Fundamentos de Programación – Programación Estructurada en C:
11.- FICHEROS

Maider Huarte Arrayago

Fundamentos de Programación – Programación Estructurada en C:

11.- FICHEROS

11.- FICHEROS

Ficheros Físicos: Un fichero físico es un conjunto de datos en un dispositivo de almacenamiento electrónico, identificados bajo un nombre. Existe por sí mismo, y se puede acceder a él independientemente del lenguaje de programación.

Ficheros Lógicos: Un fichero lógico es un conjunto de datos que sólo existen en el programa en el que lo estemos usando. Son dependientes del lenguaje de programación y permiten el manejo de los ficheros físicos del sistema desde un programa en ejecución.

El acceso a los ficheros lógicos para el manejo de ficheros físicos, se puede hacer de dos formas:

- Como **Ficheros de texto**: Los bytes de un fichero de texto constituyen caracteres ASCII. Eso quiere decir, que ciertos caracteres tendrán un significado especial. El carácter 0x1A, p. e., marca el fin de fichero; se conoce como carácter EOF, y todos los ficheros de texto lo tienen como indicación de final. Si abrimos un archivo en modo texto, no será posible leer más allá de ese byte EOF. Los números, también se guardan como cadenas de caracteres.

- Como **Ficheros binarios**: En estos ficheros, los valores de los bytes no tienen ningún significado especial (no constituyen caracteres ASCII, p. e.). En estos archivos el final del fichero se detecta de otro modo, dependiendo del soporte y del sistema operativo. La mayoría de las veces se hace guardando la longitud del fichero. Para almacenar imágenes, p. e., se suelen usar este tipo de archivos. Los números se guardan según el tipo de dato (4 bytes para los int, 8 para los double,…).

La razón de existencia de esos dos modos de acceso deriva de las diferencias entre los sistemas operativos.

11.1.- GESTIÓN DE FICHEROS EN C

La gestión de ficheros en C se hace mediante un puntero a una estructura de tipo FILE. Esa estructura FILE está definida en la librería *stdio*, por lo que hay que importar esa librería en los programas en los que se vaya a trabajar con ficheros.

Se usan también una serie de funciones que están en la misma librería *stdio*.

En un primer apartado, se verán las funciones genéricas de acceso a ficheros, que se pueden usar con todo tipo de ficheros, independientemente del tipo que sean. En un segundo apartado, se verán las funciones exclusivas de Ficheros Binarios y en un tercero, las exclusivas de Ficheros de Texto.

Las funciones exclusivas tanto de Ficheros Binarios como de Texto, son las que realizan las tareas de escritura y lectura. La exclusividad de cada grupo, se refiere a su funcionamiento adecuado, es decir, el usar una función de escritura propia de ficheros binarios, para escribir en un fichero de texto, no produce errores de compilación, pero el resultado no está garantizado. Por eso, hay que tener cuidado a la hora de elegir las funciones que se van a usar, siempre en consonancia con la naturaleza (binaria o texto) del fichero.

Sin embargo, el significado de las acciones de escritura y lectura es la misma en ambos casos:

- Escribir: coger valor(es) de variable(s) del programa y copiarlo(s) en el fichero.
- Leer: coger un(os) valor(es) del fichero y pasarlo(s) a una(s) variable(s) del programa.

Además, tanto las funciones para ficheros binarios como las de ficheros de texto, después de realizar la tarea propia de escritura o lectura, se encargan también de actualizar el llamado *índice de lectura/escritura*, para poder leer o escribir el siguiente valor.

11.1.1.- Acceso a ficheros

11.1.1.1.- Apertura de un fichero

Tal como hemos descrito al principio, para trabajar con un fichero físico desde un programa, se ha de hacer mediante un fichero lógico. En C, para trabajar con un fichero lógico, primero hay que abrirlo, asociándolo al fichero físico con el que realmente queremos trabajar.

Para la apertura de un fichero lógico, tenemos la función ***fopen***, cuyo prototipo es:

FILE *fopen(const char *<nombre>**, const char ***<modo>**);**

La sintaxis de una llamada es:

FILE * <nombre_puntero_a_fichero>**;**
...
<nombre_puntero_a_fichero>**=fopen(**<nombre>**,**<modo>**);**

Se declara primero un puntero a fichero, para poder asignarle después el fichero lógico creado en la función *fopen*, para acceder al fichero físico cuyo nombre es el primer parámetro *nombre*. El parámetro *nombre* puede tener un *path* completo que indique la situación del fichero físico en la jerarquía de directorios del

computador. Si no se indica ningún *path*, se supone que el fichero físico se encuentra en la misma carpeta que el **ejecutable** del programa.

