Ficheros en C

Para C un fichero no es más que una porción de almacenamiento de memoria.

C representa un fichero como una estructura, de hecho el fichero **stdio.h** contiene la defínción de un fichero como si fuese una estructura:

```
struct _iobuf {
  char *_ptr;   /* Puntero al buffer actual  */
  int _cnt;     /* Contador del byte actual */
  char *_base;  /* Dirección base del buffer de E/S */
  char _flag;   /* Flags de control */
  char _file;   /* Número de fichero */
};

#define FILE  struct _iobuf;   /* Notación abreviada */
```

NO ES NECESARIO UTILIZAR LOS CAMPOS DE ESTE TIPO ESTRUCTURADO.

SOLO USAREMOS LA NOTACION ABREVIADA FILE EN NUESTROS PROGRAMAS.

- El fichero se maneja como una estructura.
- **FILE** corresponde al patrón del fichero. Cuando se manejen ficheros, se utilizará este tipo de dato estructurado.

Programa sencillo de lectura de fichero

El siguiente programa lee el contenido de un fichero llamado **TEST** y lo imprime por pantalla.

```
/* Nos dice qué hay en el fichero "test" */
#include <stdio.h>

main() {
 FILE *in;     /* Declara un puntero a un fichero */
 int ch;

 /* Abre el fichero test para lectura,
     comprueba si existe */

 if ((in = fopen("test", "r")) != NULL) {

  /* in: el puntero a FILE apunta ahora a "test"
     "test" es el nombre externo del fichero,
     ahora in es el nombre interno */

  while ((ch=getc(in)) != EOF ) putc (ch, stdout);
  /* Toma caracter de in y lo coloca en la pantalla */

  fclose(in); /* Cierra el fichero */
  }
  else
   printf ("No puedo abrir el fichero \"test\".\n");
}
```

En el ejemplo anterior:

fopen() Tiene tres parámetros:

- Nombre de fichero: cadena de caracteres.
- Uso del fichero: “r”, “w” o “a”.
- Puntero a fichero que devuelve la función.

  En el ejemplo es **in**.

  Este descriptor es el que se utilizará a posteriori para referirse al fichero.

No hace falta definir la función **fopen** en el programa como una función que devuelve un puntero a fichero porque ya está declarada dentro de **stdio.h**.

```
FILE * fopen();
```

Si **fopen** no consigue abrir el fichero, devuelve **NULL**.

Es importante realizar la comprobación de que se ha abierto correctamente el fichero antes de proceder a su utilización.

fclose() Cierra el fichero.

Observar que el argumento es el puntero a fichero **in** y no el descriptor del fichero.

Cuando ha ocurrido algún problema al cerrar un fichero se devuelve el valor -1, mientras que si el cierre del fichero ha sido satisfactorio devuelve 0.

Ficheros de texto con Buffer

Las funciones **fopen()** y **fclose()** trabajan con ficheros de texto con buffer: la entrada y salida se almacenan temporalmente en un área de memoria llamada el *buffer*.

Cuando el buffer se llena, el contenido se traspasa a un bloque de memoria, y se recomienza el proceso.

Una de las tareas principales de **fclose()** es el "vaciado" del buffer, que podría haber quedado parcialmente lleno al cerrar el fichero, y que puede ocasionar que realicemos una lectura errónea en el futuro.

También se puede utilizar la función *fflush()* para "literalmente" vaciar el contenido del buffer, sin cerrar el fichero.

Nota: un fichero de texto es aquel en el que la información se almacena como caracteres, utilizando su código ASCII o similar.

Un fichero binario es aquel en el que se almacena información binaria como puede ser un código en lenguaje máquina.

Las funciones de Entrada/Salida que vamos a describir a continuación están diseñadas únicamente para trabajar con ficheros de texto.

getc() y putc()

Funcionan de forma similar a **getchar()** y **putchar()**.

Además ha de especificarse el fichero sobre el que han de leer o escribir un carácter.

```
ch = getchar();
ch = getc(in);

putchar(ch);
putc(ch, out);
```

En ambos casos, tanto **in** como **out** han de ser:

```
FILE *in, *out;
```

En el ejemplo anterior aparecía **stdout**, que no es más que la salida estándar predefinida. De la misma manera **stdin** está definida como la entrada estándar predefinida.

