Archivos - Guía Rápida

Los ficheros, en contraposición con las estructuras de datos vistas hasta ahora (variables simples, vectores, registros, etc.), son estructuras de datos almacenadas en memoria secundaria.

Para utilizar la información en memoria principal se emplea fundamentalmente la instrucción de asignación; sin embargo, para guardar o recuperar información de un fichero es necesario realizar una serie de operaciones que describiremos en este apartado.

El formato de declaración de un fichero es el siguiente:

```
FILE * nomVarFich;
```

En otros lenguajes la declaración del fichero determina el tipo de datos que se van a almacenar en él. En C la filosofía es distinta, todos los ficheros almacenan bytes y es cuando se realiza la apertura y la escritura cuando se decide cómo y qué se almacena en el mismo; durante la declaración del fichero no se hace ninguna distinción sobre el tipo del mismo.

En la operación de **apertura** se puede decidir si el fichero va a ser de **texto o binario**, los primeros sirven para almacenar caracteres, los segundos para almacenar cualquier tipo de dato.

Si deseamos leer un fichero como el autoexec.bat utilizaremos un fichero de **texto**, si queremos leer y escribir registros (struct) usaremos un fichero **binario**.

Apertura Y Cierre De Ficheros

Hasta ahora, para obtener y almacenar datos de una estructura de datos bastaba con realizar asignaciones a la misma. Para utilizar los ficheros el procedimiento es distinto.

Antes de usar un fichero es necesario realizar una **operación de apertura** del mismo; posteriormente, si se desea almacenar datos en él hay que realizar una **operación de escritura** y si se quiere obtener datos de él es necesario hacer una **operación de lectura**. Cuando ya no se quiera utilizar el fichero se realiza una **operación de cierre** del mismo para liberar parte de la memoria principal que pueda estar ocupando (aunque el fichero en sí está almacenado en memoria secundaria, mientras está abierto ocupa también memoria principal).

La instrucción más habitual para abrir un fichero es:

```
FILE * fichero;
fichero = fopen ( nombre-fichero, modo);
```

La función **fopen** devuelve un puntero a un fichero que se asigna a una variable de tipo fichero. Si existe algún **tipo de error** al realizar la operación, por ejemplo, porque se desee abrir para leerlo y éste no exista, devuelve el valor **NULL**.

El **nombre-fichero** será una cadena de caracteres que contenga el nombre (y en su caso la ruta de acceso) del fichero tal y como aparece para el sistema operativo.

El **modo** es una cadena de caracteres que indica el tipo del fichero (texto o binario) y el uso que se va ha hacer de él lectura, escritura, añadir datos al final, etc. Los modos disponibles son:

- **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.

La forma habitual de utilizar la instrucción fopen es dentro de una sentencia condicional que permita conocer si se ha producido o no error en la apertura, por ejemplo:

```
FlLE *fich = fopen("nomfich.dat", "r");
if (fich != NULL) { // Si se abrió correctamente
    // podemos trabajar con el archivo
}
else{ // control del error de apertura
    printf("Error en la apertura. Es posible que el fichero no exista");
}
```

El resultado de **fopen()** se almacena en la variable fich y después se compara fich con NULL para saber si se ha producido algún error. Toda la operación se puede realizar en la misma instrucción, tal y como aparece en el ejemplo.

Cuando se termine el tratamiento del fichero hay que cerrarlo; si la apertura se hizo con fopen el cierre se hará con **fclose(fich)**;

Para utilizar las instrucciones de manejo de ficheros que veremos en esta unidad es necesario incluir la librería <stdio.h>.

Lectura Y Escritura En Ficheros

Para almacenar datos en un fichero es necesario realizar una operación de escritura, de igual forma que para obtener datos hay que efectuar una operación de lectura. En C existen muchas y variadas operaciones para leer y escribir en un fichero; entre ellas tenemos: fread -fwrite, fgetc -fputc, fgets - fputs, fscanf -fprintf

Es aconsejable utilizarlas por parejas; es decir, si se escribe con fwrite se debe leer con fread.

Lectura y escritura de bloques (fread – fwrite)

Para leer y escribir en ficheros que no sean de texto las operaciones que se deben utilizar son fread y fwrite.

El formato de escritura en bloque es el siguiente:

```
fwrite (direccDato, tamanioDato, numeroDatos, puntFichero);
```

Escribe tantos datos como indique número de datos en el fichero, tomando los datos a partir de la dirección del dato.

