INTRODUCCIÓN

Las distintas estructuras tratadas hasta el momento, se consideran o se denominan internas (se almacenan en memoria central). Sin embargo, en numerosas ocasiones es preciso conservar los datos de modo permanente en algún dispositivo de almacenamiento, de manera que puedan existir, mientras se crean o manipulan programas u otros conjuntos de datos.

Los archivos son estructuras de datos almacenados en memoria secundaria o externa.Es una estructura constituida por colecciones de datos que se pueden guardar (grabar) para usar posteriormente mediante la ejecución de un determinado programa.

Archivos

Un archivo o fichero (file) es una colección de información relacionada entre sí (registros) con aspectos en común y organizados para un propósito específico. Los datos están organizados de tal modo que pueden ser recuperados fácilmente, actualizados, borrados y almacenados nuevamente en el archivo con todos los cambios realizados.

Los ficheros permiten guardar información en un dispositivo de almacenamiento de modo que esta "sobreviva" a la ejecución de un programa.

Es muy común tratar los datos de un archivo utilizando una estructura llamada registros y equivalen a las estructuras (struct) del lenguaje C. Un archivo así entendido es una colección de registros que poseen la misma estructura interna, se compone de una serie de campos que pueden ser de tipos distintos (incluso un campo podría ser una estructura o un array). En cada campo los datos se pueden leer según el tipo de datos que almacenen (enteros, caracteres, etc).

Como ya mencionamos, si bien los archivos son información binaria, para su uso en lenguaje C vamos a distinguir dos tipos, los archivos de texto y los archivos binarios.

Clasificación de los archivos

Existen dos tipos de archivos que tiene que ver con su contenido.

Archivos de texto

Es una secuencia de caracteres organizados en líneas terminadas por un carácter de nueva línea. Se caracterizan por ser planos, es decir todos las letras tienen el mismo formato y no hay palabras subrayadas, en negrita, o letras de distinto tamaño.

Los archivos de texto, contienen datos legibles por una persona y puedes generarlos o modificarlos desde tus propios programas o usando aplicaciones como los editores de texto. Aunque un problema puntual con el que muchas veces hay que lidiar es que es necesario usar marcas de separación entre sus diferentes elementos. Estas marcas son caracteres que decide el programador, pero es corriente que se trate de espacios en blanco, tabuladores o saltos de línea.

Los archivos de texto son, en principio, más portables, pues la tabla ASCII es un estándar ampliamente aceptado para el intercambio de ficheros de texto. No obstante, la tabla ASCII es un código de 7 bits que solo da cobertura a los símbolos propios de la escritura en inglés y algunos caracteres especiales. Los caracteres acentuados, por ejemplo, están excluidos.

Los códigos más usados son:

• ASCII. Código de 7 bits que permite incluir 128 caracteres. En ellos no están los caracteres nacionales por ejemplo la 'ñ' del español, ni símbolos de uso frecuente (matemáticos, letras griegas, etc.). Por ello se usó el octavo bit para producir códigos de 8 bits, llamado ASCII extendido (lo malo es que los ASCII de 8 bits son diferentes en cada país).
• ISO 8859-1. El más usado en occidente. Se la llama codificación de Europa Occidental. Son 8 bits con el código ASCII más los símbolos frecuentes del inglés, español, francés, italiano o alemán entre otras lenguas de Europa Occidental.
• Windows 1252. Windows llama ANSI a esta codificación. En realidad, se trata de un superconjunto de ISO 8859-1 que es utilizado en el almacenamiento de texto por parte de Windows.
• Unicode. La norma de codificación que intenta unificar criterios para hacer compatible la lectura de caracteres en cualquier idioma. Hay varias posibilidades de aplicación de Unicode, pero la más utilizada en la actualidad es la UTF-8 que es totalmente compatible con ASCII y que usando el octavo bit con valor 1 permite leer más bytes para poder almacenar cualquier número de caracteres (en la actualidad hay 50000)

Archivos binarios

Los archivos binarios, por contra, no están pensados para facilitar su lectura por parte de seres humanos (al menos no directamente). Por otro lado, requieren una mayor precisión en la determinación de la codificación de la información.

