

Archivos

El estándar de C contiene funciones varias para la edición de archivos, estas están definidas en la cabecera `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`.

fopen

Esta función sirve para abrir y crear ficheros en disco.

El prototipo correspondiente de `fopen` es:

```
FILE * fopen (const char *filename, const char *opentype);
```

Los parámetros de entrada de `fopen` son:

`filename`: una cadena que contiene un nombre de archivo válido. `opentype`: especifica el tipo de archivo que se abrirá o se creará.

Una lista de parámetros `opentype` para la función `fopen` son:

- `"r"` : abrir un archivo para lectura, el fichero debe existir.
- `"w"` : abrir un archivo para escritura, se crea si no existe o se sobrescribe si existe.
- `"a"` : abrir un archivo para escritura al final del contenido, si no existe se crea.
- `"r+"` : abrir un archivo para lectura y escritura, el fichero debe existir.
- `"w+"` : crear un archivo para lectura y escritura, se crea si no existe o se sobrescribe si existe.
- `"a+"` : abrir/crear un archivo para lectura y escritura al final del contenido

Adicionalmente hay parámetros utilizando `"b"` (binary) los cuales no serán mostrados por ahora, los archivos binarios almacenan la información en forma binaria (en ceros y en unos) y en algunos casos son difíciles de leer por el usuario o el programador mediante un editor de textos como `emacs` o el bloc de notas.

fclose

Esta función sirve para poder cerrar un fichero que se ha abierto.

El prototipo correspondiente de `fclose` es:

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

Un valor de retorno cero indica que el fichero ha sido correctamente cerrado, si ha habido algún error, el valor de retorno es la constante EOF.

Un ejemplo pequeño para abrir y cerrar el archivo llamado fichero.in en modo lectura:

```
#include <stdio.h>

int main()
{
    FILE *fp;
    fp = fopen ( "fichero.in", "r" );
    fclose ( fp );

    return 0;
}
```

Como vemos, en el ejemplo se utilizó el opentype "r", que es para la lectura.

Otra cosa importante es que el lenguaje C no tiene dentro de sí una estructura para el manejo de excepciones o de errores, por eso es necesario comprobar que el archivo fue abierto con éxito "if (archivo == NULL)". Si fopen pudo abrir el archivo con éxito devuelve la referencia al archivo (FILE *), de lo contrario devuelve NULL y en este caso se deberá revisar la dirección del archivo o los permisos del mismo. En estos ejemplos solo vamos a dar una salida con un retorno de 1 que sirve para señalar que el programa terminó por un error.

feof

Esta función sirve para determinar si el cursor dentro del archivo encontró el final (end of file). Existe otra forma de verificar el final del archivo que es comparar el carácter que trae fgetc del archivo con el macro EOF declarado dentro de stdio.h, pero este método no ofrece la misma seguridad (en especial al tratar con los archivos "binarios"). La función feof siempre devolverá cero (Falso) si no es encontrado EOF en el archivo, de lo contrario regresará un valor distinto de cero (Verdadero).

El prototipo correspondiente de feof es:

```
int feof(FILE *fichero);
```

rewind

Literalmente significa "rebobinar", sitúa el cursor de lectura/escritura al principio del archivo.

El prototipo correspondiente de rewind es:

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

Lectura

Un archivo generalmente debe verse como un string (una cadena de caracteres) que está guardado en el disco duro. 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 tamaño, FILE *archivo)
- size_t fread(void *puntero, size_t tamaño, size_t cantidad, FILE *archivo);

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

Las primeras dos de estas funciones son muy parecidas entre si. 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.

fgetc

Esta función lee un caracter a la vez del archivo que esta 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 fuente de nuestro programa. Un pequeño programa que lea ese archivo será:

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

int main()
{
    FILE *archivo;
    char character;

    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 (feof(archivo) == 0)
        {
            character = fgetc(archivo);
            printf("%c", character);
        }

        return 0;
    }
}
```

fgets

Esta función está diseñada para leer cadenas de caracteres. Leerá hasta n-1 caracteres o hasta que lea un retorno de línea. En este último caso, el carácter de retorno de línea también es leído.

El prototipo correspondiente de fgets es:

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

El primer 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 es tamaño que es el limite en cantidad de caracteres a leer para la función fgets. Y por último el puntero del archivo por supuesto que es la forma en que fgets sabrá 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);

    printf("\nEl contenido del archivo de prueba es \n\n");
    while (feof(archivo) == 0)
    {
        fgets(caracteres, 100, archivo);
        printf("%s", caracteres);
    }
    system("PAUSE");
    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. Y resulta muy útil para algunos fines como la construcción de un parser de algún tipo de archivo de texto.

fread

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

Esta función lee un bloque de una "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 "ptr". 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 (size*count).

PARAMETROS:

ptr : 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.

fscanf

La función fscanf 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, ...);
```

