Archivos

Conceptos sobre archivos

Todos los programas que se ejecutaron hasta el momento requerían de ingreso de datos para realizar determinadas tareas y luego mostrar resultados. Tales programas tienen un gran defecto y es que los datos que se cargan desaparecen cuando el programa termina, por lo tanto cada vez que se corra el programa hay que cargar los datos nuevamente.

Esto ocurre debido a que los datos ingresados quedan guardados en memoria y al terminar el programa todos los espacios de memoria reservados son devueltos al sistema operativo perdiendo todos los datos que se habían cargado.

Para solucionar el problema debería existir una forma de almacenar los datos en forma permanente, por ejemplo en un medio magnético como el disco rígido o en algún otro tipo de dispositivo. Los datos que se ingresan se deberán guardar en un archivo que se almacena en el disco o medio disponible, para contar con la información en cualquier momento.

De tal forma, si por ejemplo trabajamos con un programa de agenda en la cual se cargan datos de personas, cada vez que se ejecute el programa se irá a leer el archivo que contiene los datos de las personas, en lugar de tener que ingresarlos nuevamente.

Desde D.O.S. los nombres de archivo tienen el siguiente formato: **nombre.extensión** donde:
nombre /* es el nombre del archivo que puede tener un máximo de 8 caracteres */
extensión /* define un tipo de archivo en particular y puede tener un máximo de 3 caracteres */

Algunas de las extensiones usadas son las siguientes:

EXE	programa ejecutable
COM	programa ejecutable (el tamaño no excede los 64kb)
BAT	archivo de procesamiento por lotes
TXT	archivo de text
DOC	archivo para ser abierto por Microsoft Word
BIN	archivo binario
DAT	archivo de datos
INI	archivo usado por Windows para las condiciones iniciales de los programas
MP3	archivos de música
WAV	archivos de música

Básicamente existen 2 tipos de archivos que son los **de texto** y **binarios**. Esta clasificación hace referencia a la forma en la cual se interpretan los datos, se debe tener en claro que la única forma de guardar datos en el disco es en formato binario. Todos los archivos tienen al final una marca que indica el final del archivo. Dicha marca se conoce como EOF (End Of File).

Archivos de texto

Un archivo de texto contiene toda su información guardada en binario pero se interpreta como texto. Absolutamente todo lo que contiene debe ser interpretado como texto, ya que cuando se escribe el archivo, los datos son enviados como caracteres.

Cuando se dice que la información está guardada en formato texto se está haciendo referencia a cómo es la forma en la que hay que entender el dato, ya que sobre el disco los datos son guardados en forma de secuencia de “UNOS” y “CEROS” es decir según el sistema binario.

Supongamos que se desea guardar en un archivo de texto la siguiente cadena “Ana 12”. Lo que se guarda en el archivo son caracteres es decir la **‘A’**, la **‘n’**, la **‘a’**, el espacio **‘ ’**, el carácter **‘1’** y el carácter **‘2’**, entonces si analizamos el equivalente ASCII de los caracteres, en el archivo queda:

Carácter	A	n	a		1	2
ASCII	65	110	97	0	49	50
Binario	00100001	01011110	01100001	00100000	00110001	00110010

Lo que figura en la fila que dice binario es lo que queda en el disco rígido, observe que de acuerdo a como se tomen los datos se está cambiando la información.

Si por ejemplo se quiere guardar en el archivo de texto el número 4567 se va a guardar como caracteres, es decir que en el disco queda:

Carácter	4	5	6	7
ASCII	52	53	54	55
Binario	00110100	00110101	00110110	00110111

Observe que para guardar un número como texto la cantidad de bytes a usar igual a la cantidad de dígitos, que en este caso en particular estamos usando 4 bytes. En el caso de usar un tipo de variable int solo se necesitan 2 bytes.

Archivos binarios

En un archivo binario se guardan datos con distinto formato, es decir se pueden guardar caracteres mezclados con enteros y flotantes. Si bien todos los datos terminan escritos en el disco en sistema binario, la interpretación de los datos guardados cambia.