Escribir varios datos de un mismo tipo en un archivo binario no requiere la inserción de marcas separadoras, cada cierto número de bytes empieza un nuevo dato, por lo tanto es fácil decidir dónde empieza y acaba cada dato.

Existen dos tipos de archivos que tiene que ver con su modo de acceso.

Archivos secuenciales

Se trata de archivos en los que el contenido se lee o escribe de forma continua. No se accede a un punto concreto del archivo, para leer cualquier información necesitamos leer todos los datos hasta llegar a dicha información. En general son los archivos de texto los que se suelen utilizar de forma secuencial.

Archivos de acceso directo.

Se puede acceder a cualquier dato del archivo conociendo su posición en el mismo. Dicha posición se suele indicar en bytes. En general los archivos binarios se utilizan mediante acceso directo. De cualquier manera, cualquier archivo en C puede ser accedido de forma secuencial o usando acceso directo.

El puntero a un archivo

En el programa el archivo tiene un nombre interno que es un puntero a una estructura predefinida (puntero a archivo). Esta estructura contiene información sobre el archivo, tal y como la dirección del buffer que utiliza, el modo de apertura del archivo, el último carácter leído y otros detalles que generalmente el usuario no necesita saber. El identificador del tipo de estructura es FILE (definida en el archivo de cabecera stdio.h).

El detalle de los campos tipo FILE puede cambiar de un compilador a otro. Lo importante es conocer que existe el tipo FILE y que es necesario definir un puntero a FILE por cada archivo a procesar.

La sintaxis para declarar un puntero a un archivo es:

```
FILE *Identificador_Archivo
```

Veamos un ejemplo de declaración de un puntero a un tipo FILE:

```c
#include <stdio.h>
int main()
{
    FILE *f;//f es el identificador que elegimos para el puntero
    ...
    return 0;
}
```

Apertura de un archivo

Para procesar un archivo en C (y en todos los lenguajes de programación) la primera operación a realizar es abrir el archivo. La apertura del archivo supone conectar el archivo externo con el programa, e indicar cómo va a ser tratado: binario, de texto, etc. El programa accede a los archivos a través de un puntero a la estructura FILE, y es la función de apertura *fopen(),* la encargada de devolver dicho puntero.

El prototipo general de la función es la siguiente:

```
FILE *fopen(char *Nombre_Archivo, char *modo);
```

El *Nombre_archivo* es una cadena de caracteres que representan un nombre válido de archivo y puede incluir una especificación de directorio [Path]. por ejemplo: "C:\Prueba.txt"

Modo, apunta a una cadena que contiene el estado de apertura deseado. La siguiente tabla muestra los valores básicos permitidos para especificar el modo de apertura del archivo:

Modo	Significado
r	Abre para lectura
w	Abre para crear un nuevo archivo (si existe se pierden sus datos)
a	Abre para añadir al final
r+	Abre archivo ya existente para modificar (leer/escribir)
w+	Crea un archivo para escribir/leer (si existe, se pierden los datos)
a+	Abre el archivo para escribir/leer al final. Si no existe es como w+

En estos modos no se ha establecido el tipo de archivo, texto o binario. La opción por defecto, (aunque depende del compilador utilizado), suele ser modo texto. Para no depender del entorno es mejor indicar si es texto o binario. Se utiliza la letra "t" para modo texto, la letra "b" para modo binario, como último carácter de la cadena *modo.*

Por consiguiente los modos para abrir un archivo de texto son:

"rt", "wt", "at", o bien "r+t", "w+t", "a+t".

Los modos para abrir un archivo binario son:

"rb", "wb", "ab", o bien "r+b", "w+b", "a+b".

La forma genérica de escribir una sentencia de apertura a un archivo es la que se muestra a continuación:

```
...
FILE *f;//Puntero a un archivo
f = fopen(Nombre_Archivo, modo);
//Nombre_Archivo, es la ruta y nombre del archivo
...
```

La función *fopen()* devuelve un puntero de tipo FILE. Este puntero identifica el archivo y se usa en la mayoría de las otras funciones del sistema de archivos. No debe ser nunca alterado por el código.

