



UNIDAD 5.- ARCHIVOS

OBJETIVOS:

- Que el alumno comprenda y aprenda el concepto y cómo y dónde utilizar los archivos.
- Que el alumno realice las aplicaciones con esta estructura de datos.

TEMAS:

- 5.1.** Definición.
- 5.2.** Tipos de archivos: organizaciones y accesos.
- 5.3.** Funciones para el manejo de archivos.
- 5.4.** Tratamiento de archivos: altas, baja lógica, baja física, modificaciones, consultas, listados; representación de archivos en lenguaje C.



5.1. DEFINICIÓN

Desde que se comienza a codificar programas se utilizan variables, las cuales se usan según la aplicación específica. Esas variables primero se declaran y luego se usan en el programa para obtener el resultado deseado.

Surge un inconveniente cuando se deben ingresar datos en dichas variables, y se necesita continuar con este ingreso después de un intervalo o tiempo. Las mismas son almacenadas en memoria RAM, que es una memoria volátil, es decir, que cuando se sale del entorno del programa o se apaga la computadora, los valores de las mismas se borran indefectiblemente.

Se crearon para ello otro tipo de estructuras de datos, que almacenan valores en memorias no volátiles, es decir que perduran en el tiempo (salvo un error o una sobreescritura intencionada). Esas memorias no volátiles son los discos rígidos, pendrives, DVD, etc, que guardan los datos en forma permanente y permiten agregar y procesar los mismos luego del tiempo que se desee.

Este tipo de datos se denomina **archivo**; estructura de datos que existen en todos los lenguajes de programación.

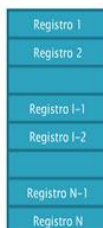
5.2. TIPOS DE ARCHIVOS: ORGANIZACIONES Y ACCESOS

ORGANIZACIONES SECUENCIAL

Es la forma básica de organizar un conjunto de registros, que forman un archivo, utilizando una organización secuencial. En un archivo organizado secuencialmente, los registros quedan grabados consecutivamente cuando el archivo se utiliza como entrada. En la mayoría de los casos, los registros de un archivo secuencial quedan ordenados de acuerdo con el valor de algún campo de cada registro. Semejante archivo se dice que es un archivo ordenado; el campo, o los campos, cuyo valor se utiliza para determinar el ordenamiento es conocido como la llave del ordenamiento. Un archivo puede ordenarse ascendente o descendentemente con base en su llave de ordenamiento.

La forma más común de estructura de archivo es el archivo secuencial. En este tipo de archivo, se usa un formato fijo para los registros. Todos los registros tienen el mismo tamaño, constan del mismo número de campos de tamaño fijo en un orden particular. Como se conocen la longitud y la posición de cada campo, solamente los valores de los campos se necesitan almacenarse; el nombre del campo y longitud de cada campo son atributos de la estructura de archivos.

Ejemplo de Archivo Secuencial.





Archivo secuencial es la forma más simple de almacenar y recuperar registros de un archivo. En esta forma de organización, se almacenan los registros uno tras otro. El primer registro almacenado se coloca al principio del archivo. El segundo se almacena inmediatamente después (no existen posiciones sin uso), el tercero después del segundo, etc. Este orden nunca cambia en la organización secuencial.

Una característica de los archivos secuenciales es que todos los registros se almacenan por posición: de primer registro, segundo registro etc.

Modo de Acceso: La forma de acceder a los registros en un archivo secuencial es secuencial, es decir uno a continuación de otro.

5.3. FUNCIONES PARA EL MANEJO DE ARCHIVOS BINARIOS

Existen dos tipos de archivos (de acuerdo a la forma de almacenar datos en la memoria externa no volátil): los archivos de texto y binarios.

- 1) Archivos de texto: poseen texto puro, por ejemplo los archivos fuente de cada programa.

Se trata de una colección de los caracteres ASCII de cada letra, número o cualquier símbolo y que tiene la particularidad de poderse leer directamente con los comandos de inspección de un archivo del Sistema Operativo que se trate (por ej, type para DOS, etc.)

- 2) Archivos binarios: contienen secuencias o bloques de valores binarios, que representan información grabada por el usuario. Estos archivos pueden guardar elementos del tipo que el usuario desee. Es común que posean registros, debido a que el registro es una estructura de datos que permite describir muy bien a cualquier entidad de la cual se quiera almacenar datos. Un archivo binario puede ser de cualquier tipo de datos: enteros, flotantes, carácter, etc.

El lenguaje C provee una gran cantidad de formas y funciones para implementar un archivo. Se darán los lineamientos generales para la construcción de ellos con un tipo de datos determinado o bien mediante estructuras (registros).

