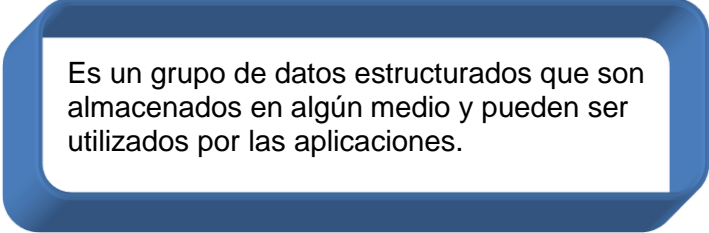


## **MANEJO DE ARCHIVOS EN C.**

Un archivo puede definirse en forma lógica diciendo que:



Es un grupo de datos estructurados que son almacenados en algún medio y pueden ser utilizados por las aplicaciones.

Las aplicaciones conocidas como Word, PowerPoint, Excel, etc. generan sus propios archivos a los cuales le anexan una extensión que los caracteriza. Por ejemplo:

.docx	Archivos de Word.
.xlsx	Archivos de Excel.
.pptx	Presentación Power.
.txt	Cuaderno de notas.
.psd	Photoshop.
.mp3	Reproductor de Windows / AIMP / Winamp / etc.

Estos archivos deben ser utilizados exclusivamente por la aplicación que los generó. Sin embargo la mayoría de las aplicaciones pueden manejar archivos de imágenes:

.jpg  
.bmp  
.tif  
etc.

Un documento de Word puede insertar una imagen .jpg.  
Una planilla en Excel puede importar un archivo de texto con formato.  
etc.

Desde un punto de vista más básico un archivo es:

Un flujo o secuencia de bytes hacia un dispositivo de almacenamiento, en donde el concepto de dato desaparece.

Esto sería mirando desde el hardware: el SO recibe un pedido de guardar o leer una cierta cantidad de bytes, los cuales carecen de todo significado para él. El encargado de verter esos bytes en el molde adecuado a fin de que tenga algún sentido para el usuario, es la Aplicación que lo solicitó.

Ocurre algo similar a cuando generábamos una reserva de memoria (una cierta cantidad de bytes) y almacenábamos algo allí. Si accedíamos a esa reserva con un puntero "tipificado" como una estructura, los bytes adquirirían automáticamente un sentido perfectamente lógico al ubicarse en cada miembro de la estructura.

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, archivos fuentes de programas, base de datos, 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.

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 específico 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.

El estándar de C contiene funciones para la edición de archivos, estas están definidas en la librería "**stdio.h**" y por lo general empiezan con la letra f, haciendo referencia a FILE. Adicionalmente se agrega un tipo **FILE**, el cual se usará como puntero a la información del archivo.

Las funciones más comunes para trabajar con archivos son:

Nombre	Función
fopen()	Abre un archivo
fclose()	Cierra un archivo
fgets()	Lee una cadena de caracteres de un archivo
fputs()	Escribe una cadena de caracteres en un archivo
putc()	Escribe un carácter en un archivo
getc()	Lee un carácter desde un archivo
fseek()	Salta al byte especificado en un archivo
fprintf()	Escribe en un archivo
fscanf()	Lee de un archivo
feof()	Devuelve verdadero si se alcanza la marca EOF (fin del archivo)
ferror()	Devuelve verdadero si a ocurrido un error
rewind()	Pone el localizador de posición de archivo al comienzo
fflush()	Vacía un archivo
remove()	Borra un archivo

La secuencia que usaremos para realizar operaciones será la siguiente:

1. Crear un puntero de tipo **FILE \***.
2. Abrir el archivo utilizando la función **fopen()** y asignándole el resultado de la llamada a nuestro apuntador.
3. Hacer las diversas operaciones (lectura, escritura, etc).
4. Cerrar el archivo utilizando la función **fclose()**.

## 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 esencial identificar un archivo específico y utilizar 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 a la estructura **FILE** que se define en "stdio.h" para el manejo de archivos. Un programa necesita utilizar punteros a archivos para leer o escribir en los mismos.

La definición de la estructura **FILE** 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 archivo y algunos campos para uso interno.

Para obtener una variable de este tipo se utiliza una secuencia como esta: **FILE \*F;**

En realidad, una variable de tipo **FILE \*** representa un flujo de datos que se asocia con un dispositivo físico de entrada/salida (el archivo "real" estará almacenado en disco).

Existen flujos de datos estándar predefinidos asociados a otros dispositivos de entrada/salida. Algunos de ellos son:

- **stdin**: representa la entrada estándar del sistema (teclado).
- **stdout**: representa la salida estándar del sistema (pantalla).
- **stderr**: representa la salida de error estándar (pantalla).

