Unidad IV.

ARCHIVOS Y FICHEROS (ALMACENAMIENTO PERSISTENTE)

3.1. Ficheros en lenguaje c.
3.2. Apertura y cierre de ficheros.
3.3. Lectura y escritura en ficheros
3.4. Recorrido de un fichero secuencial (feof)
3.5 Acceso directo a los datos (fseek)

COMPETENCIA A DESARROLLAR:

El estudiante:

- Realiza operaciones con ficheros digitales, mediante los comandos de apertura, escritura; aplicando operaciones de recorrido, y accesos directos.

3.1 FICHEROS EN LENGUAJE C

Los ficheros, en contraposición con las estructuras de datos vistas hasta ahora (variables simples, vectores, matrices, registros, etc.), son estructuras de datos almacenadas en memoria secundaria.

El formato de declaración de un fichero es el siguiente:

FILE * <*nombrefichero****>;***

Ejemplo

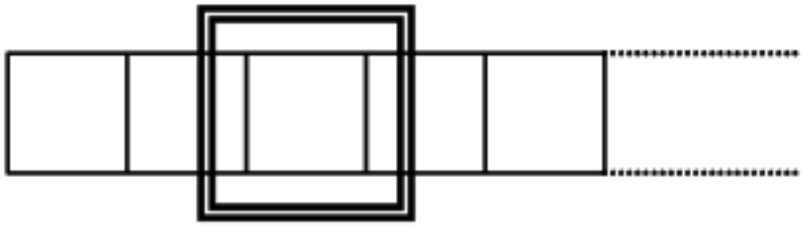
FILE *archi; *//puntero a un archivo*

En lenguaje C, todos los ficheros almacenan bytes cuando se realiza la apertura y la escritura se decide cómo y qué se almacena en el mismo, durante la declaración del fichero no se hace ninguna distinción sobre el tipo del mismo.

Los ficheros pueden ser:

- ⇒ **Texto** sirven para almacenar caracteres.
- ⇒ **Binario** para almacenar cualquier tipo de dato.

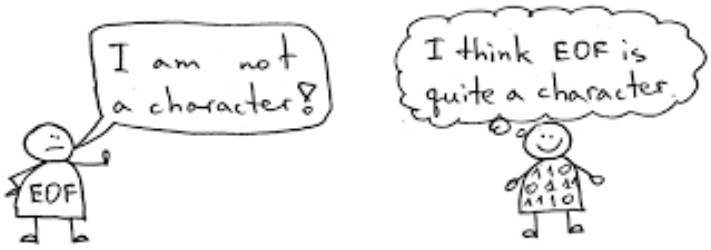
Al estar en memoria secundaria, no todos los elementos del fichero son accesibles de forma inmediata. Solamente se puede acceder cada vez a un único elemento del fichero, que se denomina ventana del fichero:



Dependiendo de cómo se desplaza la ventana por el fichero, podemos distinguir dos tipos de acceso:

⇒ **Acceso secuencial:** La ventana del fichero sólo puede moverse hacia delante a partir del primer elemento y siempre de uno en uno.
⇒ **Acceso directo**: La ventana del fichero se puede situar directamente en cualquier posición del fichero. Es un acceso similar al utilizado en los arrays.

EOF(end of file)→significa fin de archivo



3.2 APERTURA Y CIERRE DE FICHEROS

Hasta ahora, para obtener y almacenar datos de una estructura de datos bastaba con realizar asignaciones a la misma. Para utilizar los ficheros el procedimiento es distinto.

Antes de usar un fichero es necesario realizar las siguientes operaciones:

⇒ **operación de apertura** del mismo.

⇒ **operación de escritura** si se desea almacenar datos en el fichero.

⇒ **operación de lectura** si se quiere obtener datos del fichero.

⇒ **operación de cierre** cuando ya no se quiera utilizar el fichero se realiza el cierre, para liberar parte de la memoria principal que pueda estar ocupando (aunque el fichero en sí está almacenado en memoria secundaria, mientras está abierto ocupa también memoria principal).