Como muestra la tabla anterior un archivo se puede abrir en modo texto o en modo binario. En modo texto, las secuencias retorno de carro/salto de línea se convierten en caracteres de salto de línea en la lectura. En la escritura, ocurre lo contrario; los caracteres de salto de línea se convierten en retorno de carro/salto de línea. No ocurre lo mismo en archivos binarios.

Las funciones de biblioteca que devuelven un puntero, especifican que si no pueden realizar la operación, devuelven NULL. La función *fopen()* puede detectar un error al abrir el archivo; por ejemplo, el archivo no existe, o la ruta de acceso no es la correcta.

Ejemplo: fragmento de código para abrir un archivo de texto para lectura de manera correcta.

```
...
    FILE *f;
    f = fopen("prueba.txt", "r");//Abre para lectura
    if (f == NULL){
        printf("Error de apertura de archivo\n");
    }
    else
    {    //Apertura correcta
        //Se recorre el archivo para obtener información
    }
...
```

Cierre de archivos

Los archivos en C trabajan con una memoria intermedia o buffer. La entrada y salida de datos se almacena en ese buffer. Al terminar la ejecución del programa puede ocurrir que haya datos en el buffer; si no se vuelcan en el archivo este quedaría sin las actualizaciones. Siempre que se termina de procesar un archivo se termina la ejecución del programa, los archivos abiertos hay que cerrarlos justamente para que entre otras acciones se vuelque el contenido del buffer.

La función *fclose()* es la encargada de cerrar el archivo asociado al puntero file. El prototipo de la función es:

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

`Identificador_Archivo` es el puntero al archivo devuelto por la llamada a *fopen().* Si devuelve un valor cero significa que la operación de cierre ha tenido éxito, en otro caso se devuelve un número distinto de cero.

La forma genérica de escribir una sentencia de cierre de un archivo teniendo en cuenta la declaración del ejemplo anterior de apertura es la que se muestra a continuación:

```
fclose(f)//Puntero al archivo abierto previamente
```

Eliminar un archivo

La función del lenguaje C que permite eliminar un archivo especificado es *remove().* Su prototipo general es:

```
int remove(char *Nombre_Archivo);
```

El Nombre_Archivo es el nombre que recibe el archivo en el disco y la ruta si es necesario, por ejemplo:

```
remove("C:\Prueba.txt");
```

Esta función devuelve cero si tiene éxito y un valor distinto de cero si falla.

Procesamiento de archivos de texto

La biblioteca de C proporciona diversas funciones para escribir datos en el archivo a través del puntero a FILE asociado. Las funciones de entrada y de salida tienen mucho parecido con las funciones utilizadas de entrada y salida para los flujos stdin (teclado) y stdout (pantalla): *printf(), scanf(), gets(), puts()*; estas funciones tienen una versión para archivos que comienzan por la letra *f* ; con lo cual se dispone de: *fprintf*(), *fscanf*(), *fputc*(), *fgetc*().

Lectura de caracteres

Las dos funciones *getc()*, y *fgetc()* son iguales, tienen el mismo formato y la misma funcionalidad. Leen caracteres secuencialmente del archivo asociado al puntero a FILE. Devuelven el carácter leído o EOF si es fin de archivo.

Los prototipos de ambas funciones se muestran a continuación respectivamente:

```
int getc(FILE *Identificador_archivo);
```

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

Para procesar un archivo secuencial en su totalidad se escribe un bucle while que indique el fin de archivo (EOF). Para leer de un archivo de texto hasta el final, se puede utilizar el siguiente fragmento de código:

```
...
c=fgetc(f);//Se lee un caracter del archivo y se guarda en c
while (c!=EOF)
      {  /*si no es fin de archivo sigue leyendo*/
         c=fgetc(f) ;
      }
...
```

Cabe destacar que "f", es el nombre que le dimos al puntero a un FILE, que previamente fue abierto con *fopen()*.