## APERTURA DE UN ARCHIVO:

Para poder operar con un archivo, exista previamente o no, es necesario "abrirlo" mediante la función **fopen()**.

El prototipo de dicha función es: **FILE \*fopen (const char \*filename, const char \*mode);**

Respecto a este prototipo:

- **fopen** devuelve un valor de tipo **FILE \*** (o sea, un puntero a **FILE**). Por supuesto, se supone que el valor devuelto será asignado a una variable de tipo **FILE \***, que se usará en otras funciones que manipulen dicho archivo.
- Si la apertura del archivo falla, **fopen** devuelve un puntero nulo.
- El argumento **filename** (una cadena de caracteres) es el nombre "real" del archivo que va a abrirse mediante **fopen** (es decir, es el nombre con el que el archivo

aparece en el disco).

- El argumento **mode** (una cadena de caracteres): indica qué tipo de operaciones se realizarán sobre el archivo abierto y el tipo de datos que puede contener, de texto o binarios:
- Los posibles valores de mode son:

Modo	Significado
r	El archivo se abre para lectura. Si el archivo no existe, se devuelve un puntero nulo.
w	Se crea el archivo para escritura. Si ya existe un archivo con ese nombre, el archivo antiguo será eliminado
a	Si ya existe un archivo con ese nombre, se abre para escritura (al final del archivo). Si no existe, se crea.
r+	Si el archivo existe, se abre para lectura y escritura (al principio del archivo).
w+	Se crea el archivo para lectura y escritura. Si ya existe un archivo con ese nombre, el archivo antiguo será eliminado.
a+	Si el archivo existe, se abre para lectura y escritura (al final del archivo). Si el archivo no existe, se crea.

En estos modos no se ha establecido el tipo de archivo, para ello se utilizará t para especificar un archivo de texto o b para binario.

- **t: tipo texto**, si no se especifica "t" ni "b", se asume por defecto que es "t".
- **b: tipo binario**.

Es decir: "rt", "wt", "at", "r+t", "w+t", "a+t" o bien "**rb**", "**wb**", "**ab**", "**r+b**", "**w+b**", "**a+b**"

**Nota:** Sea cual sea el valor elegido para mode, debe aparecer entre dobles comillas en la llamada a fopen.

### **CIERRE DE UN ARCHIVO**

Es importante cerrar los archivos abiertos antes de abandonar la aplicación. Esta función sirve para eso. Cerrar un archivo almacena los datos que aún están en el buffer de

memoria, y actualiza algunos datos de la cabecera del archivo que mantiene el sistema operativo. Además permite que otros programas puedan abrir el archivo para su uso. Muy a menudo, los archivos no pueden ser compartidos por varios programas.

Un valor de retorno cero indica que el archivo ha sido correctamente cerrado, si ha habido algún error, el valor de retorno es la constante **EOF**. El parámetro es un puntero a la estructura FILE del archivo que queremos cerrar.

Para cerrar un archivo, se usa la función **fclose**. Su prototipo es: **int fclose(FILE \*fp);**

### **Ejemplo #1: Programa en C para abrir un archivo.**

```
#include <stdio.h>
#include <stdlib.h>

void main() {

FILE *fp;

fp=fopen("archivo.txt","r");

if(fp==NULL){
printf("Error al abrir el archivo para leer");
exit(1);
}

fclose(fp);

}
```

## **Archivos de texto.**

En el caso particular de los archivos de texto, que es lo que nos interesa en este momento, nos centraremos exclusivamente en los textos puros, generados por ejemplo por el Cuaderno de Notas o los Editores de algún lenguaje de programación.

Las características fundamentales de estos archivos son:

- Sólo pueden almacenar caracteres.
- Están formados por registros que pueden tener cualquier longitud.
- Cada registro finaliza en uno o dos caracteres de control: **CR (Retorno de Carro)** y **LF (Salto de Línea)**.
- No se pueden acceder aleatoriamente.

Las formas de leer un archivo son varias:

- Se puede leer línea a línea sin formato.
- Línea a línea con formato.
- Caracter a caracter.

### IMPORTANTE:

Hasta este momento hemos estado acostumbrados a que una cadena de caracteres siempre finaliza con '\0'. Pues en un archivo de texto esta situación no se da, ya que hemos dicho que finalizan en el par CR+LF.