El fichero lógico se puede abrir de diferentes modos, que se especifican en el segundo parámetro pasado, *modo*. La apertura del fichero lógico, implica la existencia en memoria de un índice de lectura/escritura, que, dependiendo del modo de apertura, apuntará a una zona diferente del fichero.

Los diferentes modos de apertura de un fichero se especifican mediante las constantes cadena siguientes:

- **"r"**: Abrir **fichero de texto** para **lectura**. El índice de lectura/escritura se sitúa al inicio del fichero, de forma que la lectura (no puede haber escritura) de datos podrá empezarse desde el comienzo del fichero. Si el fichero físico indicado no existe, se devuelve un puntero NULL.
- **"w"**: **Crear fichero de texto** para **escritura**. El índice de lectura/escritura se sitúa al inicio del fichero, de forma que la escritura (no puede haber lectura) de datos podrá empezarse desde el comienzo del fichero. Si existe un fichero físico con el mismo nombre, se destruirá, porque se reemplazará por el nuevo fichero creado.
- **"a"**: Abrir **fichero de texto** para **añadir** algo al final. El índice de lectura/escritura apuntará a la última posición del fichero, para poder realizar escritura (no lectura) a partir de él. Si el fichero especificado no existe, se crea.
- **"r+"**: Abrir **fichero de texto** para **leer o escribir**. El índice de lectura/escritura se sitúa al inicio del fichero. Si el fichero físico indicado no existe, se devuelve un puntero NULL.
- **"w+"**: **Crear fichero de texto** para **leer o escribir**. El índice de lectura/escritura se sitúa al inicio del fichero. Si existe un fichero físico con el mismo nombre, se destruirá, porque se reemplazará por el nuevo fichero creado.
- **"a+"**: Abrir **fichero de texto** para **leer o añadir** algo al final. El índice de lectura/escritura apuntará a la última posición del fichero. Para leer, habrá que mover el índice de lectura/escritura a la posición adecuada. Si el fichero especificado no existe, se crea.
- **"rb"**: Igual que "r", pero para ficheros binarios.
- **"wb"**: Igual que "w", pero para ficheros binarios.
- **"ab"**: Igual que "a", pero para ficheros binarios.
- **"rb+"**: Igual que "r+", pero para ficheros binarios.
- **"wb+"**: Igual que "w+", pero para ficheros binarios.
- **"ab+"**: Igual que "a+", pero para ficheros binarios.

Tal como se ve en la lista de modos anterior, los modos para ficheros binarios son iguales que los de ficheros de texto, sólo que se les añade la letra 'b' para indicar que se trata de un fichero binario.

Hay que tener cuidado con los modos que empiezan con la letra 'w' (modos de creación), ya que si el fichero existe de antes, se destruirá.

En cualquier caso, siempre hay que comprobar que el fichero se abrió de forma correcta para poder trabajar con él. Si ha ocurrido algún error, se devolverá un puntero NULL de tipo FILE.

Ejemplos:

```
FILE *punt_fich;

if((punt_fich=fopen("binario.dat","rb+"))==(FILE *)NULL)
    printf("\nERROR AL ABRIR EL FICHERO");
```

Una vez que un puntero se asigna a un fichero abierto con cualquiera de los modos, no se puede usar para gestionar otro fichero. Habría que cerrar primero el primer fichero.

Hay una función que cierra un fichero abierto con *fopen* y abre (o crea, según el modo) otro, permitiendo que se asocie el puntero que apuntaba al primero, al segundo. Se trata de la función *freopen*, cuyo prototipo es el siguiente:

FILE *freopen(const char *<nombre>**,const char ***<modo>**,FILE ***<punt_fich>**);**

Lo que *freopen* hace es, cerrar el fichero asociado a *punt_fich*, abre (o crea, según el modo) el fichero de nombre *nombre* en el modo *modo*, y lo asocia al puntero *punt_fich*. Si la apertura tiene éxito, devuelve el puntero *punt_fich*, y si no, devuelve un puntero NULL.

