Manejo de Archivos en C.

Los datos que hemos tratado hasta el momento han residido en la memoria principal. Sin embargo, las grandes cantidades de datos se almacenan normalmente en un dispositivo de memoria secundaria. Estas colecciones de datos se conocen como archivos (antiguamente ficheros).

Un archivo es un conjunto de datos estructurados en una colección de entidades elementales o básicas denominadas registros que son de igual tipo y constan a su vez de diferentes entidades de nivel más bajos denominadas campos.

Hay dos tipos de archivos, archivos de texto y archivos binarios. Un **archivo de texto** es una secuencia de caracteres organizadas en líneas terminadas por un carácter de nueva línea. En estos archivos se pueden almacenar canciones, fuentes de programas, base de datos simples, etc. Los archivos de texto se caracterizan por ser planos, es decir, todas las letras tienen el mismo formato y no hay palabras subrayadas, en negrita, o letras de distinto tamaño o ancho.

Un **archivo binario** es una secuencia de bytes que tienen una correspondencia uno a uno con un dispositivo externo. Así que no tendrá lugar ninguna traducción de caracteres. Además, el número de bytes escritos (leídos) será el mismo que los encontrados en el dispositivo externo. Ejemplos de estos archivos son Fotografías, imágenes, texto con formatos, archivos ejecutables (aplicaciones), etc.

En c, un archivo es un concepto lógico que puede aplicarse a muchas cosas desde archivos de disco hasta terminales o una impresora. Se asocia una secuencia con un archivo especifico realizando una operación de apertura.

Una vez que el archivo está abierto, la información puede ser intercambiada entre este y el programa.

Se puede conseguir la entrada y la salida de datos a un archivo a través del uso de la biblioteca de funciones; C no tiene palabras claves que realicen las operaciones de E/S. La siguiente tabla da un breve resumen de las funciones que se pueden utilizar. Se debe incluir la librería STDIO.H. Observe que la mayoría de las funciones comienzan con la letra “F”, esto es un vestigio del estándar C de Unix.

Nombre	**Función**
fopen()	Abre un archivo.
fclose()	Cierra un archivo.
fgets()	Lee una cadena de un archivo.
fputs()	Escribe una cadena en un archivo
fseek()	Busca un byte especifico de un archivo.
fprintf()	Escribe una salida con formato en el archivo.
fscanf()	Lee una entrada con formato desde el archivo.
feof()	Devuelve cierto si se llega al final del archivo.
ferror()	Devuelve cierto si se produce un error.

rewind()	Coloca el localizador de posición del archivo al principio del mismo.
remove()	Borra un archivo.
fflush()	Vacía un archivo.

El puntero a un archivo.

El puntero a un archivo es el hilo común que unifica el sistema de E/S con buffer. Un puntero a un archivo es un puntero a una información que define varias cosas sobre él, incluyendo el nombre, el estado y la posición actual del archivo. En esencia identifica un archivo especifico y utiliza la secuencia asociada para dirigir el funcionamiento de las funciones de E/S con buffer. Un puntero a un archivo es una variable de tipo puntero al tipo FILE que se define en STDIO.H. Un programa necesita utilizar punteros a archivos para leer o escribir en los mismos. Para obtener una variable de este tipo se utiliza una secuencia como esta:

FILE *F;

Apertura de un archivo. La función **fopen()** abre una secuencia para que pueda ser utilizada y la asocia a un archivo. Su prototipo es:

FILE *fopen(const char nombre_archivo, cost charmodo);

Donde nombre_archivo es un puntero a una cadena de caracteres que representan un nombre valido del archivo y puede incluir una especificación del directorio. La cadena a la que apunta modo determina como se abre el archivo. La siguiente tabla muestra los valores permitidos para modo.

Modo	Significado
r	Abre un archivo de texto para lectura.
w	Crea un archivo de texto para escritura.
a	Abre un archivo de texto para añadir.
rb	Abre un archivo binario para lectura.
wb	Crea un archivo binario para escritura.
ab	Abre un archivo binario para añadir.
r+	Abre un archivo de texto para lectura / escritura.
w+	Crea un archivo de texto para lectura / escritura.
a+	Añade o crea un archivo de texto para lectura / escritura.
r+b	Abre un archivo binario para lectura / escritura.
w+b	Crea un archivo binario para lectura / escritura.
a+b	Añade o crea un archivo binario para lectura / escritura.

La función **fopen()** devuelve un puntero a archivo. Un programa nunca debe alterar el valor de ese puntero. Si se produce un error cuando se esta intentando abrir un archivo, **fopen()** devuelve un puntero nulo.

Se puede abrir un archivo bien en modo texto o binario. En la mayoría de las implementaciones, en modo texto, la secuencias de retorno de carro / salto de línea se convierten a caracteres de salto de línea en lectura. En la escritura, ocurre lo contrario: los caracteres de salto de línea se convierten

en salto de línea. Estas conversiones no ocurren en archivos binarios.