## Lectura y escritura sin formato de un archivo de texto.

Analicemos el primer caso, el más simple: leer línea a línea sin formato. Para ello utilizaremos primero la función `fgets( )` - obtener una cadena desde un archivo. Su prototipo es:

**`char *fgets(char *Destino,total_a_leer,FILE *Fuente);`**

Va leyendo caracteres desde el archivo hasta que ocurre una de dos situaciones:

- Cuando alcanza (total\_a\_leer - 1) caracteres.
- Cuando se topa con un carácter LF (ASCII=10),

`fgets( )` **saltea el CR**, retiene el carácter **LF** al final de **Destino**, y agrega un byte **null** a fin de que la cadena finalice de la forma acostumbrada. En definitiva, la cadena tiene ahora un carácter de más. La cadena **Destino** quedaría como:

T	E	X	T	O	L	E	I	D	O	L	F	\0
---	---	---	---	---	---	---	---	---	---	---	---	----

Esto quiere decir que, si deseamos obtener una cadena como la conocíamos, debemos hacer:

**`Destino[strlen(Destino)-1] = '\0'`**

En caso de éxito, **`fgets( )`** un puntero a **Destino**.

En caso de fallo o final de archivo, regresa una dirección **NULL**.

Para escribir en un archivo se utiliza la función **`fputs( )`** cuyo formato completo sería:

**`int fputs(char *Cadena,FILE *Destino);`**

Copia la cadena finalizada en null hacia el archivo **Destino**. Agrega un carácter **CR** y el **LF** que conservó **`fgets( )`**, y el carácter de terminación null no es copiado.

En caso de éxito retorna el número de caracteres copiados.

En caso de error regresa **EOF**.

Lo interesante de esta función es que, al realizar el proceso de escritura agrega nuevamente el caracter **CR** que **fgets( )** había excluido, quedando nuevamente la cadena en disco con el par **CR+LF**.

Seguramente ambas funciones han sido pensadas para trabajar juntas.

Para ilustrar lo dicho, se muestra el siguiente programa;

**Ejemplo #2: Programa en C que copia el contenido de un archivo en otro.**

```
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// -----
void main()
{
    FILE *Fuente;
    FILE *Destino;
    char NombFuente[128] = "TextoViernes.txt";
    char NombDestino[128] = "CopiaTextoViernes.txt";
    char Linea[128];
    int i,largo;

    clrscr();

    if((Fuente=fopen(NombFuente,"rt"))==NULL){
        cprintf("NO PUDO ABRIR ARCHIVO FUENTE");
        getch(); return;
    }

    if((Destino=fopen(NombDestino,"wt"))==NULL){
        cprintf("NO PUDO CREAR ARCHIVO DESTINO");
        fclose(Fuente); getch(); return;
    }

    while(fgets(Linea,127,Fuente)!=NULL){
        largo=strlen(Linea);
        for(i=0;i<largo;i++) {
            if((Linea[i]==13) || (Linea[i]==10)){
                textcolor(LIGHTRED);
                cprintf("%d",Linea[i]);
            }
            else{
                textcolor(WHITE);
                cprintf("%c",Linea[i]);
            }
        }
    }
```

```
    }  
    getch(); cprintf("\r\n");  
    fputs(Linea, Destino);  
}  
fclose(Fuente); fclose(Destino);  
textcolor(LIGHTGREEN);  
cprintf("COPIA DEL ARCHIVO FINALIZADA ....");  
  
getch();  
}
```

Al correr este programa, se nota que el caracter **CR** (ASCII=13) no aparece en ninguna línea extraída del archivo.

Concluyendo, la función **fgets( )** sólo permite leer caracteres desde un archivo de texto, ya sea líneas completas (hasta el para CR+LF), o bien una cierta cantidad de caracteres por ciclo de lectura.

## Lectura de un archivo de texto con formato.

La función **fscanf ( )** permite la lectura con formatos, aunque su sintaxis es un poco complicada. Vamos a volver al caso de leer línea a línea un archivo de texto.

El prototipo de esta función es la siguiente;

**int fscanf (FILE \*pf, const \*Formatos, DireccDeVariables);**

El formato de dichas funciones se puede complicar bastante, por ejemplo, considere la siguiente instrucción:

```
fscanf(pArch, "%2d%5c%4f", &entero, cadena, &real);
```

La misma acepta solamente un entero de 2 dígitos, una cadena de 5 caracteres y un número real que ocupa como máximo cuatro espacios (por ejemplo 2,97; 12,5).