Ejemplo:

```
/*
stderr es un puntero a fichero declarado en stdio.h. Mediante ese puntero, las indicaciones
de errores que ocurren en un programa se escriben en el fichero al que está asociado el
puntero stderr, que normalmente, suele ser el fichero que representa a la pantalla.
El siguiente programa hace que el puntero stderr se reasigne al fichero FREOPEN.OUT, y
que se escriba una línea en el mismo.

fprintf es una función que funciona parecido a printf, pero en vez de escribir la cadena de
control que se le indica en la pantalla, lo hace en el fichero cuyo puntero se le pasa como
primer parámetro.
*/

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

int main( void )
{
    fprintf( stderr, "Esto aparecerá escrito en la pantalla\n" );

    if(freopen( "freopen.out", "w", stderr ) == (FILE *)NULL )
        printf("\nError en el freopen" );
```

```
    else
    {
        fprintf( stderr, "Esto aparecerá escrito en 'freopen.out'\n" );
        printf("\nLa reasignación se realizó con éxito\n" );
        fclose( stderr );
    }

    return 0;
}
```

/* Se puede comprobar que después de la ejecución del programa, hay un fichero *freopen.out* en la misma carpeta que el ejecutable, y abriendo ese fichero con un editor de texto simple, se ve que la frase *Esto aparecerá escrito en freopen.out* aparece en ese fichero. */

11.1.1.2.- Cierre de un fichero

Una vez usado un fichero abierto mediante el puntero asociado, **hay que cerrar** ese fichero, **si no**, los ficheros abiertos **pueden corromperse y volverse inservibles**. Así, ese puntero se puede usar para abrir otro fichero. Para ello, se usa la función *fclose*, cuyo prototipo es el siguiente:

int fclose(FILE *<punt_fich>**);**

La función *fclose* cierra el fichero lógico apuntado por *punt_fich*, y actualiza el fichero físico con los cambios hechos sobre el lógico.

Si todo ha ido bien, devuelve un **0**. Si ha habido algún problema, por ejemplo, que se haya intentado cerrar un fichero que ya estaba cerrado, devuelve un **-1**.

11.1.1.3.- Final del fichero

Hay una función especial que indica si se ha intentado leer o escribir a partir del final del fichero. Es la función *feof*, cuyo prototipo es el siguiente:

int feof(FILE *<punt_fich>**);**

La función *feof* devuelve un **0 si** en una lectura o escritura **no** se ha intentado leer o escribir a partir del final del fichero que se le indica como parámetro.

Para poder usar esta función con un fichero, ha tenido que ser abierto de algún modo, y no cerrado.

11.1.1.4.- Errores de lectura/escritura fichero

La lectura/escritura de información de un fichero, puede producir errores. Para comprobar si hubo ese tipo de errores en el último intento de lectura o escritura, hay una función especial, *ferror*, cuyo prototipo es el siguiente:

```
int ferror(FILE *<punt_fich>);
```

La función *ferror* devuelve un **0 si no** se produjo ningún error de lectura/escritura en el fichero que se le indica como parámetro. Si devuelve un valor diferente de 0, es que ocurrió algún error. El número devuelto se corresponde con una clave de error, de forma que cada tipo de error tiene su propio número.

Para poder usar esta función con un fichero, ha tenido que ser abierto de algún modo, y no cerrado.

11.1.1.5.- Reiniciar un fichero

Reiniciar un fichero supone inicializar el índice de lectura/escritura como si el fichero hubiese sido abierto en ese mismo momento. Para ello, hay una función especial, *rewind*, cuyo prototipo es el siguiente:

```
void rewind(FILE *<punt_fich>);
```

Para poder usar esta función con un fichero, ha tenido que ser abierto de algún modo, y no cerrado.

11.1.1.6.- Borrar un fichero

Para borrar físicamente un fichero, hay una función especial, *remove*, cuyo prototipo es el siguiente:

```
int remove(char *<nombre>);
```

La función *remove* borra físicamente el fichero indicado en la cadena de caracteres que se le pasa como parámetro. Se diferencia con las funciones de gestión explicadas hasta ahora, en que el fichero que tiene que borrar no se le indica mediante un puntero al mismo, sino que se hace mediante una cadena de caracteres. Si no se indica ningún path en esa cadena de caracteres, se supone que el fichero está en la misma carpeta que el ejecutable del programa.