Fin de archivo

Las funciones de biblioteca de E/S de archivos, generalmente empiezan con la letra f de file, tiene especificado que son de tipo entero de tal forma que si la operación falla devuelve EOF, también devuelve EOF para indicar que se ha leído el fin de archivo.

EOF (End Of File) es un parámetro útil para facilitar el cierre de bucles de extracción de datos desde archivos. En C, EOF es una constante de tipo entero (normalmente -1) qué es el retorno que envían distintas funciones de extracción de información desde archivos al llegar al final del mismo y no existir más datos.

Otra forma de hacer dicha comprobación es utilizar la función *feof().* Esta función es muy útil para ser utilizada en archivos binarios (donde la constante EOF no tiene el mismo significado) aunque se puede utilizar en cualquier tipo de archivo.

La función *feof()*, devuelve cero (true) cuando se lee el carácter de fin de archivo, en caso contrario devuelve un valor distinto de cero (false). El prototipo de esta función es el siguiente:

```
int feof(FILE *Identificador_archivo)
```

El siguiente fragmento de código muestra cómo se utiliza dicha función:

```
...
while (!feof(f))
        {
            c = fgetc(f);

        };
...
```

Escritura de caracteres

Para escribir carácter a carácter en un archivo de texto se invoca a alguna de estas dos funciones *putc()* y *fputc()*. Al igual que con las funciones de lectura, son idénticas y escriben un carácter asociado con el puntero a FILE. Devuelve el carácter escrito o bien EOF si no lo puede escribir.

El prototipo de ambas funciones es:

```
int putc(int Id_variable, FILE *Identificador_archivo)
```

```
int fputc(int Id_variable, FILE *Identificador_archivo)
```

Id_variable es el carácter a escribir en dicho archivo e Identificador_archivo, el puntero al archivo que se abre con *fopen()*

Para escribir en un archivo de texto, se puede utilizar el siguiente fragmento de código:

```
...
scanf("%c", &c);
while (c != '0')//Lee mientras el carácter sea distinto de 0
      {
         fputc(c, f);//Guarda c en el archivo
         printf("Ingrese el dato");
         fflush(stdin);
         scanf("%c", &c);
      };
...
```

Leer cadenas de caracteres

La función *fgets()* permite leer una cadena de caracteres del archivo; termina la lectura cuando lee el carácter de fin de línea (salto de línea) o hasta que se supere la longitud de dicha cadena. La función devuelve un puntero señalando al texto leído o un puntero nulo (NULL) si la operación provoca un error.

Su prototipo de dicha función es:

```
char*fgets(char *cadena,int longitud,FILE *Identificador_archivo);
```

El primer parámetro es la cadena en la que se desea depositar el resultado de la lectura. El segundo parámetro, un entero, es el máximo número de bytes que queremos leer en la cadena. Dicho límite permite evitar desbordamientos de la zona de memoria reservada para cadena cuando esta es más larga de lo previsto. El último parámetro es, finalmente, el archivo del que vamos a leer (previamente se ha abierto con *fopen()*).

El siguiente ejemplo muestra el uso de *fgets()*:

```
#include <stdio.h>
#define n 100
int main()
{
    FILE *archivo;
    char cadena[n];
    char *res;
    archivo = fopen("string.txt", "r");
    if (archivo != NULL)
    {
        res = fgets(cadena, n, archivo);
        while (res != NULL)
        {
            printf("%s", cadena);
            res = fgets(cadena, n, archivo);
        }
        fclose(archivo);
    }
    else
    {
        printf("Error en la apertura");
    }
    return 0;
}
```

Escribir cadenas de caracteres

La función *fputs()* escribe una cadena de caracteres, devuelve EOF si no ha podido escribir la cadena, un valor no negativo si la escritura es correcta. Escribe el texto en el archivo indicado, al final del texto colocará el carácter del salto de línea (al igual que hace la función puts).