Ahora intente imaginar qué hace la siguiente instrucción:

```
fscanf(fp1, "%*[\t\n] %[A-Za-z] %*[^A-Za-z][^\"], s1, s2);
```

- `%*[\t\n]` → Almacena los espacios en blancos, como está el símbolo `*` indica a la fn que no se guarda en una variable dichos datos
- `%[A-Za-z]` → Almacena todos los caracteres que sean letras entre “A y Z” y entre “a y z”
- `%*[^A-Za-z]` → Almacena todos los caracteres distintos a letras entre “A y Z” y entre “a y z”, como está el símbolo `*` indica a la fn que no se guarda en una variable dichos datos
- `%[^\"]` → almacena todos los caracteres hasta encontrar una “ (no la incluye a la misma)



Esta función suele tener el inconveniente de que cuando busca por una cadena, considera que la misma ha finalizado cuando se topa con el primer blanco. Sin embargo este hecho puede subsanarse de la siguiente manera:

```
FILE *Fuente;  
char Linea[128];  
char Resto[16];  
  
fscanf(Fuente,"%[^\\n ] %*\\n",Linea);
```

Este programa lee línea a línea un archivo de texto utilizando la función `fscanf( )` con los buscadores `"%[^\\n]"` donde primer corchete busca cualquier tipo de caracteres EXCEPTO el finalizador `\\n`, en tanto que `%*\\n`, solo lee dicho finalizador pero no lo guarda en ninguna variable eso lo realiza gracias al operador `*`.

**NOTA:** Se ha realizado una verificación de los caracteres incorporados a la cadena de lectura (con el lazo `for` dentro del `while`) de tal suerte que si se trata de un **LF** un **TAB** o un finalizador `'\\0'`, sean impresos en ROJO. Se ha observado que todas las cadenas carecen del par `CR+LF`, pero en cambio TODAS poseen el finalizador `'\\0'`.

### Ejemplo #3:

```
#include <conio.h>  
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
  
const TAB = 9;  
const LF = 10;  
// -----  
void main()  
{  
    FILE *Fuente;  
    char NombFuente[128] = "Texto para lectura.txt";  
    char Linea[128];  
    int i;  
  
    clrscr();  
  
    if((Fuente=fopen(NombFuente,"rt"))==NULL) {  
        printf("NO PUDO ABRIR ARCHIVO FUENTE");  
        getch(); return;  
    }  
  
    while(fscanf(Fuente,"%[^\\n]%*\\n",Linea) !=EOF) {
```

```
        for(i=0;i<=strlen(Linea);i++) {
            if((Linea[i]==TAB) || (Linea[i]==LF) || (Linea[i]=='\0'))
            {
                textcolor(LIGHTRED);
                cprintf("%4d",Linea[i]);
            }
            else{ textcolor(WHITE);
                cprintf("%c",Linea[i]);
            }
        }

        cprintf("\r\n");

    }

fclose(Fuente); getch();

}
// -----
```

Seguramente sería más ilustrativo un archivo de texto con algún formato menos elemental, por ejemplo como el siguiente:

#### **Ejemplo #4:**

Este programa lee un archivo con formato en el cual por línea se hallan escritas 3 magnitudes de punto flotante, correspondientes a la BMayor BMenor y Altura de un trapecio genérico.

Los datos serán extraídos mediante la función **fscanf( )** que en estos casos funciona realmente bien (a pesar de que para la escritura de los datos se utilizaron tabuladores).

217.38	414.29	128.67
412.38	356.66	852.37
654.28	879.32	638.29
213.55	897.33	125.69

```
/* ----- */
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>

void main()
{
    FILE    *Fuente;
    char    NombFuente[128] = "Trapecios.txt";
    double  BMayor;
```

```
double BMenor;
double Altura;

clrscr();

if((Fuente=fopen(NombFuente,"rt"))==NULL) {
    cprintf("NO PUDO ABRIR ARCHIVO FUENTE");
    getch(); return;
}

while(fscanf(Fuente,"%lf%lf%lf",&BMayor,&BMenor,&Altura)!=EOF)
    cprintf("%10.2lf %10.2lf %10.2lf\r\n",BMayor,BMenor,Altura);

fclose(Fuente);
}
/*-----*/
```

Otro caso donde también funciona bien el fscanf( ) es el siguiente:

#### **Ejemplo #5:**

Este programa lee un archivo escrito con el **Cuaderno de Notas** cuyo contenido es:

ENRIQUE SUAREZ	18	72.50	CASADO
INES MARIA RODRIGUEZ	24	51.50	SOLTERA
SOLEDAD VILLAFANEZ	37	72.00	CASADA
MARIO ALBERTO GUTIERREZ	22	68.25	CASADA
ROQUE SALOMON KRIEG	48	75.00	SOLTERO
ROMELIA DEL VALLE PEREZ	28	49.50	SOLTERA

en el cual el primer campo ocupa 30 caracteres, el segundo 10,  
el tercero 7 y el cuarto hasta el final de la línea.

Nótese que en la línea:

**fscanf(Fuente,"%30c%d%f%s\n", Nombre, &Edad, &Peso,EC);**

se han tomado en cuenta estas especificaciones.