Devuelve un **0 si** el fichero se borró correctamente.

Para poder usar esta función con un fichero, el fichero tiene que estar cerrado.

11.1.1.7.- Renombrar un fichero

Para renombrar un fichero físico, hay una función especial, *rename*, cuyo prototipo es el siguiente:

int rename(char *<nombre1>**, char** *<nombre2>**);**

Mediante la función *rename*, el fichero físico cuyo nombre se pasa como 1er parámetro (*nombre1*), pasa a llamarse como la cadena de caracteres indicada como 2do parámetro (*nombre2*).

Devuelve un **0 si** el cambio de nombre se hizo correctamente.

Para poder usar esta función con un fichero, el fichero tiene que estar cerrado.

11.1.1.8.- Gestión del índice de lectura/escritura

Al abrir un fichero se le asocia un índice de lectura/escritura para poder leer y/o escribir de forma secuencial. El valor de ese índice se actualiza automáticamente con el valor de la dirección de la siguiente posición a leer o escribir, cuando se usan las funciones correspondientes.

Hay una serie de funciones que permiten hacer una gestión explícita de ese índice:

- **Función *fseek***:

  La función *fseek*, tiene el prototipo siguiente:

  int fseek(FILE *<punt_fich>**, long** <despl>**, int** <origen>**);**

  La función *fseek*, mueve el índice de lectura/escritura asociado al fichero pasado como 1er parámetro (***punt_fich***), a una posición indicada según los otros dos parámetros. Concretamente, la nueva posición se obtendrá sumando ***despl*** bytes a la posición indicada en ***origen***.

  El parámetro ***despl.*** puede ser positivo o negativo.

  El parámetro ***origen*** puede tomar 3 valores posibles:

- 0 (constante SEEK_SET): Indica el principio del fichero.
- 1 (constante SEEK_CUR): Indica la posición actual del índice de lectura/escritura.
- 2 (constante SEEK_END): Indica el final del fichero.

No se puede especificar un desplazamiento positivo desde el final del fichero (si usamos como 3er parámetro SEEK_END); en los otros dos casos si se usa un desplazamiento negativo, el resultado no está garantizado, por lo que se recomienda usar siempre desplazamientos positivos con SEEK_SET y SEEK_CUR.

Devuelve el valor **0 si** el desplazamiento se hizo con éxito.

- **Función *ftell***:

La función *ftell*, tiene el prototipo siguiente:

long ftell(FILE *<punt_fich>**);**

La función *ftell*, devuelve un valor que indica el desplazamiento en bytes del índice de lectura/escritura del fichero pasado como 1er parámetro (***punt_fich***), desde el inicio de ese fichero.

Devuelve valores negativos si ocurre algún error.

Ejemplo:

```
/*
Este programa crea un fichero binario fichero.out. Escribe dos líneas en él (44 caracteres en
total) y mueve el cursor a 16 caracteres antes del final. Indica la distancia del índice
lectura/escritura desde el origen, y lee 16 caracteres desde esa posición en adelante.
Para leer y escribir del fichero,  se usan respectivamente, las funciones fread  y fwrite, que
se explicarán más adelante.
*/

#include <stdio.h>

int main( void )
{
    FILE *punt_fich;
    char dos_lineas[]="Esta es la primera linea.\nY esta la segunda.";
    char lee_16[16];

    if((punt_fich = fopen( "fichero.out", "wb+" )) == NULL)
        printf( "\nError al crear el fichero" );
    else
    {
        fwrite(dos_lineas,sizeof(char),44,punt_fich); //Escribe los 1os 44 caracteres de dos_lineas
        if(fseek( punt_fich, -16, SEEK_END) ==0)
        {
            printf( "\nHemos puesto el índice a 16 caracteres ANTES del final.");
            printf( "\nPosición desde el inicio: %d\n",(int)ftell(punt_fich));
```

```
            fread( lee_16, sizeof(char), 16, punt_fich ); //Lee 16 caracteres y los guarda a partir de lee_16
            printf( "\nlee_16:%.16s", lee_16 );
            printf( "\nPosición desde el inicio: %d\n",(int)ftell(punt_fich));
        }
        else
            printf("\nError en el fseek");
        fclose( punt_fich );
    }

    return 0;
}
```

11.1.2.- Ficheros binarios

11.1.2.1.- Escritura

La escritura en un fichero binario se hace mediante la función *fwrite*, cuyo prototipo es la siguiente:

unsigned int fwrite(void *<punt>**, unsigned int** <tam>**, unsigned int** <nregs>**, FILE** *<punt_fich>**);**

La función *fwrite* escribe en el fichero indicado por el 4º parámetro (***punt_fich***), la cantidad indicada como 3er parámetro (***nregs***) de elementos del tamaño indicado en el 2º parámetro (***tam***), y que en memoria se encuentran almacenados a partir de la posición indicada por el 1er parámetro (***punt***).

