterar el valor de ese puntero. Si se produce un error cuando se está intentando abrir un archivo, **fopen()** devuelve un puntero nulo. Se puede abrir un archivo bien en modo texto o binario. En la mayoría de las implementaciones, en modo texto, la secuencias de retorno de carro / salto de línea se convierten a caracteres de salto de línea en lectura. En la escritura, ocurre lo contrario: los caracteres de salto de línea se convierten en salto de línea. Estas conversiones no ocurren en archivos binarios.

La macro NULL (que es el puntero nulo) está definida en STDIO.H. Cuando se encuentra es porque la función fopne() detecta cualquier error al abrir un archivo, por ejemplo: disco lleno o protegido contra escritura antes de comenzar a escribir en él.

Si se usa **fopen()** para abrir un archivo para escritura, entonces cualquier archivo existente con el mismo nombre se borrará y se crea uno nuevo. Si no existe un archivo con el mismo nombre, entonces se creará. Si se quiere añadir al final del archivo entonces debe usar el modo **a**. Si se usa **a** y no existe el archivo, se devolverá un error. La apertura de un archivo para las operaciones de lectura requiere que exista el archivo. Si no existe, **fopen()** devolverá un error. Finalmente, si se abre un archivo para las operaciones de leer / escribir, la computadora no lo borrará si existe; sin embargo, si no existe, la computadora lo creará.

Cierre

La función **fclose()** cierra una secuencia que fue abierta mediante una llamada a **fopen()**. Escribe toda la información que todavía se encuentre en el buffer en el disco y realiza un cierre formal del archivo a nivel del sistema operativo. Un error en el cierre de una secuencia puede generar todo tipo de problemas, incluyendo la pérdida de datos, destrucción de archivos y posibles errores intermitentes en el programa. El prototipo de esta función es:

```
int fclose (FILE *archivo);
```

Donde "`archivo`" es el puntero al archivo devuelto por la llamada a **fopen()**. Si se devuelve un valor cero significa que la operación de cierre ha tenido éxito. Generalmente, esta función solo falla cuando un disco se ha retirado antes de tiempo o cuando no queda espacio libre en el mismo.

Posicionamiento

En algunos casos, 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 en cualquier parte del archivo.

función rewind()

Sitúa el cursor de lectura/escritura al principio del archivo indicado por su argumento. Su prototipo es:

```
void rewind (FILE *archivo);
```

Donde "`archivo`" es un puntero a un archivo válido.

La función fseek ()

Permite desplazar el indicador de posición del archivo al sitio desde donde el cual quieres acceder al contenido específico de éste. La sintaxis es:

```
int fseek(FILE *stream, long offset, int whence);
```

Donde `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.

La función ftell()

Se obtiene el valor actual del indicador de posición del archivo. Su prototipo es:

```
int ftell(FILE *stream);
```

El valor que devuelve 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).

Un ejemplo de las funciones de posicionamiento sería:

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

int main()
{
  char nombre[11] = "datos4.dat", mensaje[81]="";
  FILE *fichero;
  long int comienzo, final;

  fichero = fopen( nombre, "r" );
  printf( "Fichero: %s -> ", nombre );
  if( fichero )
    printf( "existe (ABIERTO)\n" );
  else
  {
    printf( "Error (NO ABIERTO)\n" );
    return 1;
  }

  if( (comienzo = ftell( fichero )) < 0 )
        printf( "ERROR: ftell no ha funcionado\n" );
  else
        printf( "Posición del fichero: %d\n\n", comienzo );

  fseek( fichero, 0L, SEEK_END );
  final = ftell( fichero );
```

```
  fseek( fichero, 0L, SEEK_SET );
  fgets( mensaje, 80, fichero );
  printf( "Tamaño del fichero \"%s\": %d bytes\n", nombre, final-comienzo+1 );
  printf( "Mensaje del fichero:\n%s\n", mensaje );
  printf( "\nTamaño del mensaje (usando strlen): %d\n", strlen(mensaje) );

  if( !fclose(fichero) )
    printf( "Fichero cerrado\n" );
  else
  {
    printf( "Error: fichero NO CERRADO\n" );
    return 1;
  }
  return 0;
}
```

Lectura

Un archivo de texto generalmente debe verse como un string (una cadena de caracteres) que está guardado en el disco duro, así como un binario debe verse como un arreglo de bytes. Para trabajar con los archivos existen diferentes formas y diferentes funciones. Las funciones que podríamos usar para leer un archivo son:

- char fgetc(FILE *archivo)
- char *fgets(char *buffer, int tamano, FILE *archivo)
- size_t fread(void *puntero, size_t tamano, size_t cantidad, FILE *archivo);
- int fscanf(FILE *fichero, const char *formato, argumento, ...);

Las primeras dos de estas funciones son muy parecidas entre sí. Pero la tercera, por el número y el tipo de parámetros, nos podemos dar cuenta de que es muy diferente, por eso la trataremos aparte junto al **fwrite** que es su contraparte para escritura.

La función fgetc()

Esta función lee un caracter a la vez del archivo que está siendo señalado con el puntero ***archivo**. En caso de que la lectura sea exitosa devuelve el caracter leído y en caso de que no lo sea o de encontrar el final del archivo devuelve **EOF**. El prototipo correspondiente de **fgetc** es:

```
char fgetc(FILE *archivo);
```

Esta función se usa generalmente para recorrer archivos de texto. A manera de ejemplo vamos a suponer que tenemos un archivo de texto llamado "prueba.txt" en el mismo directorio en que se encuentra el código fuente de nuestro programa. Un pequeño programa que lea ese archivo será:

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

int main()
{
       FILE *archivo;
```

```
        int caracter;

        archivo = fopen("prueba.txt","r");

        if (archivo == NULL){
              printf("\nError de apertura del archivo. \n\n");
         }
         else{
              printf("\nEl contenido del archivo de prueba es \n\n");
              while((caracter = fgetc(archivo)) != EOF){
                  printf("%c",caracter);
              }
         }
         fclose(archivo);
        return 0;
}
```

La función fgets()

Esta función está diseñada para leer cadenas de caracteres. Leerá hasta n-1 caracteres o hasta que lea un cambio de línea '\n' o un final de archivo EOF. En este último caso, el carácter de cambio de línea '\n' también es leído. El prototipo correspondiente de **fgets** es:

```
char *fgets(char *buffer, int tamaño, FILE *archivo);
```

El parámetro `buffer` lo hemos llamado así porque es un puntero a un espacio de memoria del tipo char (podríamos usar un arreglo de char). El segundo parámetro `tamaño` es el límite en cantidad de caracteres a leer para la funcion **fgets**. Y por último el puntero del `archivo` de refiere a que archivo debe leer.

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

int main()
{
        FILE *archivo;
        char caracteres[100];
        archivo = fopen("prueba.txt","r");
        if (archivo == NULL)
              exit(1);
        else{
            printf("\nEl contenido del archivo de prueba es \n\n");
            while (feof(archivo) == 0)
            {
                fgets(caracteres,100,archivo);
                printf("%s",caracteres);
            }
             system("PAUSE");
         }
         fclose(archivo);
        return 0;
}
```

Este es el mismo ejemplo de antes con la diferencia de que este hace uso de **fgets** en lugar de **fgetc**. La función **fgets** se comporta de la siguiente manera, leerá del archivo apuntado por archivo los caracteres que encuentre y a ponerlos en buffer hasta que lea un caracter menos que la cantidad de caracteres especificada en tamaño o hasta que encuentre el final de una línea (\n) o hasta que encuentre el final del archivo (**EOF**). El beneficio de esta función es que se puede obtener una línea completa a la vez.

La function fread()

El prototipo es:

```
size_t fread (void *buffer, size_t size, size_t count, FILE *stream);
```

Esta función lee un bloque de un "`stream`" de datos. Efectúa la lectura de un arreglo de elementos "`count`", cada uno de los cuales tiene un tamaño definido por "`size`". Luego los guarda en el bloque de memoria especificado por "`buffer`". El indicador de posición de la cadena de caracteres avanza hasta leer la totalidad de bytes. Si esto es exitoso la cantidad de bytes leídos es (`count`).

Parámetros:

- `buffer`: Puntero a un bloque de memoria con un tamaño mínimo de (size*count) bytes.
- `size`: Tamaño en bytes de cada elemento (de los que voy a leer).
- `count`: Número de elementos, los cuales tienen un tamaño "`size`".
- `stream`: Puntero a objetos FILE, que especifica la cadena de entrada.

La función fscanf()

Esta función funciona igual que scanf en cuanto a parámetros, pero la entrada se toma de un fichero en lugar del teclado.El prototipo correspondiente de **fscanf** es:

```
int fscanf(FILE *fichero, const char *formato, argumento, ...);
```

Un ejemplo de su uso, sería

```
#include <stdio.h>

int main ( int argc, char **argv )
{
       FILE *fp;
       char buffer[100];
       fp = fopen ("fichero.txt", "r");
       fscanf(fp, "%s" ,buffer);
       printf("%s", buffer);
       fclose (fp);
       return 0;
}
```

Escritura

Así como podemos leer datos desde un fichero, también se pueden crear y escribir ficheros con la información que deseamos almacenar, Para trabajar con los archivos existen diferentes formas y diferentes funciones. Las funciones que podríamos usar para escribir dentro de un archivo son:

- int fputc(int caracter, FILE *archivo)
- int fputs(const char *buffer, FILE *archivo)
- size_t fwrite(void *puntero, size_t tamano, size_t cantidad, FILE *archivo);
- int fprintf(FILE *archivo, const char *formato, argumento, ...);

La función fputc()

Esta función escribe un carácter a la vez del archivo que está siendo señalado con el puntero ***archivo**. El valor de retorno es el carácter escrito, si la operación fue completada con éxito, en caso contrario será **EOF**.

El prototipo correspondiente de **fputc** es:

```
int fputc(int carácter, FILE *archivo);
```

Mostramos un ejemplo del uso de **fputc** en un "fichero.txt", se escribirá dentro del fichero hasta que presionemos la tecla **enter**.