La instrucción más habitual para abrir un fichero es:

***FILE** * fichero;*

fichero = fopen ("nombre-fichero", "modo");

La función ***fopen*** devuelve un *puntero a un fichero* que se asigna a una variable de tipo fichero. Si existe algún tipo de error al realizar la operación, por ejemplo, porque se desee abrir para leerlo y éste no exista, devuelve el valor **NULL.**

Donde:

- ⇒ ***nombre-fichero*** será una cadena de caracteres que contenga el nombre (y en su caso la ruta de acceso) del fichero.
- ⇒ ***modo*** es una cadena de caracteres que indica el tipo del fichero (texto o binario) y el uso que se va a hacer de él lectura, escritura, añadir datos al final, etc. Los modos disponibles son:

MODO	DESCRIPCIÓN
r	Abre un fichero para lectura. Si el fichero no existe devuelve error.
w	Abre un fichero para escritura. Si el fichero no existe se crea, si el fichero existe se destruye y se crea uno nuevo.
a	Abre un fichero para añadir datos al final del mismo. Si no existe se crea.
+	Símbolo utilizado para abrir el fichero para lectura y escritura.
t	El fichero es de tipo texto. Si no se pone ni b ni t el fichero es de texto. Los modos anteriores se combinan para conseguir abrir el fichero en el modo adecuado.
b	El fichero es de tipo binario

Por ejemplo, para abrir un fichero binario ya existente para lectura y escritura el modo será **"rb+ "**;
Si el fichero no existe, o aun existiendo se desea crear, el modo será **" wb+ ".**
Si deseamos añadir datos al final de un fichero de texto bastará con poner **"a".**

La forma habitual de utilizar la instrucción **fopen** es dentro de una sentencia condicional que permita conocer si se ha producido o no error en la apertura

Por ejemplo:

```
FlLE *fich;
        if ((fich = fopen("nomfich.dat", "w")) == NULL)
                {       /* control del error de apertura * /
                        cout<< "Error en la apertura. Es posible que el fichero no
                                exista \n ";
                }
```

Cuando se termine el tratamiento del fichero hay que cerrarlo, el cierre se hará con la función **fclose (fich);**

Para utilizar las instrucciones de manejo de ficheros que veremos en esta unidad es necesario incluir la librería **<stdio.h>.**

3.3 LECTURA Y ESCRITURA EN FICHEROS DE TEXTO

Para almacenar datos en un fichero es necesario realizar una operación de escritura, de igual forma que para obtener datos hay que efectuar una operación de lectura.

En C existen muchas y variadas operaciones para leer y escribir en un fichero; entre ellas tenemos: fgetc -fputc, fgets - fputs, fread -fwrite, fscanf -fprintf

.

a) Lectura y escritura de caracteres (fgetc – fputc) y cadenas (fgets – fputs)

Los formatos de las instrucciones de lectura y escritura de caracteres y cadenas son:

⇒ **fgetc** lee un carácter del fichero, el carácter leído se almacenará en carácter leído. Cuando se llega al final del fichero devuelve EOF.

letra = fgetc (fichero)

⇒ **fputc** escribe el carácter car en el fichero. Devuelve el carácter escrito o EOF en caso de error.

fputc ('*', fichero)

POR EJEMPLO: Programa para guardar caracteres en un fichero de texto

```
#include <iostream>
#include <stdio.h>
using namespace std;

int main()
{
   FILE *archi;
   char car='*';
   //creacion de fichero
   if((archi=fopen("datos","w"))==NULL){
   cout<<"Error de apertura puede que le archivo no exista!!!";
   }
   fputc(car,archi);//guarda en fichero
   fputc('&',archi);
   fputc('g',archi);
   fclose(archi);
```

```
    //mostrar fichero
    if((archi=fopen("datos","r"))==NULL){
    cout<<"Error de apertura puede que le archivo no exista!!!";
    }
   cout<<"EL FICHERO CONTIENE LOS SIGUIENTES
CARACTERES\n";
    car=fgetc(archi);//lee caracter de fichero
    while(car!=EOF){
     cout <<car;
     car=fgetc(archi);
    }
    return 0;
}
```

b) fgets (cadena_leida, num_caracteres, fichero)

Lee **num_caracteres** del fichero y los almacena en **cadena_leida** colocando el carácter de fin de cadena '\0' en la posición **num_caracteres** de la cadena leída.