El resto de funciones de entrada y salida de datos tienen su equivalente para el manejo de ficheros, con la única diferencia que tienen que llevar un descriptor de fichero como parámetro adicional.

fprintf() y fscanf()

Requieren un argumento adicional para indicar el descriptor de fichero, que se convierte en el primer argumento de la lista.

```
/* Formato para usar fprintf () y fscanf() */
#include <stdio.h>
main(){
 FILE *fi;
 int edad;

 /* Apertura en modo lectura */
 fi = fopen("pedro.txt", "r"); /* fi apunta a "pedro.txt" */
 fscanf (fi, "%d", &edad);
 fclose (fi);

 /* Modo añadir */
 fi = fopen ("datos.txt", "a"); /* fi apunta a "datos.txt" */
 fprintf (fi,"pedro tiene %d\n", edad);
 fclose(fi);
}
```

A diferencia de **getc()** y **putc()**, aquí el puntero es el primer argumento, mientras que allí es el último.

fgets() Utiliza tres argumentos:

- Puntero al lugar de destino de la línea que se va a leer.
- Longitud de la tira que se está leyendo. La función se detiene cuando se lee el carácter de nueva línea o cuando se han leido MAXLIN -1 caracteres.
- Puntero al fichero en el que se está leyendo.

Una diferencia entre **gets()** y **fgets()**, es que la primera convierte el carácter retorno de carro en $'\backslash 0'$, mientras que fgets lo mantiene.

Además devuelve el símbolo NULL cuando ha leído un fin de fichero (EOF).

Ejemplo:

```
/* Lee de un fichero una linea cada vez */
#include <stdio.h>

#define MAXLIN 80

main() {
FILE *fi;

char *tira[MAXLIN];

fi = fopen("cuanto","r");
while (fgets(tira, MAXLIN, fi) != NULL)
    puts(tira);
}
```

fputs() Simplemente añade un argumento adicional al final de fputs() que es un puntero a fichero.

```
fputs("Esto es un ejemplo", fi);
```

donde **fi** debe ser un puntero a fichero previamente abierto mediante una sentencia fopen().

La forma general de uso es:

control = fputs(puntero-a-tira, puntero-a-fichero)

donde *control* es un entero que toma el valor EOF si se encuentra una marca de fin de fichero o un error.

Al igual que **puts()** esta función no copia el $'\backslash 0'$ del final de la tira. Pero tampoco añade un caracter de nueva línea a la salida.

Acceso aleatorio con fseek()

La funcion **fseek()** permite tratar los ficheros como vectores, moviendose directamente a un byte determinado del fichero abierto por **fopen()**.

Obviamente, esta función sólo se puede utilizar sobre ficheros que admitan acceso aleatorio.

Los argumentos de **fseek()** son los siguientes:

- Puntero a fichero.
- OFFSET: indica la distancia a que debemos movernos desde el punto de comienzo. Este parametro debe declararse de tipo **long** y puede ser positivo o negativo (movimiento hacia delante o hacia atras).
- Indicador del punto de referencia para el offset.

Modo	**El offset se mide desde**
0	el comienzo del fichero
1	posicion actual
2	fin del fichero

La funcion **fseek()** devuelve el valor 0 si todo ha ido bien, y devuelve el valor -1 si ha habido algun error.

Argumentos en línea de comando

main(argc, argv)
int argc;
*char *argv[];*

int argc: indica número de argumentos, incluído el nombre del programa ejecutable

*char *argv[]*: argv[i] es el (i+1)-ésimo argumento. El primero es el nombre del programa.

```
/* fseek() para imprimir el contenido de un fichero */
#include <stdio.h>
/* No es obligatorio usar argc y argv */
main(numero, nombres)
int numero;
char *nombres[];
{
 FILE *fp;
 long offset = 0L;      /* 0 de tipo long */
 if (numero < 2)
   puts ("Necesito un fichero como argumento\n");
 else {
  if ((fp =fopen(nombres[1], "r")) == 0)
    printf ("No puedo abrir %s", nombres[1]);
  else {
   while (fseek(fp, offset++, 0) == 0)
     putchar(getc(fp));
   fclose(fp);
  }
 }
 }
```

```
/* Reduce un fichero a su tercera parte */
#include <stdio.h>
main(argc, argv)
int argc;
char *argv[];
{
 /* Declara dos punteros a fichero */
 FILE *in, *out;
 int ch;
 static char nombre[20]; /* Fichero de salida */
 int con = 0;
 if (argc < 2) /*¿Existe fichero de entrada?*/
  printf ("No hay nombre de fichero como argumento.");
 else  {
   if ((in = fopen(argv[1], "r" )) != NULL){
      /* Copia el nombre del fichero en un array*/
     strcpy (nombre, argv[1]);
      /* Crea fichero de extensión reducida */
     strcat (nombre, ".red");
       /* Abre  fichero como salida de escritura */
     out = fopen (nombre, "w");
       /* Envía un carácter de cada tres */
     while ((ch = getc(in)) != EOF)
      if (cont++ % 3 == 0)
        putc (ch, out) ;
     fclose(in);
     fclose(out);
  }
  else
    printf ("No puedo abrir el fichero %s\n", argv[1]);
 }
}
```