En la impresión de salida se ha agregado el símbolo '!' a fin de visualizar los blancos  
adicionales que pudiera tener un campo de texto.

/\*-----\*/

```
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
```

```
void main()
{
    FILE *Fuente;
    char NombFuente[128]="Datos en lista.txt";
    char Linea[128];
    char Nombre[30];
    int Edad;
    float Peso;
    char EC[16];

    clrscr();

    if((Fuente=fopen(NombFuente,"rt"))==NULL) {
        printf("NO PUDO ABRIR ARCHIVO FUENTE");
        getch(); return;
    }

    while(!feof(Fuente)){

        fscanf(Fuente,"%30c%d%f%s\n", Nombre, &Edad, &Peso,EC);
        Nombre[30] = '\0';//para que le agregue el caracter terminador de linea

        printf("NOMBRE : %s%c\r\n", Nombre,'\n' );
        printf("EDAD : %d\r\n", Edad );
        printf("PESO : %2.2f\r\n", Peso );
        printf("EC : %s%c\r\n", EC,'\n' );
        printf("-----\r\n" );

    }

    fclose(Fuente); getch();
}
```

0	29	30	39	40	46	47
Nombre		Edad	Peso		EC	

En la instrucción **fscanf( )**:

**%30c** significa que lee 30 caracteres de corrido y los asigna a **Nombre**.  
**%d** lee una magnitud entera con formato int.  
**%f** lee una magnitud **float**.

**%s** una cadena de caracteres hasta el final de la línea.

A través de este programa, los datos han sido exitosamente extraídos y mostrados por pantalla.

**Ejemplo #6: resolver en clase**

.....

He aquí otro ejemplo donde fscanf( ):

Este programa también funciona muy bien con los formatos de datos, puesto que las cadenas están colocadas una a la par de otra en cada línea del archivo, en tanto que el dato numérico va al final y no crea ninguna confusión de lectura:

ENRIQUE MARIO OLMOS	Las Piedras 1111	350
FOSEFINA ALVAREZ	San Lorenzo 1234	212
JUAN JOSE PUCHETA	Asuncion 1920	180
MARIA JUAN ROMERO	Pje Centenario 914	95
RICARDO SERRANO	Av. Presidente Peron 238	256

---

**Ejemplo #7: resolver en clase** \*/

Sin embargo si tuviésemos otra disposición de campos, como por ejemplo:

10	30	19	
Cód	Dirección	Superf	Estado
1021	Av. Alem 2145	258.50	Habitable
1022	San Lorenzo 3520	185.00	Reparaciones
1023	Crisostomo Alvarez 1948	325.50	Excelente
1024	Lamadrid 4200	225.00	Habitable
1025	Amador Lucero 75	450.00	Excelente

Se presenta el inconveniente que al extraer el segundo campo (Dirección), el compilador asume que el primer blanco a partir de Cód pertenece ya a Dirección y ésta queda conformada por un conjunto de blancos anteriores que resulta necesario eliminar.  
¿Como lo resolvería para que funcione correctamente?

**Ejemplo #8: resolver en clase**

Este programa rescata cada cadena del archivo de texto utilizando la **función**:

## **fgets(Linea,n,FILE \* )**

Luego rescata cada sub cadena (campos que sean de texto), en variables cadenas de longitud adecuada usando la función `strncpy()`. Las sub cadenas que representen campos numéricos NO son extraídas de la cadena general.

Finalmente los campos que sean numéricos se convierten a su tipo original mediante las funciones de conversión **atoi( )** y **atof( )** pasándoles como argumentos la dirección de inicio del campo dentro de la línea general, por eso es que no es necesario extraerlo como sub cadena.

Las funciones de conversión utilizadas son:

<b>int atoi(const char *s)</b>	convierte la cadena "s" a un entero.
<b>double atof(const char *s)</b>	convierte la cadena "s" a un double.

En caso de éxito regresan el valor numérico correspondiente.  
En caso de fallo regresan 0.

**INSISTIMOS:** const char \*s es **LA DIRECCION** dentro de un cadena a partir de la cual **COMIENZA** una sub cadena de caracteres numéricos.

Para la extracción de las sub-cadenas se ha utilizado la función:

**char \*strncpy(char \*Destino, const char \*Fuente, nro\_de\_caracteres)**

donde **Destino** es la cadena char que recibirá el total de caracteres extraídos de la cadena Fuente (**Linea** en nuestro caso) a partir de la dirección **&Linea[ ]** y en un total de **n** caracteres (según la longitud del campo a extraer).

En conclusión, este es un método simple en su concepción aunque tal vez un poquito más largo. En contrapartida es muy confiable y sistemático.

