



# Laboratorio 5

## Programación de Computadores (2023-2)



# Temas del Laboratorio 5

- Manejo de archivos
- Ejercicios

# Stdio FILEs y streams

stdio.h tiene un tipo de variable FILE que se utiliza para acceder a streams a través de operaciones de escritura y lectura.

Los streams se pueden utilizar para mandar información al disco (archivos) o a la terminal (texto)

```
// stdout, stdin y stderr son variables tipo *FILE que ya estan definidas
printf("Hola mundo\n");    /*<==>*/    fprintf(stdout,"Hola mundo\n");
```

# Uso de buffers

Un buffer es un área de almacenamiento temporal que se utiliza para guardar y gestionar datos mientras se transfieren de un lugar a otro.



```

1  #include <stdio.h>
2  #include <string.h>
3  #include <stdbool.h>
4
5  #define MAXCHAR 1000
6
7  int main() {
8
9      printf("#####\n");
10     printf("### Leer un archivo de texto ###\n");
11     printf("#####\n\n");
12
13     // Abrimos el stream para leer el archivo CSV
14     FILE *fp = fopen("ejemplo.txt", "r");
15
16     // Declaramos un buffer para leer las líneas del archivo
17     char buffer[MAXCHAR];
18     char *token;
19
20     int n_linea = 1;
21     // Mientras queden líneas por leer, avanzamos
22     while (feof(fp) != true) {
23         // Leemos la línea actual y la dejamos copiada en buffer
24         if(fgets(buffer, MAXCHAR, fp)) {
25             printf("Línea %d: %s", n_linea, buffer);
26             n_linea++;
27         }
28     }
29
30     return 0;
31
32     /******
33     /* Pregunta: ¿Podemos leer archivos más grandes que nuestra /*
34     /*          memoria RAM? ¿Cómo sería posible?                /*
35     /******
36
37 }

```

```
#define MAXCHAR 1000
```

```
// Declaramos un buffer para leer las líneas del archivo
char buffer[MAXCHAR];
```

```
// Abrimos el stream para leer el archivo CSV
FILE *fp = fopen("ejemplo.txt", "r");
```

```

int n_linea = 1;
// Mientras queden líneas por leer, avanzamos
while (feof(fp) != true) {
    // Leemos la línea actual y la dejamos copiada en buffer
    if(fgets(buffer, MAXCHAR, fp)) {
        printf("Línea %d: %s", n_linea, buffer);
        n_linea++;
    }
}

```

# Algunas funciones útiles

**fopen(“archivo”, “modo”)** —> Abre un archivo en ciertos modos como “r” lectura, “w” escritura, “wb” escritura binaria “a” agregar (append), etc.

**fclose(“archivo”)** ----> Cierra un archivo.

**fgets(buffer, largo, file)** —> Lee una línea de cierto largo, obtenida de un file y la guarda en buffer.

**fwrite(\*bloque de mem, tamaño en bytes, num de elementos, archivo)**

—> Usado mayormente para escritura binaria

**fprintf(archivo, “tipo de salida”, variables)** ----> Usado para escribir en un archivo como si fuera un printf



# Algunas funciones útiles

**fseek(archivo, número de bytes a mover, punto de referencia)** ----> Se usa para mover el puntero que apunta al archivo, ejemplo :

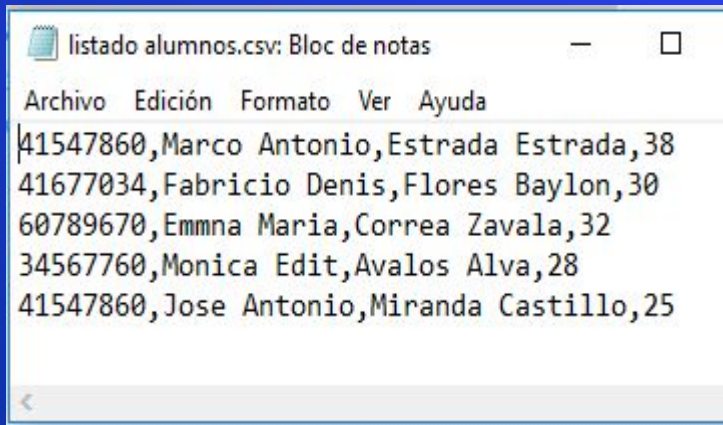
`fseek(file, 0L, SEEK_END)` // Estando en el inicio, mueve el puntero al final

**ftell(file)** ----> Entrega el número de bytes que hay desde el inicio hasta la posición actual del puntero del archivo

