Apunte ficheros/archivos C

Apunte realizado por Santiago Boero para la cátedra de Sintaxis y Semántica de los Lenguajes, Universidad Tecnológica Nacional FRBA.

Introducción

En el siguiente apunte se tratarán cuestiones relacionadas a los ficheros en C, a modo tal de poder comprender mejor ciertos conceptos y funciones que serán utilizados en los trabajos prácticos.

Para poder usar las funciones que vamos a estar mencionando, deben tener incluida la biblioteca **<stdio.h>**

Primero veamos, **¿Qué es un fichero/archivo?**

Bueno un fichero es una estructura de datos almacenada en memoria secundaria, donde si queremos guardar o recuperar/utilizar información de un fichero, vamos a tener que realizar una serie de pasos u operaciones.

El primer paso sería declarar una variable de tipo fichero, lo cual se hace de la siguiente manera:

FILE *nombre_variable_fichero;

Con esto, estamos declarando a una variable "nombre_variable_fichero" como un puntero hacia una estructura de tipo FILE, que va a ser el archivo que nosotros queremos utilizar.

Bien, ¿Ahora qué sigue? Ahora vamos a la parte de apertura y cierre de ficheros.

APERTURA Y CIERRE DE FICHEROS

Antes de poder utilizar un fichero, primero debemos realizar la **operación de apertura** del mismo. Luego, si queremos almacenar datos en él, tendremos que realizar una **operación de escritura** y si queremos obtener datos que están en el archivo realizaremos una **operación de lectura.** Una vez que terminamos de usar el archivo, como último paso, deberíamos realizar una **operación de cierre** del archivo, para así liberar la memoria que este ocupando.

¿Como abrimos el fichero?
La forma habitual de abrir el fichero es la siguiente:

```
FILE *fichero;

fichero = fopen(nombre_fichero,modo);
```

Acá utilizamos la función ***fopen***. Esta función nos devuelve un puntero a un fichero, y se lo estamos asignando a la variable "fichero" que declaramos previamente con el mismo tipo de variable.

Si llegase a ocurrir algún error al realizar esta operación, por ejemplo, porque queremos abrirlo para leerlo pero el archivo no existe, la función nos devolverá el valor de **NULL**.

El **nombre_fichero** es básicamente el nombre (y en su caso la ruta de acceso) del archivo tal y como aparece para el sistema operativo.

El modo nos indica el tipo del fichero (texto o binario) y el uso que le vamos a dar, ya sea para escritura, lectura, solo añadirle datos al final, etc. Los modos que hay son los siguientes:

- **r** abre un fichero para lectura. Si el fichero no existe devuelve error.
- **w** abre un fichero para escritura. Si el fichero no existe se crea, si el fichero existe se destruye y se crea uno nuevo.
- **a** abre un fichero para añadir datos al final del mismo. Si no existe se crea.
- **+** símbolo utilizado para abrir el fichero para lectura y escritura.
- **b** el fichero es de tipo binario.

- **t** el fichero es de tipo texto. Si no se pone ni b ni t el fichero es de texto.

Los modos anteriores se combinan para conseguir abrir el fichero en el modo adecuado.

Por ejemplo, para abrir un fichero binario ya existente para lectura y escritura el modo será "rb+"; si el fichero no existe, o aun existiendo se desea crear, el modo será " wb+ ". Si deseamos añadir datos al final de un fichero de texto bastará con poner "a", etc.

Una buena práctica que se puede utilizar con la función fopen es utilizarla dentro de una sentencia condicional (IF) para poder ver si se produjo algún error al intentar abrirlo o si se efectuó con éxito, por ejemplo:

```
FILE *fichero;
    if ((fichero = fopen("nomfichero.dat", "r")) == NULL)
{ /* control para el error de apertura */
    printf ("Error en la apertura. Es posible que el fichero no exista \n");
}
```

Recordemos que si durante la apertura del fichero, ocurre un error, la función **fopen** retorna **NULL**, por lo que la comparación daría verdadero e imprimiría el mensaje que pusimos.

Cuando se termine de utilizar el fichero, es importantísimo cerrarlo para liberar la memoria que se está utilizando.
Esto se hace con fclose(fichero);

LECTURA Y ESCRITURA EN ARCHIVOS

Para guardar datos en un archivo, tenemos que realizar una operación de escritura, así como para obtener datos tenemos que hacer una operación de lectura.

En C tenemos muchas funciones para leer y escribir en un archivo, algunas son: **fread, fwrite, fgetc, fputc, fgets, fputs, fscanf, fprintf.**

Lectura y escritura de bloques (fread – fwrite)

Para leer y escribir en ficheros, las operaciones que se deben utilizar son **fread y fwrite**.

La forma en la que se escriben estas funciones es la siguiente:

```
fwrite (direcc_dato, tamaño_dato, numero_datos, punt_fichero);
```

Analicemos cada parte:

- **Direcc_dato**: es la dirección del dato que se quiere ingresar en el archivo. Si el dato lo asignamos a una variable, podemos colocar la variable con el referenciador &.

- **Tamaño_dato**: acá tenemos que colocar el tamaño que va a necesitar en memoria lo que queremos escribir en el archivo, por ejemplo, si queremos introducir un INT, necesitamos el tamaño que ocupa un INT en memoria. Como hay tantos tipos de datos y la verdad no es necesario acordarse cuanto ocupa cada uno, lo que podemos utilizar allí es la función **sizeof()**. Esta función lo que devuelve es el espacio que ocupa un tipo de dato. Por lo cual, allí podremos colocar **sizeof(Int)**, y devolverá el espacio que ocupa un int (4 bytes).

- **Numero_datos**: Indica la cantidad de datos que vamos a ingresar en el archivo.

