estudios abiertos
SEAS
GRUPO SANVALERO

4

4 UNIDAD DIDACTICA

Estructura de Datos

4. Ficheros

ÍNDICE

OBJETIVOS

- Comprender y aplicar el concepto de flujo en el lenguaje C.
- Desarrollar las operaciones básicas con ficheros.
- Manejar los distintos modos de trabajo que existen en C para con los flujos.
- Estudiar que funcionalidades aportan las distintas librerías en cuanto al manejo y gestión de ficheros.

INTRODUCCIÓN

Veremos el peculiar concepto de fichero en el lenguaje C, este término se denomina flujo de datos porque se asemeja a un trayecto origen – destino de los propios datos.

Examinaremos profundamente como se accede a ficheros y de que formas es posible hacerlo, todo ello ilustrado con claros ejemplos explicativos.

También nos apoyaremos en las funciones de la librería <stdio.h> que serán nuestro motor de operación.

4.1. El flujo

En este apartado vamos a estudiar el concepto de flujo de datos y veremos los flujos estándares existentes en programación C.

4.1.1. Concepto de flujo

El concepto de flujo es un concepto abstracto representativo de una operación de movimiento de datos, donde desde un origen ciertos datos son enviados hacia un destino.

El término flujo también se puede encontrar referenciado por su equivalente en lengua inglesa del que es traducción "**stream**".

Así mismo también podemos hallar multitud de sinónimos para los términos origen y destino como pueden ser fuente, productor, iniciador, etc., para el primero y sumidero, consumidor, finalizador, etc., para el segundo.

Se dice que entre el origen y el destino se establece lo que se denomina canal que es por donde circulan los datos.

Este término también se suele denominar por pipe, debido a su herencia de UNIX.

Veamos una representación de un flujo:

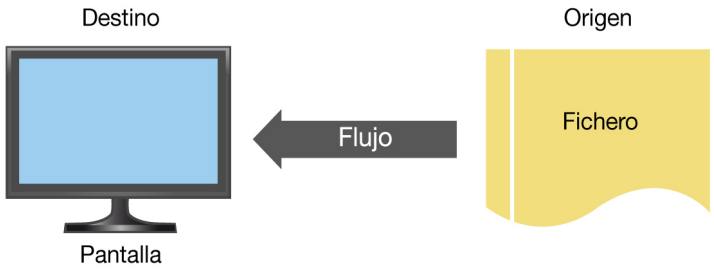

Figura 4.1. Diagrama representativo del concepto de flujo.

4.1.2. Flujos estándares en C

El lenguaje C nos brinda 3 flujos o canales, abiertos y dispuestos para su utilización directa, que están en relación con el uso de los dispositivos más frecuentes, veamos cuales son:

stdin

Se define como una variable externa del tipo:

DEFINICIÓN

```
extern FILE * stdin;
```

Está directamente asociada a la entrada estándar que es el teclado.

stdout

Su definición es similar:

DEFINICIÓN

```
extern FILE * stdout;
```

Está vinculada a la salida estándar, que no es otra que la pantalla.

stderr

Al igual que las anteriores se define como un puntero a fichero:

DEFINICIÓN

```
extern FILE * stderr;
```

Está vinculada a la salida estándar de errores, y está al igual que stdout, dirigida a la pantalla.

Aunque parezca un concepto extraño podemos decir que ya hemos trabajado con flujos, por ejemplo, cada vez que se hace uso de printf implícitamente estamos trabajando con stdout, si se quiere tomar una entrada del usuario mediante scanf, entonces estamos trabajando con la entrada estándar stdin del teclado.

AVANZADO

El lenguaje C permite redirigir cualquier flujo e incluso los estándares. También permite duplicarlos.

En C el trabajo con archivos está siempre intervenido por un buffer, así se permite trabajar con secuencias de datos independientemente de la naturaleza de las misma, pudiendo trabajar siempre de una forma estandarizada y cómoda.

4.2. Punteros a fichero: file *

C establece en su librería estándar <stdio.h> una estructura específica para el manejo de ficheros.

Esta estructura permite manejar archivos mediante un nombre lógico o interno para su fácil identificación, además almacena información relativa a la dirección del buffer que se encarga de su manejo, el control de caracteres, tipo de apertura, etc.