Los datos tienen que tener tantos bytes como especifique tamaño. La función fwrite devuelve el **número de elementos escritos**, este valor debe coincidir con numero de datos.

Para calcular el tamaño en bytes de un dato o un tipo de dato se suele utilizar la función **sizeof (dato) o sizeof (tipo-de-dato)**;

Por ejemplo:

int i, v[3]; → sizeof (i) daría lo mismo que sizeof (int)
→ sizeof (v) daría 3 veces el resultado de sizeof (V[1])

Por ejemplo:

```
FlLE *f;
int v[6], elemEscritos, num;
f = fopen ("datos.cht ", "wb ");
```

/* Para escribir los 3 últimos elementos de v (el 2, el 3 y el 4) */

```
elemEscritos = fwrite (&v[2], sizeof(int), 3, f );
```

/* Para escribir el primer elemento de v, valen las 2 instrucciones siguientes */

```
fwrite (v, sizeof (int), 1, f );
fwrite (&v[0], sizeof(int), 1, f );
```

/* Para escribir un entero valen las dos siguientes */

```
fwrite (&num, sizeof(int), 1, f);
fwrite (&num, sizeof(num), 1, f);
```

La sentencia de lectura de bloque es la siguiente:

```
fread (direccDato, tamanioDato, numeroDatos,puntFichero);
```

Lee tantos datos como indique número de datos del fichero, colocando los datos leídos a partir de la dirección del dato. Los datos tienen que tener tantos bytes como especifique tamaño del dato. La función fread devuelve el número de elementos leídos, y el valor devuelto debe coincidir con número de datos.

Ejemplos:

```
f = fopen ("datos.dat ", "rb ");
elemEscritos = fread (&v[2], sizeof(int), 3, f);
fread (v, sizeof(int), 1, f);
fread (&V[0], sizeof(int), 1, f);
fread (&num, sizeof(int), 1, f);
fread (&num, sizeof(num), 1, f);
```

Lectura y escritura formateada de texto (fscanf – fprintf)

Las instrucciones scanf y printf utilizadas normalmente para lecturas y escrituras de teclado y pantalla tienen sus correspondientes funciones para el manejo de ficheros: **fscanf y fprintf**. La única diferencia con las anteriores es la necesidad de dar como primer argumento el fichero en el que leemos o escribimos.

Para que lo escrito con fprintf pueda ser correctamente leído con fscanf es conveniente que el formato en el que se indican los tipos de datos que se van a leer o escribir sean similares para

ambas instrucciones, que los tipos de datos estén separados, por ejemplo, por un blanco y que **tengan un fin de línea al final.**

Por ejemplo, los siguientes pares de instrucciones de lectura y escritura serían compatibles:

```
int num;
char car, cad [10];
FILE *f. /* apertura del fichero */
.....
.....
fscanf (f, "%d %c %s ", &num, &car, cad);
fprintf ( f, "%d %c %s \n ", num, car, cad);
fscanf (f, "%s %c %d ", cad, &car, &num);
fprintf (f, "%s %c %d \n ", cad, car, num);
```

Lectura y escritura de caracteres (fgetc – fputc) y cadenas (fgets – fputs)

Los formatos de las instrucciones de lectura y escritura de caracteres y cadenas son:

```
caracterLeido = fgetc(fechero);
```

fgetc lee un carácter del fichero, el carácter leído se almacenará en carácter leído. Cuando se llega al final del fichero devuelve EOF.

```
fputc (car, fichero);
```

fputc escribe el carácter car en el fichero. Devuelve el carácter escrito o EOF en caso de error..

```
fgets (cadena_leida, num_caracteres, fichero);
```

Lee num_caracteres del fichero y los almacena en cadena_leida colocando el carácter de fin de cadena '\0' en la posición num_caracteres de la cadena leida.