- **Punt_fichero**: es el fichero donde queremos escribir el dato.

Por ejemplo:

```
FILE *salida;

salida = fopen ("Salida.txt","w");

int unNumero = 5;

fwrite(&unNumero,sizeof(unNumero),1,salida);

fclose(salida);
```

De esta manera hicimos los siguiente:

1. Declaramos la variable salida como un puntero a un tipo de dato archivo.
2. Le asignamos a salida el puntero que apunta hacia el archivo "salida.txt", en donde vamos a escribir el valor.
3. Declaramos un entero de con valor 5.
4. Lo que se hace en el **fwrite** es: Escribir en el archivo el número que tiene almacenado unNumero, en este caso el numero 5; indicamos el espacio que va a tener lo que va a escribir. Colocamos sizeof(unNumero) para que almacene según el tipo de dato que contenga esa variable, esto sirve mucho para cuando no sabemos bien el tamaño o lo que podrá contener dicha variable. En este caso tiene el tamaño de un Int; Escribe todo en el archivo salida.
5. Cerramos el archivo.

Todo esto es para escribir en bloque dentro de un archivo.

Para leer en bloque, lo que vamos a usar es:

```
fread (direcc_dato, tamaño_dato, numero_datos,punt_fichero);
```

En este caso vemos que tenemos la misma estructura que el fwrite, la diferencia acá es que en **direcc_dato** ahora se va a guardar el dato que se lea del archivo. Los demás campos se comportan de la misma forma (tamaño_dato indica el tamaño del dato que se va a leer del archivo; numero_datos indica la cantidad de datos que se deben leer; punt_fichero es el archivo que se debe leer).

Un ejemplo seria:

```
FILE *entrada;

entrada = fopen ("Entrada.txt","r");

int unNumero;

fread(&unNumero,sizeof(unNumero),1,entrada);

fclose(entrada);
```

Ahora, **¿Qué pasaría si queremos leer más de un dato a la vez?**

Bueno, lo que hay que entender en este caso, es que cada dato que se lea ocupara un espacio distinto en memoria, por lo cual, si quisiésemos por ejemplo leer 3 datos, requeriríamos que se almacenen en un vector de 4 espacios, como en el siguiente ejemplo:

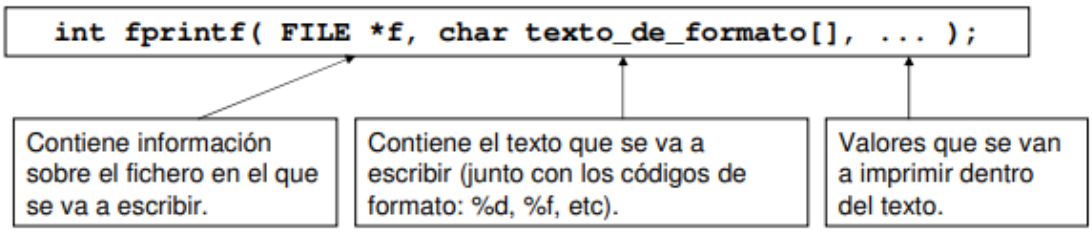
```
FILE *entrada;

entrada = fopen("Entrada.txt","r");

int vector_numeros[5];

fread(&vector_numeros[0], sizeof(vector_numeros), 3, entrada);

fclose(entrada);
```

En este caso, le estamos indicando al fread que lea 3 datos del archivo entrada. ¿Cómo los va a almacenar? Los va a almacenar desde la posición 0 hasta la posición 2, cada dato ordenado dentro del vector.

Esta seria una forma de leer de a varios datos a la vez, pero hay otras formas, como por ejemplo con punteros.

Otras formas de lectura y escritura

- fprintf: Escribirá en el archivo lo que se declarará en la cadena. Su estructura es la siguiente:



Por ejemplo:

```
FILE *salida;

salida = fopen ("Salida.txt","w");

fprintf(salida,"Fin del archivo");
```

Esto nos escribirá "Fin del archivo" en el archivo "Salida.txt"

- fgetc: Esta función sirve para una lectura carácter a carácter del archivo. Cuando llega al final del archivo, devuelve EOF. Su estructura es:

carácter_leido = fgetc (fichero);

Por ejemplo:

```
FILE *entrada;

entrada = fopen ("Entrada.txt","r");

caracter = fgetc(entrada);
```

Acá caracter va a tener el valor del primer carácter del archivo "Entrada.txt"

- fputc: Escribe un carácter en el archivo indicado.
  Su estructura es:

  fputc(carácter, archivo)

  Por ejemplo:

```
FILE *salida;

salida = fopen ("Salida.txt","w");

char caracter = 'a';

fputc(caracter, salida);
```

  Esto escribirá el carácter a en el archivo "Salida.txt"

- fgets: Esta función nos permite leer más de un carácter en el archivo.
  Su estructura es:

  fgets (cadena_leida, num_caracteres, archivo);

  Acá, en cadena_leida se van a almacenar la cantidad de caracteres que le indicamos leer a la función en num_caracteres.

  Por ejemplo:

```
FILE *entrada;

entrada = fopen ("Entrada.txt","r");

char caracter[5];

fgets(caracter,3,entrada);
```

Almacenara en carácter los primeros 3 caracteres que aparezcan en el archivo entrada.

- fputs: Escribe la cadena indicada en el archivo. Su estructura es:

fputs (cadena_escribir, fichero);

Por ejemplo:

```
FILE *salida;

salida = fopen ("Salida.txt","w");

char caracter[15] = "Practica fputs";

fputs(caracter, salida);
```

Acá escribirá en el archivo salida la cadena “Practica fputs”