manipulacion de archivos con C

crear un archivo:

la funcion **fopen** recibe 2 parametros:

1. el nombre del archivo en disco que quiero abrir. en linux no es tan necesario especificar la extension, y va entre comillas dobles.
2. este parametro de uno o dos caracteres, especifica el tipo de operaciones que quiero hacer con el archivo.
   algunas de las opciones para el segundo parametro son:
   r) solo lectura: cursor al principio
   r+) lectura y escritura: cursor al principio
   w)crea un nuevo archivo o resetea a vacío el existente: cursor al principio
   w+)igual a w pero permite leer
   a) agrega escritura al final del archivo o crea uno nuevo si no existe, cursor al final.
   a+)agrega escritura al final y permite leer.en ubuntu el cursor va al principio, en mac, android y BSD al final
   rb) leer binario
   wb) escribir binario
   fopen devuelve un descriptor al archivo,

```
FILE *myFile = fopen("datos.bin", "rb");
if (myFile == NULL)
{
printf("\n================\ncannot open file\n================\n");
return 1;
}
printf("\n=============\nsuccess fopen\n=============\n");

/* aqui usaremos fread y fwrite, manejaremos los datos obtenido y demas.
luego es necesario cerrar el archivo que hemos abierto */

printf("\n=============\nclosing file\n=============\n");

int closeStatus = fclose(myFile);

closeStatus != 0 ? printf("error closing file (%i)\n=============\n",
closeStatus) : printf("success closing file (%i)\n=============\n",
closeStatus);
```

para cerrar el archivo usamos la funcion **fclose** que recibe simplemente el descriptor del archivo abierto e intenta cerrarlo, si devuelve cero, se cerro correctamente.

es necesario cerrarlo para que otros programas puedan acceder al archivo

leer el contenido del archivo

para comezar usaremos **fgetc**, que recibe el descriptor del archivo abierto, y devuelve un entero que representa el ASCII del caracter que esta leyendo el *cursor*. el cual por ejemplo podriamos castear a char para imprimirlo por pantalla:

```
int charLeido = fgetc(myFile);
printf("he leido el caracter %c\n",(char)charLeido);
```

para evitar errores como por ejemplo intentar leer estando al final del archivo,
podemos aprovechar el caracter EOF *end of file* y la funcion **feof**, la cual recibe el descriptor del archivo abierto y devuelve false o cero si el caracter actual es *distinto de EOF* y en caso de que sea EOF, devuelve true, o no-cero.
para el mismo objetivo podriamos usar un simple

```
if(charLeido!=EOF){}
```

manipulacion del cursor:

hay dos funciones principales que nos permiten saber donde esta el cursor y moverlo a donde querramos: la primera es **ftell()** que recibe el descriptor al archivo abierto y devuelve un long, este long es un numero entero que representa la posicion del cursor en el archivo. qu ese puede imprimir con el placeholder %ld.
devuelve -1 si hay algun error.

la otra funcion que usaremos es **fseek** que recibe el descriptor,un numero entero positivo o negativo que indica la cantidad de pasos y la direccion que quiero que el cursor se mueva, y el tercer parametro indica desde donde se va a mover el cursor:

-SEEK_SET:desde el inicio del documento
-SEEK_END:desde el final del documento
-SEEK_CUR:desde la posicion actual del cursor.

de modo que por ejemplo, para mover hacia la izquierda cinco veces el cursor desde donde se encuentre en el momento, usaríamos:

```
fseek(myFile,-5,SEEK_CUR);
```

CALCULAR EL TAMAÑO TOTAL DEL DOCUMENTO:
puedo abrir el archivo en modo lectura,

llamar a fseek y mover mi cursor al final del documento,
con ftell puedo saber el largo total del documento en bytes, ya que la cantidad total de caracteres es la cantidad de bytes,
Al final podria usar de nuevo fseek para llevar mi cursor al inicio del documento, o usar la funcion **rewind(myFile)** que hace exactamente eso... o puedo continuar con cualquier otra tarea que quiera.

otra opcion para leer el archivo, es la funcion **fgets()**

```
char buffer[50];
fgets(buffer,50,myFile);
```

si fgetc permite guardar en un buffer el valor de un caracter, fgets permite hacerlo con el contenido de un string.
Recibe 3 parametros:

1. un puntero a el buffer en el cual quiero guardar cada string, lo mas lógico seria definir el buffer como un string (char buffer[50] por ejemplo) y al ser un array ya es un puntero, de modo que al pasarlo como argumento seria solo el nombre y ya.
2. la cantidad maxima de caracteres que quiero que lea o que guarde en el buffer, aqui conviene poner el la misma cantidad con la que se definio el buffer (50, en nuestro ejemplo).
3. el descriptor del archivo.

si fgets logra su objetivo, devuelve un puntero que apunta ala direccion de memoria del buffer (es decir devuelve el miismo buffer) y si tiene algun error devuelve NULL.
a demás, fgets detiene su lectura cuando:

-llega al numero de caracteres definido en el param 2
-llega a un salto de linea
-llega a un EOF

cabe anotar, que si definimos 50 caracteres, lee un maximo de 49, ya que incluye el salto de linea al final.

escribir en el archivo

fputc recibe como primer parametro entre comillas simples, un caracter, y como segundo parametro, el descriptor del archivo con el que etsamos trabajando.

```
fputc('h',myFile);
```

dependiendo del tipo de apertura del archivo, el cursor estará en alguna posicion o habremos usado fseek para ubicarlo a nuestro antojo. en ese lugar insertará el caracter que se le indique, y si se itera usando fputs, los insertará uno despues del anterior en un orden natural.

si sale bien devuelve un caracter numerico que representa el ascii del caracter escrito,
si No sale bien, devuelve EOF.

fputs trabaja del mismo modo, pero recibiendo un string completo entre comillas dobles.

```
fputs("hola, ¿que tal?",myFile);
```

si sale bien, devuelve un valor no negativo (si, lo que sea, puede ser cero, espacio o lo que sea no negativo).
si sale mal devuelve EOF.

escribir en el archivo volcando un buffer a nuestro archivo

cuando en lugar de un caracter, una palabra o un texto, quiero volcar una estructura mas compleja a un archivo, conviene usar la funcion **fwrte**, que recibe cuatro parametros:

1. Un puntero al buffer que queremos volcar a nuestro archivo.
2. el tamaño de este buffer, se suele usar sizeof del tipo de dato que quiero guardar.
3. la cantidad de veces que quiero guardar ese tamaño (se usa si quiero guardar varias instancias de esa estructura).
4. el descriptor al archivo.

devuelve un entero, que debe ser igual al especificado en el param 3, de ser menor, es un error, ya que no ha logrado la tarea completa.

```
//definimos un tipo de dato personalizado para el ejemplo:
struct medicion {
  unsigned short anio;
  unsigned char mes;
  unsigned char dia;
  float temperatura;
  unsigned char uv;
  unsigned char viento;
};
// ahora, dentro de mi main, instancio esta struct
// con la medicion por ejemplo del lunes.
int main(){
struct medicion lunes={
    .anio 2023, .mes=11, .dia=28, .temperatura= 23.6, .uv=6, .viento=22
  };

//para luego volcarla a mi archivo con fwrite
// en este ejemplo crearemos un archivo BINARIO para guardarlo

FILE *myFile = fopen("mediciones.bin", "wb");
//aqui tratamos los posibles errores de apertura
   int writeStatus = fwrite(&lunes, sizeof(struct medicion),3,myFile);
```

```
   (writeStatus != 3) ? perror("error en el fwrite") : printf("fwrite()
exitoso");
   fclose(myfile);
  return 0;
}
```

leer un archivo binario con contenido estructurado

para este fin cambiaremos el uso de fgetc o fgets por **fread**,
que recibe los mismos 4 parametros que fwrite, pero la direccion va en sentido opuesto

1. el puntero al buffer donde quiero guardar el contenido
2. cantidad de bytes que quiero guardar, usando sizeof de mi estructura.
3. el numero de veces que quiero guardar este dato.
4. el descriptor del archivo

```
void print_medicion(struct medicion *lunes){
  printf("fecha: %d/%d/%d\n%f grados, %d UV,viento de %d km/h\n",lunes-
>dia,lunes->mes,lunes->anio,lunes->temperatura,lunes->uv, lunes->viento);
}
```

y del mismo modo que fwrite, devuelve un numero que debe coincidir con la cantidad de veces que le hemos pedido que complete la tarea.
una vez hecho esto podriamos usar nuestra funcion print_medicion y debemos ver en consola correctamente los datos de nuestra estructura.

un ejemplo de cuando tendriamos mas de una estructura para el fwrite o el fread seria:

```
struct medicion semana1[]={

{
  .anio 2023, .mes=11, .dia=21, .temperatura= 23.6, .uv=5, .viento=22
},{
  .anio 2023, .mes=11, .dia=22, .temperatura= 25.6, .uv=6, .viento=23
},{
  .anio 2023, .mes=11, .dia=23, .temperatura= 23.6, .uv=8, .viento=22
},{
  .anio 2023, .mes=11, .dia=24, .temperatura= 23.2, .uv=6, .viento=25
},
{
  .anio 2023, .mes=11, .dia=25, .temperatura= 23.6, .uv=9, .viento=22
},{
  .anio 2023, .mes=11, .dia=26, .temperatura= 24.3, .uv=6, .viento=20
},{
```

```
    .anio 2023, .mes=11, .dia=27, .temperatura= 23.5, .uv=65, .viento=22
  }
  }
```

en este caso se aplicaría la misma formula pero con el tercer parametro en 7