Por ejemplo:

```
FlLE *f;
char cad[5];
fgets (cad, 5, f);
```

almacena en cad la cadena " Buen " si se lee la línea " Buenas tardes " del fichero f. Si se realizan dos lecturas, en la segunda se almacena en cad la cadena “as t ".

```
fputs (cadenaEscribir, fichero);
```

escribe la cadena en el fichero.
Por ejemplo:.

```
fputs (cad, f);
```

Ejemplo : Copiar un fichero de texto en otro.

```
#include <stdio.h>
#include <stdlib.h>
int main ( ) {
    FILE *fin = fopen("DATOSIN.DAT", "rt");
    FILE *fout = fopen("DATOSOUT.DAT" , "wt");
    char c, x;
    if ((fin != NULL) && (fout != NULL)){
        c = fgetc(fin);
        while (c != EOF) {
            x = fputc (c, fout);
            if (x!= c){
               printf ("Error de escritura");
            }
            c = fgetc(fin);
        }
        fclose (fin);
        fclose (fout);
    }
    else{
        printf ("Error en la apertura de ficheros de salida \n" );
    }
    return 0;
}
```

Recorrido De Un Fichero Secuencial (feof)

Las lecturas y escrituras en un fichero se realizan en la posición en la que se encuentra el puntero del fichero. Al abrir el fichero el puntero está antes del primer dato del mismo. Cada vez que se realiza una operación de lectura o escritura el puntero se mueve hasta apuntar al dato y después lo lee o escribe.

Por ejemplo, en un fichero f con dos datos (O y 1) al abrir el fichero tendríamos la siguiente situación:

Posición del puntero	puntero→		
Datos	0	1	eof

Si realizamos la operación fread(&i, sizeof(int), 1, f); en la variable i tendremos almacenado el 0 y la situación después de la lectura será:

Posición del puntero	puntero→		
Datos	0	1	eof

Para leer todos los datos de un fichero basta con realizar lecturas sucesivas hasta que se lee el final del fichero. En un ejemplo anterior hemos visto cómo se puede detectar el final del fichero teniendo en cuenta el dato leído (leer hasta que fgetc( ) devuelva EOF). Esta operación es correcta, pero es más habitual utilizar una función de C que nos indica cuándo se ha leído el último dato:

```
feof (fichero);
```

devuelve un valor distinto de 0 cuando se ha alcanzado el final del fichero.
Ejemplo Escribir cinco registros en un fichero y leerlo posteriormente.
Solución:

```
#include <stdio.h>
typedef struct {
    int num;
    char cad[10];
    char car;
} t_reg;
int crearFichero () {
    FILE *fich = fopen("fichreg.dat", "wb");
    int i, er_dev = 0;
    t_reg r;
    if ((fich != NULL) {
        for (i = 0; i < 5; i + + ) {
            r.num = i;
            r.car='a'+1;
            printf("De un nombre: ");
            gets(r.cad);
            fwrite(&r, sizeof(r), 1, fich);
        }
```

```
        fclose (fich);
    }
    else {
        printf ("Error en apertura del fichero para escritura\n");
        er_dev = 1;
    }
    return er_dev;
}
int leerFichero (){
    FILE *fich = fopen("fichreg.dat", "rb");
    t-reg r;
    int er_dev = 0;
    if ((fich != NULL){
        fread (&r, sizeof(r), 1, fich);
        while (!feof(fich)){
            printf ("%d: %s: %c\n" , r.num, r.cad, r.car);
            fread (&r, sizeof(r), 1, fich);
        }
        fclose (fich);
    }
    else {
        printf ( "Error en apertura del fichero para lectura \n " );
        er_ dev = 1;
    }
    return er_dev;
}

int main() {
    int error;
    error = crearFichero();
    if (!error){
        leerFichero();
    }
    return 0;
}
```

Acceso Directo A Los Datos (Fseek)

Cuando se lee un dato de un fichero y después el que está a continuación de él, y así sucesivamente, se dice que se está realizando una lectura secuencial del mismo. Cuando se puede acceder a cualquier dato de un fichero sin tener que pasar por anteriores se está

realizando un acceso directo a los datos. La función que permite situarse en un determinado dato del fichero es:

```
fseek (fichero, posicion, origen);
```

que coloca el puntero del fichero a tantos bytes del origen como indica posición contando a partir del origen señalado. Los orígenes posibles son:
SEEK_SET o 0: principio del fichero.
SEEK_CUR o 1: posición actual.
SEEK_END o 2 : final del fichero.

fseek devuelve 0 si no ha habido ningún error.

Por ejemplo, en un fichero que almacene números enteros la instrucción

```
fseek (f, 0, SEEK_SET); //colocará el puntero al principio del fichero.
fseek (f, 3*sizeof(int), SEEK_CUR); //colocará el puntero 3 posiciones más allá de la posición actual del puntero.
```

Para saber cuál es la posición en la que está el puntero del fichero C proporciona la función siguiente: **ftell(fich);** que devuelve la posición actual en bytes del puntero del fichero con respecto al principio del mismo.