La macro NULL está definida en STDIO.H. Este método detecto cualquier error al abrir un archivo: como por ejemplo disco lleno o protegido contra escritura antes de comenzar a escribir en él.

Si se usa **fopen()** para abrir un archivo para escritura, entonces cualquier archivo existente con el mismo nombre se borrará y se crea uno nuevo. Si no existe un archivo con el mismo nombre, entonces se creará. Si se quiere añadir al final del archivo entonces debe usar el modo a. Si se usa a y no existe el archivo, se devolverá un error. La apertura de un archivo para las operaciones de lectura requiere que exista el archivo. Si no existe, **fopen()** devolverá un error. Finalmente, sus se abre un archivo para las operaciones de leer / escribir, la computadora no lo borrará si existe; sin embargo, si no existe, la computadora lo creará.

Cierre de un archivo. La función **fclose()** cierra una secuencia que fue abierta mediante una llamada a **fopen()**. Escribe toda la

información que todavía se encuentre en el buffer en el disco y realiza un cierre formal del archivo a nivel del sistema operativo. Un error en el cierre de una secuencia puede generar todo tipo de problemas, incluyendo la pérdida de datos, destrucción de archivos y posibles errores intermitentes en el programa. El prototipo de esta función es:

int fclose(FILE *F);

Donde F es el puntero al archivo devuelto por la llamada a **fopen()**. Si se devuelve un valor cero significa que la operación de cierre ha tenido éxito. Generalmente, esta función solo falla cuando un disco se ha retirado antes de tiempo o cuando no queda espacio libre en el mismo.

Para introducir u obtener datos de un archivo tenemos las siguientes cuatro funciones:

fprintf() y fscanf() Estas funciones se comportan exactamente como **prinft()** y **scanf()** discutidas anteriormente, excepto que

operan sobre archivo. Sus prototipos son:

int fprintf(FILE *F, const char *cadena_de_control, ...); int fscanf(FILE *F, const char *cadena_de_control, ...);

Donde F es un puntero al archivo devuelto por una llamada a fopen(). fprintf() y fscanf() dirigen sus operaciones de E/S al archivo al que apunta F.

Las funciones **fgets()** y **fputs()** pueden leer y escribir cadenas a o desde los archivos. Los prototipos de estas funciones son:

char *fputs(char *str, FILE *F); char *fgets(char *str, int long, FILE *F);

La función **puts()** escribe la cadena a un archivo especifico. La función **fgets()** lee una cadena desde el archivo especificado hasta que lee un carácter de nueva línea o longitud-1 caracteres.

Si se produce un EOF (End of File) la función gets retorna un NULL.

Funcion feof()

Cuando se abre un archivo para entrada binaria, se puede leer un valor entero igual de la marca EOF. Esto podría hacer que la rutina de lectura indicase una condición de fin de archivo aún cuando el fin físico del mismo no se haya alcanzado. Para resolver este problema, C incluye la función **feof()**, que determina cuando se ha alcanzado el fin del archivo leyendo

datos binarios. La función tiene el siguiente prototipo:

int feof(FILE *F);

Su prototipo se encuentra en STDIO.H. Devuelve cierto si se ha alcanzado el final del archivo, en cualquier otro caso, 0. Por supuesto, se puede aplicar este método a archivos de texto también.

Ahora bien para el ejemplo anterior usted incluirá los datos de la forma: Nombre del alumno1 nota

Nombre del alumno2 nota .....

Algunas veces usted necesitara manipular por separado el nombre del alumno y su nota, para esto es necesario separarlo en campos. Se puede realizar introduciendo caracteres delimitadores entre campo y campo, por ejemplo:

Esto generara un archivo de tipo:

fprinft(C,"%s;%d \n",nombre,cal);

Nombre del alumno1;nota Nombre del alumno2;nota .....

La función **rewind()** inicializa el indicador de posición, al principio del archivo, indicado por su argumento. Su prototipo es:

void rewind (FILE *F);

Donde F es un puntero a un archivo válido. Esta función se encuentra en STDIO.H

La función **ferror()** determina si se ha producido en error en una operación

sobre un archivo. Su prototipo es:

int ferror(FILE *F);

Donde F es un puntero a un archivo válido. Devuelve cierto si se ha producido un error durante la ultima operación sobre el archivo. En caso contrario, devuelve falso. Debido a que cada operación sobre el archivo actualiza la condición de error, se debe llamar a **ferror()** inmediatamente después de la operación de este tipo; si no se ase así, el error puede perderse. Esta función se encuentra en STDIO.H

La función **remove()** borra el archivo especificado. Su prototipo es el siguiente: **int remove(char *nombre_archivo);**

Devuelve cero si tiene éxito. Si no un valor distinto de cero.

La función **fflush()** escribe todos los datos almacenados en el buffer sobre el archivo asociado con un apuntador. Su prototipo es:

i**nt fflush(FILE *F);**Si se llama esta función con un puntero nulo se vacian los buffers de todos los archivos abiertos. Esta función devuelve cero si tiene éxito, en otro caso, devuelve EOF.