Es decir, escribe en el fichero, una serie de elementos, todos del mismo tamaño, y que en memoria están almacenados a partir de una dirección concreta.

La escritura en el fichero se hará de forma secuencial. La función *fwrite* escribirá lo que se le indique a partir de la posición apuntada por el índice de lectura/escritura del fichero, y después actualizará su valor, para que apunte a la siguiente posición de memoria después de las escritas. Si al escribir se pasara del final del fichero, el tamaño del fichero quedaría cambiado después de la escritura con *fwrite*.

Devuelve un número entero positivo, que se corresponde con la cantidad de elementos de tamaño ***tam*** escritos correctamente. Si no hubo ningún error, el número devuelto será igual al 3er parámetro pasado en la llamada a la función (***nregs***).

Ejemplo:

```
/*
Este programa crea un fichero binario "fichero.out", y escribe en él alfabeto al revés, en una
línea. Después, lee lo escrito en el fichero, confirmando su contenido.
 */
```

```
#include <stdio.h>

int main( void )
{
    FILE *punt_fich;
    char cadena[26];
    int i, cant_leidos, cant_escritos;

    if( (punt_fich = fopen( "fichero.out", "wb+" )) != NULL )
    {
        for ( i = 0; i < 26; i++ )
            cadena[i] = (char)('z' - i);

        cant_escritos = fwrite( cadena, sizeof(char), 26, punt_fich );
        printf( "Se han escrito %d caracteres\n", cant_escritos );

        rewind(punt_fich);

        cant_leidos = fread( cadena, sizeof( char ), 26, punt_fich );
        printf( "Cantidad de caracteres leídos = %d\n", cant_leidos );

        printf( "Contenido de la cadena = %.26s\n", cadena );

        fclose( punt_fich );
    }
    else
        printf( "Error en la apertura del fichero\n" );

    return 0;
}
```

11.1.2.2.- Lectura

La lectura de elementos de un fichero binario se hace mediante la función *fread* cuyo prototipo es la siguiente:

unsigned int fread(void *<punt>**, unsigned int** <tam>**, unsigned int** <nregs>**, FILE** *<punt_fich>**);**

La función *fread* lee del fichero indicado por el 4º parámetro (***punt_fich***), la cantidad indicada como 3er parámetro (***nregs***) de elementos del tamaño indicado en el 2º parámetro (***tam***), y los escribe en memoria, a partir de la posición indicada por el 1er parámetro (***punt***).

Es decir, lee del fichero, una serie de elementos, todos del mismo tamaño, y los almacena en memoria a partir de una dirección concreta.

La lectura del fichero se hará de forma secuencial. La función *fread* leerá lo que se le indique a partir de la posición apuntada por el índice de lectura/escritura del fichero, y después actualizará su valor, para que apunte a la siguiente posición

después de las leídas. Si en la lectura se llegara al final del fichero, no se seguiría leyendo a partir de él.

Devuelve un número entero positivo, que se corresponde con la cantidad de elementos de tamaño ***tam*** leídos correctamente. Si no hubo ningún error, el número devuelto será igual al 3er parámetro pasado en la llamada a la función (***nregs***).

Ejemplo1:

```
/*
Abre el fichero fichero.dat como binario para lectura, y va leyendo cadenas de 100
caracteres de él, hasta llegar al final. Controla si ocurre algún error en la lectura. La última
cadena leída puede ser de menos caracteres que de 100, pero no se produce ningún error.
 */

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

int main( void )
{
    int error=0,contador, total = 0;
    char cadena_leida[100];
    FILE *punt_fich;

    if( (punt_fich = fopen( "fichero.dat", "rb" )) != NULL )
    {
        while( (feof( punt_fich )==0)&&error==0)
        {
            contador = fread( cadena_leida, sizeof( char ), 100, punt_fich );
            if( ferror( punt_fich )!=0 )
            {
                printf( "Ha habido algún error" );
                error=1;
            }
            else
                printf( "Caracteres leídos: %d\n", contador );
            total += contador;
        }
        printf( "Caracteres leídos en total: %d\n", total );

        fclose( punt_fich );
    }
    else
        printf("\nError al abrir el fichero");

    return 0;
}
```