Otro detalle importante es que hemos utilizado **la propia** función **fgets( )** como detectora del final de archivo (en lugar de **feof( )**). Es más confiable y funciona muy bien.

----- \*/

## **Lectura de un archivo carácter a carácter.**

Una lectura de este tipo tiene algunas ventajas interesantes. Para ello utilizaremos la función:

### **Int fgetc(FILE \*)**

regresa el próximo carácter leído del archivo.

- En caso de éxito retorna el carácter leído.
- En caso de fallo retorna **EOF**.
- En caso de final de archivo regresa **EOF**.

Cuando leemos un archivo de texto línea a línea, suele ocurrir que los tabuladores (ASCII=9) entran a formar parte de la línea extraída. Mediante **fgetc( )** pueden filtrarse caracteres no deseables (o convertirse a lo que en el texto representan).

El siguiente código fuente ilustra esta situación:

#### **Ejemplo #9:**

Este programa lee un archivo de texto a sabiendas de que cada línea finaliza en el par CR+LF. Por eso es que la función **LeerArchivo( )** lee carácter a carácter y detecta cuando dicho par se halla presente tomando los recaudos del caso.

---

```
----- */

#include<conio.h>
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

intLeerLinea ( FILE *, char * );
// -----

void main()
{
    FILE *Fuente;
    char  NombFuente[128] = "D:\\DISCO F\\Leng214\\Archi08.cpp";
    char  Linea[128];

    clrscr();

    if((Fuente=fopen(NombFuente,"rt"))==NULL){
        printf("NO PUDO ABRIR EL ARCHIVO FUENTE");
        getch(); return;
    }
    while(LeerLinea(Fuente,Linea)){
        printf("%s\\r\\n",Linea);
        getch();
    }
}
```

```
fclose(Fuente);
}
// -----
int LeerLinea(FILE *pF,char *Linea)
{
    const CR  = 13;
    const LF  = 10;
    const TAB = 9;

    char c;
    int i = 0;
    int j;

    while((c=fgetc(pF))!=EOF){
        switch(c) {
            case CR :   break;
            case LF :   Linea[i]='\0';
                        return(1);
            case TAB :   for(j=0;j<4;j++) Linea[i+j]=' ';
                        i=i+j;
                        break;
            default  :   Linea[i]=c; i++;
        }
    }
    return(0);
}
```

La función **LeerLinea( )** detecta además cuando el archivo ha sido leído completamente y en ese caso retorna 0.

- Realiza lecturas línea a línea sabiendo que cada línea de texto finaliza en el par **CR+LF** obvia el **CR** y en lugar del **LF** coloca un finalizador **'\0'**.
- Si encuentra en caracter **TAB** (ASCII=9) agrega 4 espacios en la cadena.

## Escribir un archivo de texto con formato.

Ahora vamos a realizar un proceso inverso: partiendo de los datos individuales, grabar un archivo de texto con formato.  
Para ello es conveniente que analicemos una función sumamente útil para este evento:



```
int sprintf(char *Buff, const char *Formatos,ListaDeVariables)
```



En caso de éxito retorna el número de bytes de la cadena.  
En caso de error, retorna **EOF**.

Esta instrucción no incluye el finalizador '\0' al final de la cadena, por lo cual tendremos que agregarlo en la cadena de formatos.

Nada mejor que un buen ejemplo para aclarar dudas:

#### **Ejemplo #10:**

Este programa genera una tabla con la función matemática:

$$F(x) = A.[\text{Sen}(x)/x]$$

y lo graba como un archivo de texto con formato utilizando la función:

**printf(Cad,"...formatos...",Variables). Para darle formato a la fila a guardar**

**fprintf(pArchivo,"%s",Cad). Para guardar dicha fila en el archivo**

La idea es importar luego esta Tabla desde Excel y procederá su graficación.

**IMPORTANTE:** Excel no acepta el punto como decimal, sino la coma.  
Por lo tanto deben convertirse a dicho símbolo.

---

```
----- */  
  
#include <conio.h>  
#include <stdio.h>  
#include <stdlib.h>  
#include <math.h>  
#include <string.h>  
  
// -----  
void main()  
{  
    double  A = 10;  
    char    Linea[128];  
    FILE    *Tabla;  
    char    NombTabla [128] = "D:\\DISCO F\\Leng214\\Tabla.dat";  
    double  Krad  = M_PI/180;  
    double  Fx;  
    int     x;  
    int     x_ini  = -360;  
    int     x_fin  = 360;
```

```
int Paso_x = 10;
int i;

clrscr();

if((Tabla=fopen(NombTabla,"wt"))==NULL) {
    cprintf("NO PUDO CREAR ARCHIVO DE DATOS");
    getch(); return;
}

for(x=x_ini;x<=x_fin;x+=Paso_x) {

    if(x==0) Fx=A;

    else Fx=A*(sin(x*Krad))/(x*Krad);

    sprintf(Linea,"%10d%10.2lf\n",x,Fx); // crea el formato.

    // --- reemplaza el (.) decimal por la (,) p/que Excel lo acepte.
    for(i=0;i<strlen(Linea);i++)
        if(Linea[i]=='.') Linea[i]=',';

    fprintf(Tabla,"%s",Linea); // lo graba en disco.
    cprintf("%s\r",Linea); // lo muestra por pantalla.

}

fclose(Tabla);
cprintf("TABLA FINALIZADA....");
getch();
}
```