La estructura definida FILE se implementa de la siguiente forma:

```
typedef struct{
  short          level;
  unsigned       flags;
  char           fd;
  unsigned char  hold;
  short          bsize;
  unsigned char *buffer, *curp;
  unsigned       istemp;
  short          token;
} FILE;
```

No se trabaja directamente con la estructura sino con un puntero a dicha estructura: FILE *, este es el “modus operandi” de todas las funciones de <stdio.h> involucradas en el manejo de ficheros.

4.3. Trabajo con ficheros

A partir de este momento nos centraremos en cómo se implementan y aplican todas las funcionalidades que C brinda bajo su librería estándar < stdio.h > para operar con los distintos tipos de ficheros que podemos encontrar.

RECUERDA

Ni C ni ninguna de sus librerías estándar nos aportan funcionalidades para el trabajo con ficheros indexados (Revisar anexo final donde se explican las diferencias entre ficheros indexados, ficheros secuenciales y ficheros de acceso directo).

4.3.1. Apertura de ficheros

Es la primera operación que se debe realizar con un archivo antes de poder trabajar con él.

Con ello conectamos el fichero físico con nuestra representación lógica de dicho fichero, esto es, con un FILE *.

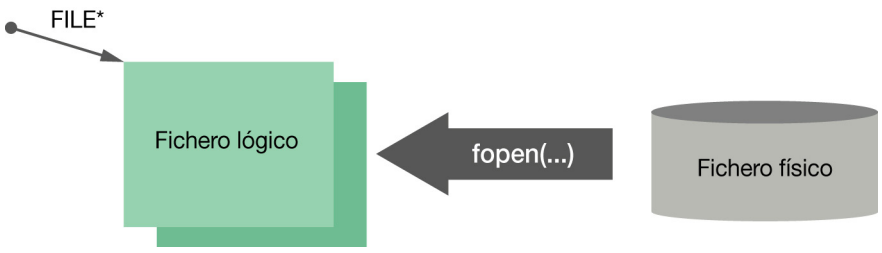

Figura 4.2. Diagrama representativo de la apertura de ficheros.

Para eso se utiliza la función fopen cuyo prototipo es el siguiente:

DEFINICIÓN

FILE * **fopen**(const char* nombre_fichero, const char* modo);

A la función fopen se le deben pasar como argumentos el nombre del fichero físico con su ruta completamente cualificada, y el modo de acceso que deseamos, (a continuación veremos los distintos modos de apertura de un fichero).

La función nos devolverá un FILE * que contendrá la dirección a una estructura FILE que nos permitirá referenciar el fichero.

AVISO

Si el fichero no ha podido abrirse se devuelve NULL, en caso de apertura satisfactoria se deberá recoger el puntero devuelto para referenciar al fichero.

Pero no olvide consultar siempre si el valor del FILE * devuelto ha sido NULL, ya que en caso afirmativo no se podrá operar con el fichero con el que se pretendía trabajar.

4.3.1.1. Modos de apertura

Veamos los distintos modos en los que un fichero puede ser abierto, a continuación se expondrán valores válidos para el segundo parámetro de fopen.

r	Apertura únicamente de lectura.
w	Apertura para escritura, si el fichero ya existe lo crea nuevo vacío. Todos los datos que pudieran existir se pierden.
a	Apertura para escritura al final del fichero. Añade datos. Escribe a partir de la marca anterior de fin de fichero.
r+	Apertura para actualización (Lectura y/o Escritura). Es requisito imprescindible que el fichero tenga existencia previa.
w+	Apertura para actualización (Lectura y/o Escritura), si el fichero ya existe lo crea nuevo vacío. Todos los datos ya existentes se perderán.
a+	Apertura para actualización al final del archivo (Lectura y/o Escritura), si el fichero no existe lo crea.

El lenguaje C distingue entre 2 tipos o clases de ficheros atendiendo a la naturaleza de sus datos:

- Ficheros de Texto.
- Fichero Binarios.

Para indicar a qué tipo de ficheros queremos acceder indicaremos en el modo de apertura una t si el fichero de trabajo es un fichero de texto, mientras que indicaremos con una b si se trata de un fichero binario.

Así tendremos por tanto los siguientes modos de apertura válidos:

rt, rb, wt, wb, at, ab, r+t, r+b, w+t, w+b, a+t y a+b.

NOTA

Normalmente los compiladores sino se indica tipo de fichero entiende fichero de texto, por lo tanto los siguientes modos de apertura también son válidos:
r, w, a, r+, w+, a+

A continuación y, a modo de ejemplo, se presenta un código con la apertura de un fichero de texto en modo lectura. Para comprobar si el fichero se ha abierto o no, ejecuta el código sin que el fichero ejemplo.txt exista, luego crea el fichero ejemplo.txt en la ruta indicada y vuelve a ejecutar el programa. Verás la diferencia en la ejecución.