```
#include <stdio.h>

int main ( int argc, char **argv ){
	FILE *fp;
	char caracter;
	fp = fopen ( "fichero.txt", "a+t" ); // para escritura al final y tipo texto
	printf("\nIntroduce un texto al fichero: ");
	while((caracter = getchar()) != '\n'){
		printf("%c", fputc(caracter, fp));
	}
	fclose ( fp );
	return 0;
}
```

La función fputs()

La función **fputs** escribe una cadena de texto en un archivo. La ejecución de la misma no añade el carácter de retorno de línea ni el carácter nulo final. El valor de retorno es un *número no negativo* o **EOF** en caso de error. Los parámetros de entrada son la cadena a escribir y un puntero a la estructura **FILE** del fichero donde se realizará la escritura. El prototipo correspondiente de **fputs** es:

```
int fputs(const char *buffer, FILE *archivo);
```

Para ver su funcionamiento mostramos el siguiente ejemplo:

```
#include <stdio.h>

int main ( int argc, char **argv ){
	FILE *fp;
	char cadena[] = "Mostrando el uso de fputs en un fichero.\n";
	fp = fopen ( "fichero.txt", "r+" );
	fputs( cadena, fp );
	fclose ( fp );
	return 0;
}
```

La función fwrite()

Esta función está pensada para trabajar con registros de longitud constante y forma pareja con **fread**. Es capaz de escribir hacia un fichero uno o varios registros de la misma longitud almacenados a partir de una dirección de memoria determinada. El valor de retorno es el número de registros escritos, no el número de bytes. Los parámetros son: un puntero a la zona de memoria de donde se obtendrán los datos a escribir `buffer`, el tamaño de cada registro `size`, el número de registros a escribir `count` y un puntero a la estructura **FILE** del fichero al que se hará la escritura `stream`. El prototipo correspondiente de **fwrite** es:

```
size_t fwrite (void *buffer, size_t size, size_t count, FILE *stream);
```

Un ejemplo concreto del uso de **fwrite** es:

```
#include <stdio.h>

int main ( int argc, char **argv ){
	FILE *fp;
	char cadena[] = "Mostrando el uso de fwrite en un fichero.\n";
	fp = fopen ( "fichero.txt", "r+" );
	//char cadena[]... cada posición es de tamaño 'char'
	fwrite( cadena, sizeof(char), sizeof(cadena), fp );
	fclose ( fp );
	return 0;
}
```

La función fprintf()

La función **fprintf** funciona igual que printf en cuanto a parámetros, pero la salida se dirige a un archivo en lugar de a la pantalla. El prototipo correspondiente de **fprintf** es:

```
int fprintf(FILE *archivo, const char *formato, argumento, ...);
```

Podemos ver un ejemplo de su uso, abrimos el documento "fichero.txt" en modo lectura/escritura y escribimos dentro de él.

```c
#include <stdio.h>

int main ( int argc, char **argv )
{
	FILE *fp;
	char buffer[100] = "Esto es un texto dentro del fichero.";
	fp = fopen ( "fichero.txt", "r+" );
	fprintf(fp, buffer);
	fprintf(fp, "%s", "\nEsto es otro texto dentro del fichero.");
	fclose ( fp );
	return 0;
}
```

Varios

Función feof()

Cuando se abre un archivo para entrada binaria, se puede leer un valor entero igual de la marca EOF. Esto podría hacer que la rutina de lectura indicase una condición de fin de archivo aún cuando el fin físico del mismo no se haya alcanzado. Para resolver este problema, C incluye la función **feof()**, que determina cuando se ha alcanzado el fin del archivo leyendo datos binarios. La función tiene el siguiente prototipo:

```c
int feof (FILE *archivo);
```

Devuelve cierto (1) si se ha alcanzado el final del archivo; en cualquier otro caso, 0 (falso). También se puede aplicar este método a archivos de texto.

La función ferror()

Determina si se ha producido en error en una operación sobre un archivo. Su prototipo es:

```c
int ferror(FILE *archivo);
```

Donde `archivo` es un puntero a un archivo válido. Devuelve cierto si se ha producido un error durante la última operación sobre el archivo. En caso contrario, devuelve falso. Debido a que cada operación sobre el archivo actualiza la condición de error, se debe llamar a **ferror()** inmediatamente después de la operación de este tipo; si no se hace así, el error puede perderse.

La función remove()

Borra el archivo especificado. Su prototipo es el siguiente:

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

Devuelve cero si tiene éxito. Si no un valor distinto de cero.

La función fflush()

Escribe todos los datos almacenados en el buffer sobre el archivo asociado con un apuntador. Su prototipo es:

```
int fflush(FILE *archivo);
```

Si se llama esta función con un puntero nulo se vacían los buffers de todos los archivos abiertos. Esta función devuelve cero si tiene éxito, en otro caso, devuelve EOF.

La función rename()

Renombra un archivo, 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.