POR EJEMPLO: Programa para guardar una cadena en un fichero

```
#include <iostream>
#include <stdio.h>
using namespace std;

int main()
{
    FILE *archi;
    char cade[15];
    char cade1[15]="";
    //creacion de fichero
    if((archi=fopen("docu","w"))==NULL){
    cout<<"Error de apertura puede que le archivo no exista!!!";
    }
    cout<<"nombre de tu mejor amigo?";
    fflush(stdin);
    gets(cade);//LEE UNA CADENA DESDE TECLADO
    fputs(cade,archi);//guarda una cadena en el fichero
    fclose(archi);

    //mostrar fichero
    if((archi=fopen("docu","r"))==NULL){
    cout<<"Error de apertura puede que le archivo no exista!!!";
```

```
    }

   cout<<"EL CONTENIDO DEL FICHERO ES \n";
    //fgets(cade,15,archi);//lee una cadena del fichero
    while(!feof(archi)){
     fgets(cade1,15,archi);//lee una cadena del fichero en cade1
     cout <<cade1;
    }
    fclose(archi);
    return 0;
}
```

EJERCICIO DE APLICACIÓN: copia de un fichero de texto en otro

3.4 LECTURA Y ESCRITURA EN FICHEROS BINARIOS

c) Lectura y escritura de bloques (fread – fwrite)

Para leer y escribir en ficheros que no sean de texto las operaciones que se deben utilizar son fread y fwrite.

El formato de escritura en bloque es el siguiente:

fwrite (direcc_dato, tamaño_dato, numero_datos, punt_fichero);

Escribe tantos datos como indique número de datos en el fichero, tomando los datos a partir de la dirección del dato.

Para calcular el tamaño en bytes de un dato o un tipo de dato se suele utilizar la función **sizeof (dato) o sizeof (tipo-de-dato);**

Por ejemplo:

```
int i, v[3];        sizeof (i) daría lo mismo que sizeof (int)
                    sizeof (v) daría 3 veces el resultado de sizeof (v[1])
```

Por ejemplo:

```
FlLE *f;
int v[6], elem_escritos, num;
f = fopen (“datos.cht ", "wb ");

/* Para escribir los 3 últimos elementos de v (el 2, el 3 y el 4) */

 elem_escritos = fwrite (&v[2], sizeof(int), 3, f );

/* Para escribir el primer elemento de v, valen las 2 instrucciones
   siguientes */
```

```
fwrite (v, sizeof (int), 1, f );
fwrite (&v[0], sizeof(int), 1, f );

/* Para escribir un entero valen las dos siguientes */

fwrite (&num, sizeof(int), 1, f);
fwrite (&num, sizeof(num), 1, f);
```

La sentencia de lectura de bloque es la siguiente:

fread (direcc_dato, tamaño_dato, numero_datos,punt_fichero);

Lee tantos datos como indique número de datos del fichero, colocando los datos leídos a partir de la dirección del dato. Los datos tienen que tener tantos bytes como especifique tamaño del dato. La función **fread** devuelve el número de elementos leídos, y el valor devuelto debe coincidir con número de datos.

Por Ejemplo:

```
f = fopen (“datos.dat ", "rb ");
elem-escritos = fread (&v[2], sizeof(int), 3, f);
fread (v, sizeof(int), 1, f);
fread (&V[0], sizeof(int), 1, f);
fread (&num, sizeof(int), 1, f);
fread (&num, sizeof(num), 1, f);
```

d) Lectura y escritura formateada de texto (fscanf – fprintf)

Para que lo escrito con **fprintf** pueda ser correctamente leído con **fscanf** es conveniente que el formato en el que se indican los tipos de datos que se van a leer o escribir sean similares para ambas instrucciones, que los tipos de datos estén separados, por ejemplo, por un blanco y que tengan un fin de línea al final.

Por ejemplo: los siguientes pares de instrucciones de lectura y escritura serían compatibles:

```
int num;
char car, cad [10] ;
FILE *f.
/* apertura del fichero */
.....
.....
fscanf (f, "%d %c %s ", &num, &car, cad);
fprintf ( f, "%d %c %s \n ", num, car, cad);
fscanf (f, "%s %c %d ", cad, &car, &num);
fprintf (f, "%s %c %d \n ", cad, car, num);

 }
```

3.5 RECORRIDO DE UN FICHERO SECUENCIAL (feof)

La lectura y escritura en un fichero se realizan en la posición en la que se encuentra el puntero del fichero. Al abrir el fichero el puntero está antes del primer dato del mismo. Cada vez que se realiza una operación de lectura o escritura el puntero se mueve hasta apuntar al dato y después lo lee o escribe.

Por ejemplo, en un fichero f con dos datos (0 y 1) al abrir el fichero tendríamos la siguiente situación:

Posición del puntero	***puntero*** →		
Datos		0	1	**EOF**

Si realizamos la operación **fread(&i, sizeof(int), 1, f);** en la variable **i** tendremos almacenado el 0 y la situación después de la lectura será:

Posición del puntero	**puntero** →		
Datos	0	1	**EOF**

Para leer todos los datos de un fichero basta con realizar lecturas sucesivas hasta que se lee el **final del fichero**. (leer hasta que *fgetc( )* devuelva *EOF).*

Esta operación es correcta, pero es más habitual utilizar una función de C que nos indica cuándo se ha leído el último dato:

feof (fichero)

devuelve un valor distinto de 0 cuando se ha alcanzado el final del fichero.