EJEMPLO

```
#include <stdio.h>

void main()
{
  FILE* pfLector;

  char* NombreFichero= "C:\\ejemplo.txt";

  int iSalida;

  if((pfLector=fopen(NombreFichero, "rt"))==NULL)
{
  printf("\nNo se pudo abrir el fichero...");
  iSalida = 1;
}else{
  printf("\nEl fichero se abrió correctamente");
  int iSalida = 0;
  fclose(pfLector);
}

  getch();
}
```

4.3.2. Macros

Cuando se trabaja con fichero hay un par de macros incluidas en <stdio.h> que son muy utilizadas:

Macro NULL

Esta macro ya fue estudiado en la unidad referente a gestión de memoria, más concretamente en el manejo de punteros.

Este NULL se utiliza de igual modo que el que ya fue descrito, ya que FILE *, es realmente un puntero, y se utiliza para comprobar si dicho puntero a fichero almacena una dirección de memoria válida o por el contrario el puntero no apunta a nada.

Macro EOF

Esta macro se utiliza para detectar el final de fichero su origen está en la función feof cuyo prototipo es el siguiente:

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

Esta función devuelve un valor distinto de 0 sino se alcanzó la marca de fin de fichero.

4.3.3. Cierre de ficheros

Es una buena costumbre el cerrar los fichero que ya no se van a utilizar, e incluso cerrarlos, si momentáneamente no van a ser accedidos. Esto evita muchos problemas de concurrencia. No obstante los sistemas operativos cierran los ficheros cuando se destruyen sus descriptores, para que no se produzcan inconsistencias de datos.

Esto evita muchos problemas de concurrencia.

No obstante los sistemas operativos cierran los ficheros cuando se destruyen sus descriptores, para que no se produzcan inconsistencias de datos.

Pero es buena costumbre el cerrarlos en nuestros programas.

Este hecho de cerrar los ficheros, es más importante cuando tratamos con operaciones de escritura, ya que el hecho de cerrar un fichero implica llevar el contenido del buffer al dispositivo físico. De no hacer esta operación, podría suceder que no se actualizase el fichero físico con el buffer intermedio.

Para cerrar ficheros se utiliza la función fclose cuyo prototipo es el siguiente y que ya ha sido utilizada en el anterior código:

```
int fclose(FILE * puntero  a  fichero);
```

Esta función devuelve un resultado de valor cero (0) si la operación de cierre resulto satisfactoria.

4.4. Funciones para trabajar con ficheros

Daremos una buena batida a las principales funciones para la gestión de ficheros que nos ofrece la librería estándar <stdio.h>

4.4.1. Lectura de un carácter

Para la lectura de caracteres podemos hacer uso de **fgetc** cuyo prototipo es el siguiente:

```
int fgetc(FILE * flujo);
```

Lee el siguiente carácter de un FILE*, y devuelve el valor en forma de entero sin signo.

Cuando se produce un error o se alcanza la marca de fin de fichero se devuelve EOF.

4.4.2. Escribir un carácter

Para escribir un carácter en el flujo asociado <stdio.h> nos brinda la función **fputc** cuyo prototipo es:

```
int fputc (int caracter, FILE*  flujo);
```

A fputc, le pasamos el carácter que deseamos escribir, y la dirección del flujo destino.

Si la operación se realizó con éxito se devuelve el propio carácter, sino se devuelve EOF.

4.4.3. Lectura de cadenas

Para recuperar una cadena de caracteres desde el archivo asociado, disponemos de la función **fgets**:

Su prototipo es:

```
char* fgets(char* cadena, int n, FILE* flujo);
```

Las cadenas como tales no existen en el lenguaje C, cuando se habla de cadenas nos estamos refiriendo a secuencias de char, (char [ ]).

La recuperación de la cadena se lleva a cabo hasta que es leído el carácter de fin de línea (\n) o hasta alcanzar el valor de n-1 caracteres leídos.

Si la operación se concluyó con éxito se devuelve un puntero a la cadena recuperada en caso contrario se envía un valor NULL.

4.4.4. Escritura de cadenas

Para escribir una cadena de caracteres en un flujo disponemos de la función **fputs**.

El prototipo de fputs se enuncia de la siguiente forma:

```
int fputs(const char* cadena, FILE* stream);
```

A la función se le pasa como argumentos la dirección de la cadena que se desea enviar al flujo destino, y la dirección del propio flujo.