El prototipo de dicha función es:

```
int fputs(char *cadena, FILE *Identificador_archivo);
```

El siguiente es un ejemplo de un algoritmo que lee cadenas y las escribe o guarda en un archivo:

```c
#include <stdio.h>
#include <string.h>
int main()
{   FILE *archivo;
    char cadena[100];
    char res;
    archivo = fopen("string.txt", "w");
    if (archivo != NULL)
       {
        printf("Va a leer una cadena? S/N ");
        fflush(stdin);
        scanf("%c", &res);
        while (res == 'S')
        {
          fflush(stdin);
          gets(cadena); //Lee una cadena de caracteres del teclado
          strcat(cadena, "\n");
          fputs(cadena, archivo); //Graba la cadena leída
          printf("Va a leer otra cadena? S/N ");
          fflush(stdin);
          scanf("%c", &res);
        }
        fclose(archivo);
    }
    else{
        printf("Error en la apertura");
    }
    return 0;
}
```

Además de las funciones básicas de E/S, el sistema de E/S con buffer incluye *fprintf()* y *fscanf()*. Estas funciones se comportan exactamente igual que las funciones *printf()* y *scanf()*, excepto en que operan sobre archivos.

Función fprintf

Se trata de la función equivalente a la función *printf*() sólo que esta permite la escritura en archivos de texto. El formato es el mismo que el de la función *printf()*, sólo que se añade un parámetro al principio que es el puntero al archivo en el que se desea escribir.

La ventaja de esta instrucción es que aporta una gran versatilidad a la hora de escribir en un archivo de texto.

El prototipo de la función *fprintf()* es el siguiente:

```
int fprintf(FILE *Identificador_archivo,"Modificadores",variables)
```

El ejemplo a continuacion muestra el uso de *fprintf()*:

```
#include <stdio.h>
int main(){
    char n; //Control de acceso
    int Legajo;
    int edad;
    FILE *f;
    f= fopen("prueba3.txt", "w");
    if (f != NULL)
    {
        printf("Desea ingresar un empleado? S/N ");
        scanf("%c", &n);
        while (n == 'S')
        {
            printf("Introduzca el Legajo del empleado: ");
            scanf("%d", &Legajo);
            printf("Introduzca la edad del empleado: ");
            scanf("%d", &edad);
            fprintf(f, "%d\t%d\n", Legajo, edad);
            printf("Desea ingresar un empleado? S/N ");
            fflush(stdin);
            scanf("%c", &n);
        }
        fclose(f);
    }
    else
        printf("Error en la apertura\n");
return 0;
}
```

Note que al escribir en el archivo con *fprintf()* los datos se guardan separados por una marca de tabulación (\t).

Función fscanf

Se trata de la equivalente al *scanf ()* de lectura de datos por teclado. Funciona igual sólo que requiere un primer parámetro que sirve para asociar la lectura a un puntero de archivo. El resto de los parámetros se manejan igual que en el caso de *scanf()*.

El prototipo es de la función *fscanf()* es:

```
int fscanf(FILE *Identificador_archivo,"Modificadores",&variables)
```

El siguiente ejemplo muestra el uso de *fscanf()*:

```
#include <stdio.h>
int main()
{
    char n;
    int Legajo;
    int edad;
    FILE *f;
    f= fopen("prueba3.txt", "r");
    if (f != NULL)
    {
        while (!feof(f))
        {
            fscanf(f, "%d\t%d\n", &Legajo, &edad);
            printf("registro: %d\t%d\n", Legajo, edad);
        }
        fclose(f);
    }
    else{
        printf("Error de apertura");
    }
    return 0;
}
```

Note que la lectura con fscanf() es igual que con un scanf(), se antepone el operador (&) a la variable que lo requiera.

Función fflush

Como ya se mencionó en apartados anteriores, esta función vacía el buffer, en este caso sobre el archivo indicado. Si no se pasa ningún puntero se vacían los búferes de todos los archivos abiertos. Se puede pasar también la corriente estándar de entrada stdin para vaciar el búfer de teclado (necesario si se leen caracteres), de otro modo algunas lecturas fallarían).

Esta función devuelve cero si todo ha ido bien y la constante EOF en caso de que ocurriera un problema al realizar la acción. La sintaxis de esta función es:

```
int fflush(FILE *Identificador_archivo);
```

Procesamiento de archivos binarios

Para abrir un archivo en modo binario hay que especificar la opción b en el modo. Una de las características de los archivos binarios es que optimizan la memoria ocupada por un archivo, sobre todo con campos numéricos.