Este programa genera una Tabla de valores para luego, desde Excel, importarla y graficarla con todas las facilidades que éste posee. La función matemática para la generación es:

$$F(x) = \text{Seno}(x) / x$$

para un rango de valores comprendido entre  $-360^\circ$  a  $+360^\circ$ .

Obviamente las funciones trigonométricas requieren argumentos en radianes, por lo cual definimos  $Krad = M\_PI / 180$ .

En definitiva, nuestras variables son el ángulo **x** (int) en grados, y **Fx** de tipo double.  
La línea:

```
sprintf(Linea,"%10d%10.2lf\n",x,Fx);
```

define perfectamente cómo quedará configurada cada línea a imprimir en el archivo:

- Un entero en un campo de 10 espacios.
- Un double en un campo de 10 espacios y con 2 decimales.
- Un finalizador de cadena '\0'.

Un detalle importante es que EXCEL no admite los (.) como indicador decimal, sino la coma (,). Ello lo solucionamos de manera muy sencilla:

```
for(i=0;i<strlen(Linea);i++)  
    if(Linea[i]!='.') Linea[i]=',';
```

Finalmente el volcado de **Linea** hacia el archivo destino viene dada por:

```
fprintf(Tabla,"%s",Linea);
```

## Actualizar un archivo de texto.

Esta operación consiste en abrir un archivo ya existente para agregarle información, pero abrirlo de tal manera que no se destruya lo ya existente. Para ello uno de los parámetros de fopen( ) debe ser:

**rt+** Permite actualizar un archivo existente y además para lectura/escritura.  
**at+** Abre un archivo existente, va al final, y si no existe lo crea.  
**at** Abre para actualizar, va al final, y si no existe lo crea para escritura.

El siguiente programa permite visualizar cómo trabajar con esta directiva:

### Ejemplo #11:

/\*

Este programa abre un archivo existente para agregarle información. Para ello utiliza el parámetro "**at**" en fopen( ). Esto hace que el puntero de archivo vaya al final del mismo. Luego abre otro archivo (también existente) y copia su contenido al final del primero.

**NOTA:** Para verificación se abre el archivo resultante con el Cuaderno de Notas.

----- \*/

```
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// -----
void main( )
{
    FILE *Fuente1;
    FILE *Fuente2;
    char NombFuente1[128] = "D:\\DISCO F\\Leng214\\Propiedades en venta.txt";
    char NombFuente2[128] = "D:\\DISCO F\\Leng214\\Propiedades en venta II.txt";
    char Linea[128];

    clrscr();

    if((Fuente1=fopen(NombFuente1,"at"))==NULL) {
        cprintf("NO SE PUEDE ABRIR ARCHIVO FUENTE1");
        getch(); return;
    }

    if((Fuente2=fopen(NombFuente2,"rt"))==NULL) {
        cprintf("NO SE PUEDE ABRIR EL ARCHIVO FUENTE2");
        fclose(Fuente1); getch(); return;
    }

    while(fgets(Linea,127,Fuente2)!=0) {
        fputs(Linea,Fuente1);
    }

    fclose(Fuente1); fclose(Fuente2);
    cprintf(" ARCHIVO ACTULIZADO EXITOSAMENTE");

}
// -----
```

Aquí hemos utilizado el par **fgets( )**, **fputs( )** que ya habíamos analizado anteriormente.  
Hemos tenido cuidado de abrir el segundo archivo en modo de lectura a fin de no dañarlo.

La función:

**Fuente1=fopen(NombFuente1,"at")**

prepara al archivo original para agregarle datos.  
Como siempre es conveniente verificar si el archivo pudo abrirse.