Si la cadena acaba en '\0' la copia como tal, sino añade un salto de línea y la copia sin la marca de fin '\0'.

Si la operación concluye con éxito devuelve un valor positivo, sino devuelve EOF.

4.4.5. Lectura de datos binarios

La función para la lectura de estructuras binarias es **fread**, esta lee de un archivo de n bloques de bytes y los deja en un buffer específico apuntado por un puntero sin tipo.

El prototipo de la función fread es:

```
size_t fread(void *ptr, size_t t, size_t n, FILE * flujo)
```

Esta función recibe como parámetros:

- Un puntero sin tipo que referenciará a la estructura donde se almacenarán los datos leídos.
- El tamaño del elemento que va a ser leído.
- El número de elementos que se leerán.
- Un puntero de tipo fichero, que referencia el fichero donde están almacenados los datos. Esto es el origen del flujo.

4.4.6. Escritura de datos binarios

Para escribir estructuras binarias disponemos de la función **fwrite**.

Su prototipo responde a la siguiente firma:

```
size_t fwrite(const void* ptr, size_t size, size_t n, FILE* flujo);
```

Esta función, es su correspondiente para escritura de datos binarios a la anteriormente descrita fread, que se utiliza para su lectura. Los parámetros de fwrite son los siguientes:

- Un puntero sin tipo que referenciará a la estructura que contiene la dirección de los datos que van a ser escritos.
- El tamaño del elemento que va a ser escrito.
- El número de elementos de este tipo que se escribirán.
- Un puntero de tipo fichero, que referencia el flujo donde se escribirán los datos. El destino de los datos.

La función devuelve el número de elementos que realmente fueron escritos, este valor se suele utilizar para comprobar si la escritura se ejecutó correctamente.

4.4.7. Funciones para ficheros de acceso directo

A continuación se describe otro set de funciones de la librería < stdio.h> utilizadas en fichero cuyo tipo de acceso es directo o aleatorio.

Rebobinado

Esta función se denomina **rewind,** y no es exclusiva de ficheros de acceso directo, aunque debido a su modo de operación se incluye en esta categoría.

Mediante **rewind** situamos el puntero del flujo referenciado al comienzo del mismo.

Veamos su prototipo:

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

Posicionamiento

Los ficheros de acceso directo permiten situarse en una posición puntual a lo largo del fichero, la función utilizada para implementar esta operatoria es fseek, veamos cual es su prototipo:

```
int fseek(FILE * flujo, long desplazamiento, int referencia);
```

La función recibe los siguientes parámetros:

- Flujo objeto del desplazamiento.
- Cantidad de bytes desplazados, expresados en un valor tipo long.
- Referencia de desplazamiento, desde donde partiremos para desplazarnos.

NOTA

Para referenciar el desplazamiento < stdio.h> nos brinda 3 constantes en forma de macros:

SEEK_SET, cuyo valor se corresponde con 0. Y qué referencia el inicio del flujo.

SEEK_CUR, su valor correspondiente es 1, y referencia la posición actual.

SEEK_END, le corresponde el valor 2, y referencia la posición final.

Obsérvese que SEEK_SET, y SEEK_CUR desplazarán hacia el final del fichero con valores positivos, mientras que con SEEK_END, ocurre exactamente lo contrario.

También <stdio.h> nos brinda otra función complementaria que es ftell, utilizada para consultar la posición actual del puntero expresada como el número de bytes desplazados desde el inicio del flujo.

Su prototipo responde a la siguiente redacción:

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

RESUMEN

- En el lenguaje C, los ficheros se tratan como flujos, los flujos implican un transporte de datos desde un origen a un destino.
- El lenguaje C hace una especial distinción entre ficheros de texto y ficheros binarios.
- En C cuando hablamos de ficheros siempre tenemos que tener presente que habrá un buffer intermedio.
- Existen diversos modos de apertura de un fichero en función de su naturaleza y de la operatoria que se vaya a realizar.
- Es importante el cerrar los ficheros sobre todo en procesos de escritura para actualizar determinados flags del fichero y para forzar la escritura del buffer.
- La librería estándar < stdio.h > nos proporciona un amplio set de funciones para el trabajo con ficheros.
- Dentro de la propia < stdio.h > hay un subconjunto de funciones para tratar con ficheros de acceso directo.
- Ni C, ni ninguna de sus librerías estándares da soporte a ficheros indexados.