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

```
#include <stdio.h>
int main ()
{
    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 archivo, también se pueden crear y escribir archivos 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, ...);

fputc

Esta función escribe un carácter a la vez del archivo que esta 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 escribira dentro del fichero hasta que presionemos la tecla enter.

```
#include <stdio.h>

int main ()
{
    FILE *fp;

    char character;

    fp = fopen ( "fichero.txt", "r+" );

    printf("\nIntrouce un texto al fichero: ");

    while((character = getchar()) != '\n')
    {
        printf("%c", fputc(character, fp));
    }

    fclose ( fp );

    return 0;
}
```

fputs

La función fputs escribe una cadena en un fichero. No se 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;
}
```

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, el tamaño de cada registro, el número de registros a escribir y un puntero a la estructura `FILE` del fichero al que se hará la escritura.

El prototipo correspondiente de `fwrite` es:

```
size_t fwrite(void *puntero, size_t tamano, size_t cantidad, FILE
*archivo);
```

Un ejemplo concreto del uso de `fwrite` con su contraparte `fread` y usando funciones es:

```
/*
 *      FicheroCompleto.c
 *
 *      Copyright 2009 Julio César Brizuela <brizuela@linux-qx1k>
 */

#include <stdio.h>

void menu();
void CrearFichero(FILE *Fichero);
void InsertarDatos(FILE *Fichero);
void VerDatos(FILE *Fichero);

struct sRegistro {
    char Nombre[25];
    int Edad;
    float Sueldo;
} registro;

int main()
{
    int opcion;
    int exit = 0;
    FILE *fichero;

    while (!exit)
    {
        menu();
        printf("\nOpcion: ");
        scanf("%d", &opcion);
```

```
        switch(opcion)
        {
            case 1:
                CrearFichero(fichero);
                break;
            case 2:
                InsertarDatos(fichero);
                break;
            case 3:
                VerDatos(fichero);
                break;
            case 4:
                exit = 1;
                break;
            default:
                printf("\nopcion no valida");
        }
    }

    return 0;
}

void menu()
{
    printf("\nMenu:");
    printf("\n\t1. Crear fichero");
    printf("\n\t2. Insertar datos");
    printf("\n\t3. Ver datos");
    printf("\n\t4. Salir");
}

void CrearFichero(FILE *Fichero)
{
    Fichero = fopen("fichero", "r");

    if(!Fichero)
    {
        Fichero = fopen("fichero", "w");
        printf("\nArchivo creado!");
    }
    else
    {
        printf("\nEl fichero ya existe!");
    }

    fclose (Fichero);

    return;
}
```



```
void InsertarDatos(FILE *Fichero)
{
    Fichero = fopen("fichero", "a+");

    if(Fichero == NULL)
    {
        printf("\nFichero no existe! \nPor favor creelo");
        return;
    }

    printf("\nDigita el nombre: ");
    scanf("%s", registro.Nombre);

    printf("\nDigita la edad: ");
    scanf("%d", &registro.Edad);

    printf("\nDigita el sueldo: ");
    scanf("%f", &registro.Sueldo);

    fwrite(&registro, sizeof(struct sRegistro), 1, Fichero);

    fclose(Fichero);

    return;
}

void VerDatos(FILE *Fichero)
{
    int numero = 1;

    Fichero = fopen("fichero", "r");

    if(Fichero == NULL)
    {
        printf("\nFichero no existe! \nPor favor creelo");
        return;
    }

    fread(&registro, sizeof(struct sRegistro), 1, Fichero);

    printf("\nNumero \tNombre \tEdad \tSueldo");

    while(!feof(Fichero))
    {
        printf("\n%d \t%s \t%d \t%.2f", numero, registro.Nombre,
            registro.Edad, registro.Sueldo);
        fread(&registro, sizeof(struct sRegistro), 1, Fichero);
        numero++;
    }
}
```

```
    fclose(Fichero);  
  
    return;  
}
```

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 el.

```
#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;  
}
```