Todo Sistema Operativo guarda locaciones o direcciones de memoria para las variables que fueron declaradas, las cuales pueden ser almacenadas en otras variables que puede manejar el usuario. Estas variables son especiales y se denominan variables de tipo puntero, es decir, pueden almacenar una dirección de memoria de otra variable, éstas son llamadas variables de tipo puntero.

Declaración de un puntero: se escribe el tipo de la variable a la que se quiere apuntar y luego el nombre del puntero pero precedido por el asterisco. El tipo de la variable a la cual apuntará el puntero, se le llama tipo base del puntero.

Ej: `int *p;` /*la variable puntero p apunta a una variable de tipo int. El tipo base es int*/



Para declarar un puntero que pueda contener la dirección de la variable de gestión de un archivo, sólo debe tenerse en cuenta que el tipo base de ese puntero debe ser del tipo FILE.

DECLARACION DEL PUNTERO A LA VARIABLE DE GESTION DEL ARCHIVO

Ejemplificaremos la declaración con el siguiente bloque de programa:

```
#include <stdio.h>

struct registro
{
    int leg;
    char nombre [25];
    float promedio;
};

main()
{
    registro reg;
    FILE *arch; /*arch es la variable puntero a FILE*/
}
```

Tener en cuenta, que no se declara un archivo, sino un puntero para guardar la dirección de la variable de gestión de un archivo. Todas las funciones para acceder y manejar el contenido de un archivo, requieren que se pase como parámetro el puntero a la variable de gestión, y luego cada función accede al archivo en disco usando la información propia de ella.

APERTURA DEL ARCHIVO

La apertura de un archivo se hace a través de la función **fopen** que se encarga de crear la variable de gestión y abrir el canal. La función **fopen** recibe dos parámetros de tipo cadena de caracteres. El primero es el nombre que tiene el archivo en disco (puede incluir la unidad de disco y el camino de directorio). El segundo parámetro es un código de caracteres que indica la forma de abrir el archivo:

Para comenzar a trabajar con el archivo, se debe iniciar el proceso de apertura. Esto se hace mediante la función **fopen**.

La sintaxis de la misma es:

<nombre lógico de archivo> = fopen (nombre físico del archivo, modo de apertura);

- *Nombre lógico del archivo:* es el puntero a la estructura FILE cuando se declara el archivo
- *Nombre físico del archivo:* es el que aparecerá en el listado del directorio del disco.
- *Modo de apertura:* es la forma de trabajo con el archivo.



El modo de apertura es la forma de trabajo con el archivo. Como podemos escribir, leer, modificar, eliminar, los elementos del archivo, el modo de apertura nos indicara solo dos formas: abrir el archivo para lectura o abrirlo para escritura (también para agregar datos en el archivo).

Un ejemplo de apertura es el siguiente:

```
arch = fopen ("datos.dat", "r+b");
```

donde el segundo parámetro es el modo de apertura, que pueden ser:

- *rb*: Sólo lectura. El archivo debe existir, caso contrario retorna NULL
- *wb*: Solo escritura. Si no existe el archivo lo crea, caso contrario borra su contenido y lo abre vacío.
- *ab*: Sólo escritura. Si no existe el archivo lo crea, caso contrario no lo destruye. Los registros que se graban lo hacen al final.
- *r+b*: Lectura y escritura. El archivo debe existir previamente.
- *w+b*: Lectura y escritura. Si no existe el archivo lo crea, caso contrario lo borra y lo abre vacío.
- *a+b*: Lectura y escritura. Si no existe el archivo lo crea, caso contrario no lo borra. Añade al final.

CIERRE DE ARCHIVOS.

Esta operación equivale a destruir la variable de gestión asignada al archivo, guardar en el archivo en disco la información que todavía estuviera en memoria y cerrar el canal de comunicación con el disco. La función para realizar esto es ***fclose()*** la cual toma como parámetro el puntero a la variable de gestión del archivo que se quiere cerrar.

Ej: *fclose(arch);*

ESCRITURA DE UN REGISTRO EN UN ARCHIVO.

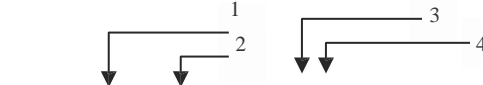
Escribir un registro en un archivo equivale a grabar dicho registro en el archivo, el cual debe estar abierto en un modo que permita la escritura. El registro se grabará en la posición que indique en ese momento el puntero de registro activo del archivo, luego ese puntero se incrementa en uno automáticamente.

La función usada para grabar un registro es ***fwrite()*** que recibe 4 parámetros.