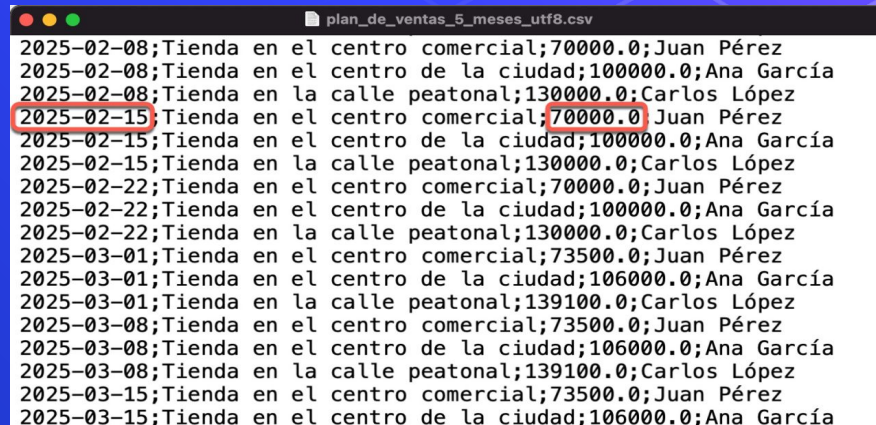
**feof(file)** ----> Devuelve true si se ha llegado al final del archivo

# Archivos csv

Los archivos .csv (Comma Separated Values) son un tipo de archivo que como su nombre dice, contiene datos separados por comas (en realidad puede ser cualquier token), son un tipo de archivo escrito en texto plano, que muchos sistemas interpretan como planillas, por su estilo de escritura basado en columnas, su finalidad es albergar datos de forma simple y compatible para varios sistemas.



```
listado alumnos.csv: Bloc de notas
Archivo  Edición  Formato  Ver  Ayuda
41547860,Marco Antonio,Estrada Estrada,38
41677034,Fabricio Denis,Flores Baylon,30
60789670,Emmna Maria,Correa Zavala,32
34567760,Monica Edit,Avalos Alva,28
41547860,Jose Antonio,Miranda Castillo,25
```

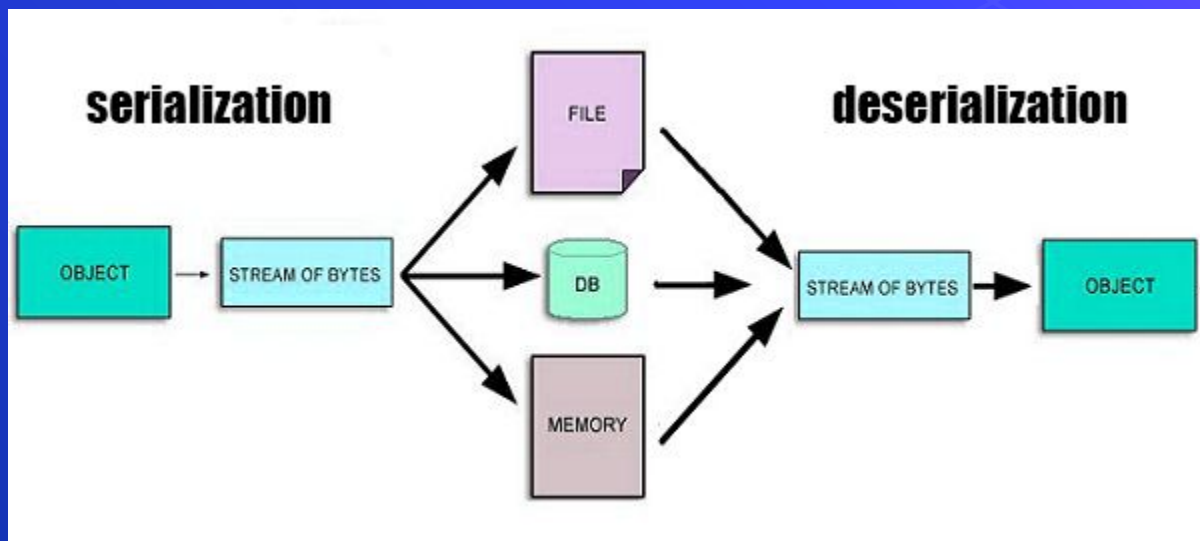


```
plan_de_ventas_5-meses_utf8.csv
2025-02-08;Tienda en el centro comercial;70000.0;Juan Pérez
2025-02-08;Tienda en el centro de la ciudad;100000.0;Ana García
2025-02-08;Tienda en la calle peatonal;130000.0;Carlos López
2025-02-15;Tienda en el centro comercial;70000.0;Juan Pérez
2025-02-15;Tienda en el centro de la ciudad;100000.0;Ana García
2025-02-15;Tienda en la calle peatonal;130000.0;Carlos López
2025-02-22;Tienda en el centro comercial;70000.0;Juan Pérez
2025-02-22;Tienda en el centro de la ciudad;100000.0;Ana García
2025-02-22;Tienda en la calle peatonal;130000.0;Carlos López
2025-03-01;Tienda en el centro comercial;73500.0;Juan Pérez
2025-03-01;Tienda en el centro de la ciudad;106000.0;Ana García
2025-03-01;Tienda en la calle peatonal;139100.0;Carlos López
2025-03-08;Tienda en el centro comercial;73500.0;Juan Pérez
2025-03-08;Tienda en el centro de la ciudad;106000.0;Ana García
2025-03-08;Tienda en la calle peatonal;139100.0;Carlos López
2025-03-15;Tienda en el centro comercial;73500.0;Juan Pérez
2025-03-15;Tienda en el centro de la ciudad;106000.0;Ana García
```