11.1.3.- Ficheros de texto

11.1.3.1.- Escritura en un fichero de texto

La escritura en un fichero de texto se puede hacer de varias formas:

- **Función *putc***:

  La función *putc*, tiene el prototipo siguiente:

  int putc(char <c>**,FILE ***<punt_fich>**);**

  La función *putc*, escribe el carácter pasado como 1er parámetro (***c***), en la posición del índice de lectura/escritura asociado al fichero pasado como 2º parámetro (****punt_fich***), y actualiza ese índice para que apunte a la siguiente posición del fichero.

  Devuelve el mismo carácter escrito, si la escritura se realizó con éxito. Si hubo algún problema, devuelve el carácter EOF.

- **Función *fputs***:

  La función *fputs*, tiene el prototipo siguiente:

  int fputs(char *<cadena>**,FILE ***<punt_fich>**);**

  La función *fputs*, escribe todos los caracteres de la cadena de caracteres pasada como 1er parámetro (****cadena***) menos el carácter '\0', a partir del índice de lectura/escritura del fichero indicado como 2º parámetro (****punt_fich***). Después de la escritura, el índice de lectura/escritura queda actualizado al valor de la siguiente posición después de lo escrito.

  Devuelve **0 si** la escritura se realizó correctamente, si no, devuelve EOF.

- **Función *fprintf***:

  La función *fprintf*, tiene el prototipo siguiente:

  int fprintf(FILE *<punt_fich>**,char ***<cadena_control>**,…);**

  La función *fprintf*, funciona igual que la función ***printf***, sólo que en vez de escribir lo que se le indica en lo parámetros, en la pantalla, se escribe a partir del índice de lectura/escritura del fichero indicado en el 1er parámetro. Después de la escritura, se actualiza el índice, como en los anteriores casos.

  Devuelve el número de caracteres escritos en el fichero. Si devuelve un valor negativo, es que ocurrió algún error en la escritura.

11.1.3.2.- Lectura de un fichero de texto

La lectura en un fichero de texto se puede hacer de varias formas:

- **Función *getc***:

  La función *getc*, tiene el prototipo siguiente:

  int getc(FILE *<punt_fich>);

  La función *getc*, lee el carácter al que apunta el índice de lectura/escritura asociado al fichero pasado como parámetro (***punt_fich***), y actualiza ese índice para que apunte a la siguiente posición.

  Devuelve el código ASCII del carácter leído, si la lectura se realizó con éxito. Si no pudo leer ningún carácter (por algún problema o porque el índice de lectura/escritura está al final del fichero), devuelve el valor EOF.

- **Función *fgets***:

  La función *fgets*, tiene el prototipo siguiente:

  char *fgets(char *<cadena>,int <num>,FILE *<punt_fich>);

  La función *fgets* está diseñada para leer cadenas de caracteres. Leerá hasta ***num-1*** caracteres, siendo ***num*** el 2º parámetro pasado, o hasta que lea un carácter de retorno de línea ('\n'), del fichero pasado como 3er parámetro (***punt_fich***), y lo almacenará a partir de la posición de memoria apuntada por el puntero pasado como 1er parámetro (***cadena***).

  El parámetro ***num*** nos permite limitar la lectura para evitar desbordar el espacio disponible en ***cadena***. Si encuentra un carácter '\n' antes de leer los ***num-1*** caracteres para los que tiene espacio, deja de leer, introduce '\n' en la ***cadena***, y después introduce el carácter '\0' propio de los strings, en la ***cadena***.

  Devuelve el mismo puntero ***cadena** pasado como primer parámetro, si se leyó con éxito, y un valor (char *)NULL si en la lectura se hubo algún error. Si al llamar a la función, el índice de lectura/escritura apuntaba al final del fichero, es decir, al carácter EOF, se devolvería puntero (char *)NULL, ya que no se podrá leer nada del fichero. Si se ha podido leer algo antes del EOF, no habrá ningún error, aunque no se hayan podido leer todos los caracteres indicados por *num-1*.