## Reposicionar el puntero de archivo al comienzo del mismo.

Si al ejecutar el programa que lee o escribe sobre un archivo de texto, se torna necesario reposicionar el puntero al inicio del mismo, existe la función **rewind( )**. He aquí un ejemplo:

```
/* --- TALLER DE LENGUAJES I 20104
Leer y releer con rewind.cpp
```

Este programa abre un archivo de texto existente, lee en vacío hasta el final del mismo y a continuación reposiciona el puntero de archivo al comienzo del mismo mediante "rewind( )". Procede nuevamente a leer línea a línea, pero esta vez mostrando por pantalla.

```
----- */

#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// -----
void main( )
{
    FILE *Fuente;
    char NombFuente[128]= "D:\\DISCO F\\Leng214\\Propiedades en venta.txt";
    char Linea[128];

    clrscr();

    if((Fuente=fopen(NombFuente,"rt"))==NULL){
        printf("NO PUDO ABRIR ARCHIVO FUENTE");
        getch(); return;
    }

    // ... busca el final del archivo -----

    while(fgets(Linea,127,Fuente)!=0);

    rewind(Fuente);
    while(fgets(Linea,127,Fuente)!=0){
        Linea[strlen(Linea)-1]='\0';
        printf("%s\r\n",Linea);
    }

    fclose(Fuente);
}
```

// -----

La línea anterior al rewind( ), lee en vacío el archivo con la intención de llevar el puntero al final del mismo. A continuación lo reposiciona al inicio con instrucción rewind( ).

A continuación algunos ejemplos con archivos de texto.

/\* --- TALLER DE LENGUAJES I 2014  
Buscar medida en pulgadas.cpp

Este programa toma una medida nominal en mm y busca el valor mas proximo en pulgada, considerando una precisión de 1/16 de pulg. Para ello se valdrá de un archivo de texto con medidas normalizadas entre 1/16 a 3 pulgadas en el cual la primera columna son su equivalente en mm y la segunda su nomenclatura en pulg. Este archivo ha sido confeccionado

desde EXCEL el cual toma la coma (,) como indicador decimal. Este simbolo debe ser cambiado a (.) al leer cada línea de archivo a fin de que C pueda interpretarlo correctamente.

----- \*/

```
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
```

// -----

```
void main()
{
    FILE *Fuente;
    char NombFuente[128] = "D:\\DISCO F\\Leng214\\Tabla en pulgadas.txt";
    char Linea[128];
    float MedNom = 21.3; // --- en [mm]
    float MedLeida;
    float MedAnt;
    float DiffAnt = 1000;
    float DiffAct;
    char CodEnPulgAct[16];
    char CodEnPulgAnt[16];
    int i;

    clrscr();

    if((Fuente=fopen(NombFuente,"rt"))==NULL){
```

```
        cprintf("NO PUDO ABRIR ARCHIVO FUENTE");
        getch(); return;
    }

    while(fgets(Linea,127,Fuente)!=0) {
        Linea[strlen(Linea)-1]='\0';        // ...eliminar el LF
        *(strstr(Linea,","))='.';            // ...cambia (,) por (.)
        cprintf("%s\r\n",Linea);

        MedLeida=atof(&Linea[0]);            // ...rescata 1ra columna
        strncpy(CodEnPulgAct,&Linea[16],16); // ... cod en pulg como cad
        DiffAct=fabs(MedLeida-MedNom);
        if(DiffAct>DiffAnt) break;
        else DiffAnt=DiffAct;
        MedAnt=MedLeida;
        strcpy(CodEnPulgAnt,CodEnPulgAct);

    }

    cprintf("MedLeida = %2.3f\r\n",MedAnt);
    cprintf("CodEnPulg = %s\r\n",CodEnPulgAnt);

    fclose(Fuente);
}

/* --- TALLER DE LENGUAJES I 2014
    Cargar un vector con datos en archivo.cpp

    Este programa define un vector de enteros de 16 elementos al cual carga-
    ra con valores extraídos de un archivo de texto configurado en una sola
    línea. Para ello utilizara una lectura con formato mediante fscanf( ).

    ----- */

#include <conio.h>
#include <stdio.h>
#include <stdlib.h>

const DIM = 16;
typedef int TVect[DIM];

// -----
void main( )
{
    FILE *Fuente;
    char NombFuente[128] = "D:\\DISCO F\\Leng214\\Valores para el vector.txt";
```

```
TVect Vect;  
int i;  
  
clrscr( );  
  
if((Fuente=fopen(NombFuente,"rt"))==NULL) {  
    printf("NO PUDO ABRIR ARCHIVO FUENTE");  
    getch( ); return;  
}  
  
for(i=0;!feof(Fuente);fscanf(Fuente,"%d\n",&Vect[i]),i++);  
  
fclose(Fuente);  
for(i=0;i<DIM;i++)  
    printf("%4d\r\n",Vect[i]);  
}  
// -----
```

El archivo "Valores para el vector.txt" contiene los siguientes datos:

125 317 425 785 602 818 520 189 732 911 635 455 333 795 183 506