# Serialización

Serializar un objeto consiste en transformar su estado en un formato (como un flujo de bytes o texto) que permite almacenarlo o transmitirlo, de manera que luego pueda recuperarse y reconstruirse mediante el proceso inverso, llamado deserialización.



# ¿Cómo serializar?

El proceso es bien simple y hoy nos centraremos en serialización binaria.

- 1- Seleccionar un objeto, puede ser un arreglo, estructura, objeto, etc.
- 2- Crear un flujo de salida, abrir un archivo o buffer en modo binario.
- 3- Convertir el objeto a bytes, o usar una función de escritura que haga automáticamente el proceso como **“fwrite()”**.
- 4- Escribir el flujo de bytes en el archivo.
- 5- Cerrar el flujo.

Esto genera un archivo .bin que contiene la información del archivo en bytes.

# ¿Cómo deserializar?

Este proceso de igual forma es muy simple, sin embargo puede variar al momento de reconstruir el objeto.

1- Abrir un archivo o buffer en modo lectura.

2- Leer bytes, esto se puede hacer de diversas formas, en C funciones como **ftell()**, **fseek()**, **fread()** pueden ser útiles.

3- Reconstruir el objeto, transformar los bytes al objeto original, en muchos casos podemos hacer una lectura de los datos en vez de deserializar por completo, bastaría con interpretar los datos del archivo, no necesariamente hay que volver a crear el mismo objeto, si nuestro objetivo es sólo interpretar su contenido.

4- Cerrar el flujo.

# Ejercicio 1

No solo podemos trabajar con archivos .txt en C, podemos abrir muchos tipos de archivos, entre ellos los .csv, en Github estará un archivo .csv que contiene todos los juegos ingresados en metacritic hasta cierta fecha, incluyendo nombre, r-date, desarrollador, nota de usuario,etc.

Usando el archivo games-data.csv genere un programa en C que pida 2 enteros positivos entre 0 y 100 , y luego imprima en pantalla el nombre de todos los juegos que su “score” se encuentre entre los valores ingresados.

```
stack@stack-Nitro-ANV15-51:~/Documentos/Labs C/Lab 5$ gcc archivo_CSV.c -o 1
stack@stack-Nitro-ANV15-51:~/Documentos/Labs C/Lab 5$ ./1
Ingrese un valor minimo y maximo respectivamente
Puntuacion minima: 60
Puntuacion maxima: 80

The Book of Unwritten Tales 2: 80
Sid Meier's Civilization Revolution: 80
Donkey Kong Jungle Beat: 80
Call of Duty: Modern Warfare: 80
Dungeon Defenders: 80
Ultimate Marvel vs. Capcom 3: 80
I Am Dead: 80
Mario Power Tennis: 80
Final Fantasy VIII Remastered: 80
The Sims 3: 80
```

## Ejercicio 2

Junto al archivo del ejercicio anterior, encontrarán un .c que deben editar, ignoren las funciones para medir tiempo, son necesarias para después.

Debe crear un programa que tome un arreglo de enteros (previamente creado en el programa) y se le aplique serialización binaria, luego abrir el mismo archivo e interpretar sus datos, es decir que imprima en pantalla la cantidad de bytes y la cantidad de números enteros presentes en el archivo.

¿Qué puede decir los tiempos de ejecución?

¿Será más rápido escribir y leer el array directamente en un txt?