Ejemplo:

```
main()
{
    FILE *m;
    registro r;
    m=fopen ("maestro.dat", "ab");
    if (m == NULL)
    {
        printf ("no se pudo abrir el archivo");
    }
}
```

```
else
{
    scanf ("%d",&r.legajo);
    scanf ("%s",&r.nombre);
    scanf ("%f",&r.promedio);

    
    fwrite (&r, sizeof (alumno), 1, m);

    fclose (m);
}
}
```

Parámetro 1: dirección de la variable cuyo contenido se quiere grabar en el archivo (& indica que se toma la dirección de la variable).

Parámetro 2: tamaño en bytes de la variable que se quiere grabar.

Parámetro 3: cantidad de veces que el valor de la variable será grabado en el archivo.

Parámetro 4: puntero a la variable de gestión del archivo en donde se hará la grabación.

LECTURA DE UN REGISTRO DE UN ARCHIVO.

Leer un registro es tomar una copia del registro señalado por el puntero de registro activo y llevarlo a una variable de memoria. Luego, el puntero de registro activo se incrementa en uno automáticamente. Para hacer una lectura se usa la función **fread()** y sus parámetros son análogos a los de *fwrite()*.

Es decir, que esta función lee de un archivo N bloques de bytes y los almacena en un buffer. El número de bytes de cada bloque (tamaño) se pasa como parámetro al igual que el número de bloques y la dirección del buffer donde se almacena.

DETERMINACION DE FIN DE ARCHIVO.

La función **feof()** recibe como parámetro el puntero a la variable de gestión de un archivo y retorna un uno si el puntero de registro activo es igual al tamaño del archivo o devuelve un cero en caso contrario.

REPOSICIONAMIENTO DE REGISTRO ACTIVO – ACCESO DIRECTO

Es imprescindible que el programador pueda acceder a cierto y determinado registro dentro de un archivo, y para ello es necesario poder cambiar el valor del puntero de registro activo.

La función **fseek** devuelve un valor de tipo int; si se realizó el posicionamiento, la función devuelve cero (0), caso contrario devuelve uno (1).



Esta función recibe 3 parámetros:

- el primero es el puntero al buffer del archivo que se quiere acceder,
- el segundo es el número del byte que se quiere acceder dentro del archivo, a partir de la posición indicada por el tercer parámetro.
- el tercer parámetro puede ser 0, 1 ó 2. Estos valores, se representan con las siguientes constantes y a continuación se detalla su significado:

Constante	Valor	Significado
SEEK_SET	0	Reposiciona comenzando desde el principio del archivo.
SEEK_CUR	1	Reposiciona comenzando desde la posición actual del puntero de
SEEK_END	2	Reposiciona comenzando desde el final del archivo

fseek , a través del segundo parámetro, salta a la dirección de un byte determinado. Para realizar este cálculo se multiplica el número relativo del registro que se quiere acceder por el tamaño en bytes de un registro del archivo (que se calcula con *sizeof*).

Por ejemplo, si se desea acceder al tercer registro comenzando desde cero, la instrucción sería:

fseek (*m*, $2 * \text{sizeof}(\text{alumno})$, *SEEK_SET*)

OBTENCION DEL VALOR DEL PUNTERO DE REGISTRO ACTIVO.

Para verificar cuál es el valor del puntero del registro activo se usa la función ***ftell*** que toma como parámetro un puntero al buffer de un archivo, y retorna un valor de tipo long que indica la posición actual del puntero de registro activo desde el principio del archivo. Esta función retorna el valor -1L (o sea, el valor -1 en formato long) si se produce algún error al intentar determinar el valor del puntero.

REINICIALIZACION DE UN ARCHIVO

Cuando se desea volver a procesar un archivo, es necesario reubicar el puntero de registro activo al principio del archivo. Esta acción puede hacerse de varias formas:

- a) cerrar el archivo y volver a abrirlo
- b) usar *fseek* indicando que se posicione en el byte cero desde el principio del archivo
- c) usar la función ***rewind*** que es análoga a llamar a *fseek*.



5.4. TRATAMIENTO DE ARCHIVOS: ALTAS, BAJA LÓGICA, BAJA FÍSICA, MODIFICACIONES, CONSULTAS, LISTADOS; REPRESENTACIÓN DE ARCHIVOS EN LENGUAJE C.

