**Manejo de archivos**

C define la estructura de datos [FILE](http://c.conclase.net/librerias/?ansitip=FILE) en el fichero de cabecera [stdio.h](http://c.conclase.net/librerias/?ansilib=stdio) para el manejo de archivos. Nosotros siempre usaremos punteros a estas estructuras.

La definición de ésta estructura depende del compilador, pero en general mantienen un campo con la posición actual de lectura/escritura, un buffer para mejorar las prestaciones de acceso al fichero y algunos campos para uso interno.

Primeramente se debe declarar un puntero a FILE.

Por ejemplo FILE \* arch;

Este puntero estará asociado con el archivo físico que esté guardado en el disco.

**Apertura de archivos**

El archivo puede ser abierto para leer, para escribir o para leer y escribir.

* *FILE \* fopen(const char \* nom, const char \* modo)*

*nom*: Nombre del archivo físico guardado en el disco.

*modo*:

“r”: Abrir un archivo para leer. Si el archivo no existe o no se encuentra, se obtiene un error. El puntero esta al principio

“w”: Abrir un archivo para escribir. Si el archivo no existe, se crea; y si existe, su contenido se destruye para ser creado de nuevo. El puntero esta al principio

“a”: Abrir un archivo para añadir registros al final del mismo. Si el archivo no existe, lo crea. Lleva el puntero al final.

“r+”: Abrir un archivo para leer y escribir. El archivo debe existir. El puntero esta al principio

“w+”: Abrir un archivo para leer y escribir. Si el fichero no existe, se crea; y si existe, su contenido se destruye para ser creado de nuevo. El puntero esta al principio

“a+”: Abrir un archivo para leer y añadir. Si el fichero no existe se crea. El puntero esta al final

Ejemplo: arch=fopen(“Empleados.txt”,”w”);

Después de abrir un archivo el puntero de Lectura/escritura está al principio del archivo, excepto para el modo añadir que está al final.

La función fopen devuelve un puntero a una estructura de tipo FILE que define el buffer asociado con el archivo abierto. Un puntero nulo indica un error en la apertura.

Agregar t si el archivo se abre en formato de texto y b si se abre en formato binario. Si no se especifica "t" ni "b", se asume por defecto que es "t".

Ejemplo: rb, a+b ab+

Se coloca un if para el caso de que no lo haya podido abrir (puntero==NULL)

+ printf(no se pudo abrir el archivo)

+ exit(-1)

**Cerrar un archivo**

* *int fclose(FILE \*pf);*

Si la operación de cerrar un archivo se ejecuta satisfactoriamente, la función fclose devuelve un cero. Si ocurre un error entonces devuelve un EOF(una constante).

Ejemplo: fclose(arch)

**Entrada/Salida con formato (se utiliza para archivos de texto)**

Los datos pueden ser escritos con formato en un archivo y leídos con formato de un archivo utilizando las funciones fprintf, fscanf.

* *int fprintf(FILE \*pf, const char \*formato[,argumentos])*
* *int fscanf(FILE \*pf, const char \*format[,argumentos])*

Ejemplo: fprintf(arch,”%s;%d;%.2f\n”,nombre,dni,sueldo)

fscanf(arch,”%s,%d,%f”,nombre,&dni,&sueldo)

**Entrada/Salida utilizando variables de tipo estructura (registros)**

Los datos pueden ser escritos y leídos como registros o bloques con las funciones fwrite y fread, como un conjunto de datos de longitud fija.

* *size\_t write(const void \* buffer, size\_t n, size\_t c, FILE \*pf);*

La función fwrite permite escribir c elementos de longitud n bytes almacenados en el buffer especificado, en el archivo apuntado por pf. Esta función devuelve el número de elementos actualmente escritos. Si este número es menor que c, entonces es que ha ocurrido un error.

* *size\_t fread(void \* buffer, size\_t n, size\_t c, FILE \*pf);*

La función fread permite leer c elementos de longitud n bytes, del archivo apuntado por pf y los almacena en el buffer especificado. Esta función devuelve el número de elementos actualmente leídos, que puede ser menor que c si ocurre un error.

**Acceso aleatorio**

* *int fseek(FILE \*pf, long desp, int pos);*

**Ejemplo:** fseek(arch,0,SEEK\_SET) Ubica el puntero de lectura/escritura al comienzo del archivo.

**pos**:

SEEK\_SET principio del archivo

SEEK\_CUR posición actual del puntero de lectura/escritura

SEEK\_END Final del archivo

La función fseek mueve el puntero de lectura/escritura asociado con el archivo apuntado por pf, a una nueva localización desplazada desp bytes de la posición dada por el argumento pos. El desplazamiento desp puede ser positivo o negativo.

La función fseek devuelve un 0 si no se ha producido un error y un distinto de 0 en caso contrario.

* *long ftell (FILE\* pf);*

La función ftell da como resultado la posición actual del puntero de lectura/escritura dentro del fichero apuntado por pf, o el valor -1L si ocurre un error. Esta posición es relativa al principio del archivo.

* *void clearerr(FILE \*pf);*

Desactiva el indicador de error y el indicador de fin de fichero para un determinado archivo, poniéndolos a valor 0.

* *int feof(FILE \*pf);*

Cuando en una operación de lectura sobre un archivo se intenta leer más allá de la marca de fin de fichero, automáticamente el sistema activa el indicador de fin de fichero asociado con él. Para saber el estado de este indicador para un determinado archivo, hay que invocar a la función feof. La marca de fin de fichero es añadido automáticamente por el sistema cuando crea el archivo.

La función devuelve un valor distinto de 0 cuando se intenta leer un elemento del archivo y nos encontramos con un eof. En caso contrario devuelve un 0.

* *void rewind(FILE \*pf);*

La función rewind pone el puntero de lectura/escritura del fichero apuntado por pf al comienzo del mismo.

Rewind desactiva los indicadores de error y de fin de fichero (fseek no)

**Ejercicios de ejemplo:**

Creación de un archivo secuencial: “creararchsec.cpp”

Lectura de un archivo secuencial: “leerarchsec.cpp”

Uso de archivo secuencial: “investigarctas.cpp”

Creación de archivo aleatorio: “creararchaleatorio.cpp”

Lectura de archivo aleatorio: “leerarchaleatorio.cpp”

Aplicación de manejo de archivos: “modeloABM.cpp”