Ejemplo:

```
/*
Este programa demuestra cómo se puede usar la función fgets para recoger una frase
introducida por el usuario desde el teclado. La frase podrá contener espacios en blanco y
tabulaciones y se podrá indicar una longitud máxima para la misma.
Así, se presenta como la mejor alternativa para recoger frases enteras introducidas por el
usuario por teclado.
*/

#include <stdio.h>

#define MAX 10

int main()
{
    char frase[MAX+1];
    char *direccion;

    printf("\n");

    direccion=fgets(frase,MAX+1,stdin);
    if(frase[strlen(frase)-1]=='\n')
        frase[strlen(frase)-1]='\0';

    printf("\nPuntero frase: %d",frase);
    printf("\nPuntero direccion: %d",direccion);
    printf("\nfrase: %s",frase);

    printf("\n\n");

    return 0;
}

/*
```

Caso 1:

```
Introduzca una frase (mßx. 10 caracteres): 0123456

Puntero frase: 1245044
Puntero direccion: 1245044
frase: 0123456
```

Se ha introducido una frase más corta que la que se esperaba. Para indicar que la frase se ha terminado, el usuario a pulsado *return*.

Caso 2:

```
Introduzca una frase (mßx. 10 caracteres): 0123456789012345

Puntero frase: 1245044
Puntero direccion: 1245044
frase: 0123456789
```

Se ha introducido una frase más larga que la que se esperaba. La función sólo ha recogido los caracteres para los que se ha reservado sitio (10).

Caso 3:

```
Introduzca una frase (mßx. 10 caracteres): 012 45       789012

Puntero frase: 1245044
Puntero direccion: 1245044
frase: 012 45   789
```

Se ha introducido una frase más larga que la que se esperaba, con caracteres espacio en blanco y tabulación. La función ha recogido los caracteres para los que se ha reservado memoria, incluyendo los espacios en blanco y las tabulaciones.
*/

- **Función *fscanf***:

La función *fscanf*, tiene el prototipo siguiente:

int fscanf(FILE *<punt_fich>,char *<cadena_control>,...);

La función *fscanf*, funciona igual que la función ***scanf***, sólo que en vez de leer lo que se le indica en lo parámetros, desde el teclado, lo lee a partir del índice de lectura/escritura del fichero indicado en el 1er parámetro.

Devuelve el número de conversiones de argumentos que se hayan realizado de forma correcta, según lo indicado en la cadena de control (***cadena_control***). Si se produjo algún error, devuelve EOF.

Ejemplo:

```
/*
Este programa crea un fichero de texto fichero.out y escribe en él, usando las funciones de
escritura para ficheros de texto, lo siguiente:
aabcdefghijklmnopqrstuvwxyz
i:
26

Siendo ese 26 el número de caracteres en el alfabeto de la 1ª línea. Después, usando las
funciones de lectura de ficheros de texto, lee su contenido y lo saca por pantalla.
*/

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


int main( void )
{
    char c='a',c_leido;
    char alfabeto[27],alfabeto_leido[30];
    char *cad_fscanf;
    int i,resul;

    FILE *punt_fich;
```

```
        punt_fich = fopen( "fichero.out", "w+" );
        if( punt_fich == NULL )
            printf( "\nError en la apertura del fichero" );
        else
        {
            putc(c,punt_fich);

            for ( i = 0; i < 26; i++ )
                alfabeto[i] = (char)('a' + i);
            alfabeto[26]='\0';

            fputs(alfabeto,punt_fich);
            putc('\n',punt_fich);


            fprintf( punt_fich, "i:\n%d",i);

            rewind(punt_fich);

            c_leido=getc(punt_fich);
            printf( "\nc_leido: %c", c_leido );

            if(fgets(alfabeto_leido,30,punt_fich)==NULL)
                printf("\nError al leer una línea");
            else
            {
                printf("\nalfabeto_leido: %s", alfabeto_leido);
                printf("\nlongitud alfabeto_leido: %d",strlen(alfabeto_leido));
            }

            cad_fscanf=(char *)malloc(strlen("i:")+1);

            fscanf( punt_fich, "%s\n%d", cad_fscanf,&i );

            printf( "\ncad_fscanf: %s",cad_fscanf);
            printf( "\n%d", i );

            fclose( punt_fich );
        }

        return 0;
    }
```