ALTA ARCHIVOS BINARIOS

Una operación de alta significa ingresar valores en un archivo, grabarlos en un soporte magnético. Este tipo de archivos puede ser utilizado para guardar tipo de datos definidos por el programa o registros definidos por el programados

Archivo de float

Ingresa N valores reales y los graba en un archivo llamado "notas.dat"

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

```
main (void)
```

```
{
```

```
    FILE *arch;
```

```
    int n,i;
```

```
    float nota;
```

```
    /*Alta*/
```

```
    printf("Ingrese la cantidad de empleados: ");
```

```
    scanf("%d",&n);
```

```
    arch=fopen("notas.dat","w+b");
```

```
    printf("\n\nIngrese lasn nota:\n");
```

```
    for (i=0;i<n;i++)
```

```
    {
```

```
        printf("Nota: ");
```

```
        scanf("%f",&nota);
```

```
        fwrite(&nota,sizeof(float), 1,arch);
```

```
    }
```

```
    fclose(arch);
```

```
}
```

Archivo Con Registros

Ingresa N registros, cuya configuración lógica es:

- Código
- Descripción
- Precio

Los mismo se graban en un archivo llamado "articulo.dat"



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

struct registro
{
    int codigo;
    char descri[30];
    float precio;
    bool borrado;
};

main (void)
{
    registro reg;
    FILE *arch;
    int n,i;

    /*Alta*/
    printf("Ingrese la cantidad de articulos: ");
    scanf("%d",&n);

    arch=fopen("articulos.dat","w+b");

    printf("\n\nIngrese los datos del registro\n");
    for (i=0;i<n;i++)
    {

        printf("\nCodigo  : ");
        scanf("%d",&reg.codigo);
        _flushall();
        printf("Descripcion: ");
        gets(reg.descri);
        printf("Precio: ");
        scanf("%f",&reg.precio);
        reg.borrado=false;
        fwrite(&reg,sizeof(registro),1,arch);
    }
    fclose(arch);
}
```

LISTADO

Es una operación que permite mostrar todos los datos grabados en el archivo.

Archivo de float

```
#include <stdio.h>

main (void)
```



```
{  
    FILE *arch;  
    float nota;  
  
    /*Listado*/  
  
    arch=fopen("notas.dat","r+b");  
  
    printf("Notas almacenadas:\n");  
    fread(&nota,sizeof(float),1,arch);  
    while (!feof(arch))  
    {  
        printf("Nota: %.2f\n",nota);  
        fread(&nota,sizeof(float),1,arch);  
    }  
    fclose(arch);  
}
```

Archivo Con Registros

```
#include <stdio.h>  
#include <string.h>  
  
struct registro  
{  
    int codigo;  
    char descri[30];  
    float precio;  
    bool borrado;  
};  
  
main (void)  
{  
    registro reg;  
    FILE *arch;  
  
    /*Listado*/  
    arch=fopen("articulos.dat","r+b");  
  
    printf("\n\nDatos grabados en el archivo\n");  
    fread(&reg,sizeof(registro),1,arch);  
    while (!feof(arch))  
    {  
  
        printf("\nCodigo:%d\n",reg.codigo);  
        printf("Descripcion: ");
```



```
    puts(reg.descri);  
    printf("Precio: %.2f\n", reg.precio);  
    fread(&reg, sizeof(registro), 1, arch);  
}  
fclose(arch);  
}
```

CONSULTA

Es una operación que permite mostrar todos los datos grabados en el archivo que cumplan una determinada condición.

Archivo de float

```
/*Se muestran las notas aprobadas (nota>6)*/
```

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

```
main (void)
```

```
{
```

```
    FILE *arch;
```

```
    float nota;
```

```
/*Listado*/
```

```
arch=fopen("Notas.dat", "r+b");
```

```
printf("Notas almacenadas:\n");
```

```
fread(&nota, sizeof(float), 1, arch);
```

```
while (!feof(arch))
```

```
{
```

```
    if (nota>6)
```

```
    {
```

```
        printf("Nota: %.2f\n", nota);
```

```
    }
```

```
        fread(&nota, sizeof(float), 1, arch);
```

```
    }
```

```
fclose(arch);
```

```
}
```

Archivo Con Registros

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

```
#include <string.h>
```

```
struct registro
```

```
{
```



```
int codigo;  
char descri[30];  
float precio;  
bool borrado;  
};  
  
main (void)  
{  
    registro reg;  
    FILE *arch;  
  
    /*Listado*/  
    arch=fopen("articulos.dat", "r+b");  
  
    printf("Datos grabados en el archivo\n");  
    fread(&reg, sizeof(registro), 1, arch);  
    while (!feof(arch))  
    {  
        if (reg.precio > 100)  
        {  
            printf("\nCodigo: %d\n", reg.codigo);  
            printf("Descripcion: ");  
            puts(reg.descri);  
            printf("Precio: %.2f\n", reg.precio);  
        }  
        fread(&reg, sizeof(registro), 1, arch);  
    }  
    fclose(arch);  
}
```