Los modos para abrir un archivo binario tal y como se explicó más arriba son los siguientes:

"rb", "wb", "ab", o bien "r+b", "w+b", "a+b".

El siguiente fragmento de código muestra la forma de abrir un archivo binario para lectura:

```
fopen("Archivo.num","rb");
```

Lectura de un archivo binario

Los archivos binarios están especialmente indicados para guardar registros o estructuras en C. La biblioteca de C proporciona dos funciones especialmente útiles para el proceso de entrada y salida de archivos con buffer, son *fread()* y *fwrite()*.

Función fwrite

Se trata de la función que permite escribir en un archivo datos binarios del tamaño que sea.

El prototipo para la función *fwrite()* es:

```
int fwrite(void *ptr,int tam,int numdatos, FILE *Identificador_Archivo);
```

Descripción del prototipo:

- Ptr: Puntero a la posición de memoria que contiene el dato que se desea escribir.
- Tam: Tamaño de los datos que se desean escribir (suele ser calculado con sizeof).
- Numdatos: Indica los bloques de bytes que se escribirán en el archivo. Cada elemento tendrá el tamaño en bytes indicado y su posición será contigua a partir de la posición señalada por el argumento ptr.
- Identificador_Archivo: Puntero al archivo en el que se desean escribir los datos.

El siguiente ejemplo sencillo muestra cómo utilizar la función fwrite(). Crea un archivo binario, y carga 10 números consecutivos.

```
#include <stdio.h>
#define n 10
int main()
{
    FILE *f;
    f= fopen("numeros.dat", "wb");
    if (f != NULL)
    {
        for (int i = 1; i <= n; i++)
        {
            fwrite(&i, sizeof(int), 1, f);
        }
        fclose(f);
    }
    else
    {
        printf("Error en la apertura del archivo");
    }
return 0;
}
```

Escritura de un archivo binario

Función fread

Se trata de una función absolutamente equivalente a la anterior, sólo que en este caso la función lee del archivo.

El prototipo de la función *fread()* es:

```
int fread(void *ptr,int tam,int numdatos,FILE *Identificador_Archivo);
```

La descripción es la misma que para *fwrite()*.

A continuación el ejemplo muestra cómo utilizar la función *fread()*. Abre el archivo creado en el ejemplo anterior y lo muestra por pantalla:

```
#include <stdio.h>
#define n 10
int main()
{
    int i;
    FILE *f;
    f= fopen("numeros.dat", "rb");
    if (f != NULL)
    {
        fread(&i, sizeof(int), 1, f);
        while (!feof(Archivo))
        {
            printf("%d\n", i);
            fread(&i, sizeof(int), 1, f);
        }
        fclose(f);
    }
    else
    {
        printf("Error en la apertura del archivo");
    }
}
```

Uso de archivos de acceso directo

Hasta ahora todas las funciones de proceso de archivos vistas han trabajado con los mismos de manera secuencial, a modo de introducción veremos algunas funciones de utilidad para este tipo de acceso.

El acceso directo a los datos de un archivo se hace mediante su posición; es decir el lugar relativo que ocupan. Este tipo de acceso tiene la ventaja de que se pueden leer y escribir registros en cualquier orden y posición y son muy rápidos de acceder a la información que contienen.

En los archivos de acceso directo se entiende que hay un indicador de posición en los archivos que señala el dato que se desea leer o escribir. Las funciones *fread()* o *fwrite()* vistas anteriormente mueven el indicador de posición cada vez que se usan y el programa mantiene a través de un puntero la posición actual.

Función fseek

El acceso directo se consigue si se modifica el indicador de posición hacia la posición deseada. Eso lo realiza la función *fseek()* cuyo prototipo es:

```
int fseek(FILE * fichero, long desplazamiento, int origen);
```

Esta función coloca el cursor en la posición marcada por el origen desplazándose desde allí el número de desplazamiento indicado por el segundo parámetro (que puede ser negativo).