3.6 ACCESO DIRECTO A LOS DATOS *(fseek)*

Cuando se puede acceder a cualquier dato de un fichero sin tener que pasar por anteriores se está realizando un acceso directo a los datos.
La función que permite situarse en un determinado dato del fichero es:

fseek (fichero, posicion, origen);

que coloca el puntero del fichero a tantos bytes del origen como indica posición contando a partir del origen señalado. Los orígenes posibles son:

SEEK_SET o 0 → Principio del fichero.
SEEK_CUR o 1 → Posición actual.
SEEK_END o 2 → Final del fichero.

fseek devuelve 0 si no ha habido ningún error.

Por ejemplo, en un fichero que almacene números enteros la instrucción:

fseek (f, 0, SEEK-SET*)* → coloca el puntero al principio del fichero.

fseek (f, 3*sizeof(int), SEEK-CUR)→ coloca el puntero 3 posiciones más allá de la posición actual del puntero.

Para saber cuál es la posición en la que está el puntero del fichero C proporciona la función siguiente: **ftell (fich);**

que devuelve la posición actual en bytes del puntero del fichero con respecto al principio del mismo.

EJEMPLO: Programa para guardar y mostrar datos de un fichero binario

```
#include <iostream>
#include<stdio.h>
using namespace std;
//estructura de lista de clientes
struct datos{
int num;
char nombre[20];
char genero;
};
void crear_fichero();
void leer_fichero();
//PROGRAMA PRINCIPAL
int main()
{
   cout << "ARCHIVOS BINARIOS" << endl;
   crear_fichero();
   leer_fichero();
   return 0;
}
//PROCEDIMIENTO CREAR FICHERO BINARIO
void crear_fichero(){
FILE *f;
datos r;
int i,n;
if((f=fopen("fichero.dat","ab"))==NULL)
{
   cout<<"error en apertura";
}
cout<<"cuantos clientes tiene?";
cin>>n;
for(i=1;i<=n;i++){
   r.num=i;//asignar numero al cliente
   cout<<"Ingrese su nombre:";
   fflush(stdin);
   gets(r.nombre);
   cout<<"Ingrese Genero masculino-->m femenino-->f";
```

```
    cin>>r.genero;


  fwrite(&r,sizeof(r),1,f);
  }
  fclose(f);
  }
  //PROCEDIMIENTO LEER DE FICHERO BINARIO
  void leer_fichero()
  {
  FILE *f;
  datos r;
  if((f=fopen("fichero.dat","rb"))==NULL)
  {
     cout<<"error en apertura";
  }
  fread(&r,sizeof(r),1,f);
  while(!feof(f)){
  cout<<"numero: "<<r.num<<"  "<<"nombre: "<<r.nombre<<" "<<"genero:
  "<<r.genero<<" ";
  fread(&r,sizeof(r),1,f);
  }
  fclose(f);
  }
```

3.7 TIPOS DE DATO DEFINIDO POR EL USUARIO

- **REGISTRO O ESTRUCTURA→**Permite almacenar en una sola estructura datos de diferente tipo

```
struct estudiantes{
int edad; //campo edad del registro
char nombre[20];
bool estado;
float nota;
}

//PROGRAMA PRINCIPAL
/nt main(){

estudiantes es; //VARIABLE DE TIPO ESTUDIANTES

sum=es.edad; // acceso a un campo de la estructura o estructura

estudiantes aes[10];

}
```

- **ARREGLO**

0	1	2	3	4	5	6	7	8	9

- **ARREGLO DE ESTRUCTURAS**

MIS ESTUDIANTES

POSI/DIRE	EDAD	NOMBRE	ESTADO	NOTA

- **IMPRESIÓN DE DATOS DE LA ESTRUCTURA**

```
cout<<" edad:" <<aes[0].edad;
cout<< " nombre:"<<aes[0].nombre;
cout<< " estado"<<aes[0].estado;

cout<<" edad:" <<aes[1].edad;
cout<< " nombre:"<<aes[1].nombre;
cout<< " estado"<<aes[1].estado;
```

- **LECTURA DE DATOS DE LA ESTRUCTURA**

```
cout<< " INGRESE EDAD DEL ESTUDIANTE";
cin>>aes[0].edad
```