Es decir con la función *fseek()* se sitúa el puntero en una posición aleatoria, dependiendo del desplazamiento y el origen relativo que se pasan como argumentos.

El argumento origen puede tener tres valores representados por las siguientes macros (definidas en stdio.h):

- SEEK_SET: Cuenta desde el principio del archivo.
- SEEK_CUR: Cuenta desde la posición actual del puntero.
- SEEK_END: Cuenta desde el final del archivo.

La función *fseek()* devuelve un valor entero, distinto de cero si se comete un error en su ejecución; cero si no hay error.

El siguiente ejemplo muestra el uso de la función *fseek()*:

```
#include <stdio.h>
int main()
{
    int *aux;
    FILE *f;
```

```
    f= fopen("numeros.dat", "rb");
    if (f != NULL)
    {
        //Se ubica en la quinta posición
        fseek(f, sizeof(int) * 4, SEEK_SET);
        fread(&aux, sizeof(int), 1, f);
        printf("El valor en la posicion es: %d", aux);
        fclose(f);
    }
    else
    {
        printf("Error en la apertura");
    }
    return 0;
}
```

El ejemplo anterior muestra por pantalla el quinto elemento que se encuentre en el archivo, (sizeof(int)*4 devuelve el quinto elemento, ya que el primero está en la posición cero).

Hay que tener siempre presente que los desplazamientos sobre el fichero se indican en bytes. Si hemos almacenado enteros de tipo int en un fichero binario, deberemos tener la precaución de que todos nuestros fseek tengan desplazamientos múltiples de sizeof(int).

Función ftell

Se trata de una función que obtiene el valor actual del indicador de posición del archivo (la posición en la que se comenzaría a leer con una instrucción de lectura). En un archivo binario es el número de byte en el que está situado el cursor desde el comienzo del archivo.

El prototipo de dicha función es:

```
long ftell(FILE *Identificador_archivo);
```

El siguiente ejemplo muestra el uso de la función *ftell()* para obtener el número de registros de un archivo:

```
#include <stdio.h>
int main()
{
    FILE *f = fopen("numeros.dat", "rb");
```

```
    int nReg; /*Guarda el número de registros*/
    if (f != NULL)
    {
        fseek(f, 0, SEEK_END);
        nReg = ftell(f) / sizeof(int);
        printf("Nro. de registros en el archivo = %d", nReg);
    }
    else
    {
        printf("Error en la apertura del archivo");
    }
    return 0;
}
```

En el caso de que la función *ftell()* falle, da como resultado el valor -1.

Funciones fgetpos y fsetpos

Ambas funciones permiten utilizar marcadores para facilitar el trabajo en el archivo. Sus prototipos son:

```
int fgetpos(FILE *fichero, fpos_t *pos);
```

```
int fsetpos(FILE *fichero, fpos_t *pos);
```

La función *fgetpos()* obtiene el valor actual del indicador de posición de archivo y lo almacena en el objeto al que apunta pos . La función *fsetpos()* puede utilizar posteriormente la información almacenada en pos para restablecer el puntero del argumento a su posición en el momento en que se llamó a *fgetpos()* .

Dicho de otra forma la función *fgetspos()* almacena en el puntero pos, la posición actual del cursor del archivo (el indicador de posición), para ser utilizado más adelante por *fsetpos()* para obligar al programa a que se coloque en la posición marcada. En el caso de que todo vaya bien ambas funciones devuelven cero.
El tipo de datos *fpos_t* está declarado en la librería stdio.h y normalmente se corresponde a un número long. Es decir normalmente su declaración es: typedef long *fpos_t*; Aunque eso depende del compilador y sistema en el que trabajemos.

Ejemplo de uso de estas funciones:

```c
#include <stdio.h>
int main()
{
    FILE *f;
    fpos_t pos;
    f= fopen("file.txt", "w+");
    fputs("HOLA 1!\n",f);
    fgetpos(f, &pos);
    fputs("HOLA 2!\n", f);
    fsetpos(f, &pos);
    fputs("Esto sobreescribe HOLA 2! guardado anteriormente",f);
    fclose(archivo);
    return (0);
}
```