Si tomamos el último ejemplo dado para archivos de texto y queremos escribir en el archivo el número 4567, en el programa va a estar definido como un entero y por lo tanto se escribirán 2 bytes al archivo. El número quedaría de la siguiente forma:

Número	4567
Binario	00010001 11010111

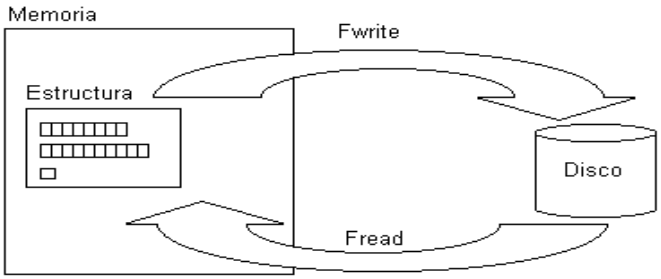
Normalmente el hecho de trabajar con archivos binarios está asociado con el uso de estructuras, dado que la forma más simple de trabajar es cargar una estructura para luego escribirla en el archivo y de esa forma guardar los datos. Si bien se puede trabajar con archivos binarios y no usar estructuras, el hecho de utilizarlas permite tener un programa mejor armado y más consistente.

Trabajo sobre archivos En C, todas las operaciones que se realizan sobre archivos son hechas a través de funciones. Básicamente existen 2 categorías de funciones para trabajar con archivos y son las que usan “buffer” y las que acceden directamente al archivo. Durante el transcurso del apunte se usarán solamente la primera categoría, o sea aquellas que usan “buffer”. El hecho de utilizar un buffer significa que no se tiene acceso directo al archivo y que cualquier operación que se desee realizar (lectura o escritura) va a ser hecha sobre el buffer.

Cuando el buffer se llena o se vacía se actualizan los datos desde y hacia el archivo. Algunas de las funciones usadas para trabajar con archivos son:

fopen( ); / fclose( ); / fcloseall( ); / fread( ); / fwrite( ); / feof( ); / fseek( ); / ftell( ); / rewind( );

Se debe tener en cuenta que al usar las funciones sobre archivos estamos trabajando con un intermediario que accede al archivo. Por lo tanto todos los datos que van al archivo (escritura) y que vienen (lectura) se encuentran en memoria y fread () y fwrite() toman o dejan esos datos en el archivo. Gráficamente podemos verlo de la siguiente forma.



La estructura FILE

Para trabajar con archivos en C, las funciones utilizan un **puntero** a la **estructura FILE**. Dicha estructura se encuentra definida en el archivo **<stdio.h>** y se detalla a continuación.

```
typedef struct {
    int             level;          /* full/empty level of buffer */
    unsigned        flags;          /* File status flags */
    char fd;        fd;             /* File descriptor */
    unsigned char   hold;           /* Ungetc char if no buffer */
    int             bsize;          /* Buffer size */
    unsigned char   _FAR *buffer;   /* Data transfer buffer */
    unsigned char   _FAR *curp;     /* Current active pointer */
    unsigned        istemp;         /* Temporary file indicator */
    short           token;          /* Used for validity checking */
}FILE;
```

Entonces para el manejo de archivos es indispensable definir un puntero a la estructura FILE, como por ejemplo:

```
FILE * parch;     /* donde parch es el puntero a la estructura FILE */
```

Apertura y cierre de un archivo

Apertura del archivo

Cada vez que se necesite trabajar con un archivo, lo primero que se debe hacer es abrirlo. Si no se realiza esto no se puede leer ni escribir en el mismo. La función que me permite la apertura del archivo es **fopen( )**. El formato de la función fopen( ) es el siguiente:

```
FILE * fopen (const char *NombreDeArchivo, const char *Modo);
```

donde:

NombreDeArchivo
/* Es una cadena de caracteres que representa el archivo, es decir se pone la ruta y el nombre del archivo */

Modo
/* Es una cadena de caracteres que determina el modo en el que será abierto el archivo */

Los modos en los que se puede abrir un archivo están detallados en la siguiente tabla:

Modo	Detalle
r	Abre un archivo de texto para operaciones de lectura
w	Abre un archivo de texto para operaciones de escritura
a	Abre un archivo de texto para añadir datos
rb	Abre un archivo de binario para operaciones de lectura *
wb	Abre un archivo de binario para operaciones de escritura // si no existe lo crea *
ab	Abre un archivo de binario para añadir datos *
r+b	Abre un archivo de binario para operaciones de lectura / escritura *
w+b	Abre un archivo de binario para operaciones de lectura / escritura
a+b	Abre un archivo de binario para operaciones de lectura / escritura
r+	Abre un archivo de texto para operaciones de lectura / escritura
w+	Abre un archivo de texto para operaciones de lectura / escritura

Modo escritura (w)
Si se abre un archivo para operaciones de escritura (w, wb, w+b, w+) y el archivo no existe se creará, pero si existe todos los datos del archivo serán borrados.
Si el archivo que se desea abrir tiene atributo de solo lectura, ó el disco está lleno, ó el disco tiene un problema en la superficie, etc., la función **fopen( )** devuelve error.
Cada vez que se abra un archivo en este modo, el indicador de posición se encuentra al comienzo del archivo. Si se abre un archivo en el modo w ó wb, solamente se puede escribir. Si se intenta leer datos del archivo no va a aparecer ningún error, simplemente lo que aparezca como dato leído no va a reflejar la realidad.

Modo lectura (r)
Si se abre un archivo para operaciones de lectura (r, rb, r+b, r+), si el archivo no existe la función **fopen( )** devuelve error. Cada vez que se abra un archivo en este modo, el indicador de posición se encuentra al comienzo del archivo.
Si se abre un archivo en el modo r ó rb, solamente se pueden realizar lecturas. No tiene ningún efecto realizar operaciones de escritura sobre al archivo, es decir, por mas que se intente escribir sobre el archivo no se va a poder.

Modo append (a)
Si se abre un archivo para agregar datos (a, ab, a+b, a+) y el archivo no existe se creará, caso contrario el indicador de posición del archivo queda posicionado al final del mismo de forma de poder agregar datos.

Cada vez que se agregan datos se hace al final del archivo. Con el modo append no tienen efecto las operaciones de desplazamiento a través del archivo.

Valor retornado: Si el archivo es abierto exitosamente, la función devuelve un **puntero** a la **estructura FILE** asociada al archivo. En caso de detectarse un error devuelve **NULL**.

Notas:

- Se pueden abrir varios archivos al mismo tiempo siempre y cuando exista por lo menos un **puntero a FILE** para cada uno.
- La cantidad de archivos que se pueden abrir al mismo tiempo depende del sistema operativo y de la variable de sistema FILES que se encuentra en el archivo **config.sys**. Tenga en cuenta que cuando se ejecuta el programa hay abiertos 5 archivos que son propios de C.
- **No se debe modificar el valor del puntero** devuelto por **fopen( )** (perdería el archivo).

Ejemplo:

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

int main () {
    FILE * parch = fopen ("d:\\banco.dat", "rb" );
    if(parch != NULL){//Si el archivo se abrió correctamente podemos trabajar
        /// acá incluimos nuestro código
        fclose (parch);
    }
}
```

En el ejemplo se utiliza un if para detectar la correcta apertura del archivo. Recuerde que cuando la función fopen() retorna NULL significa que se ha fallado en abrir el archivo en el modo solicitado.

Para crear un archivo por primera vez se debe usar el modo **w**, pero primero nos debemos asegurar que el archivo no exista, ya que en ese caso el contenido del archivo se borra. Para contemplar esta situación se modifica levemente el código del ejemplo.

```c
int main () {
    FILE *parch = fopen ( "d:\\banco.dat" , "rb" ); //Se abre en modo lectura.
    if (parch == NULL) // Si el archivo no pudo ser abierto
        parch = fopen ( "d:\\banco.dat" , "wb" ); // se abre en modo escritura.
```

```
        if (( parch != NULL) { //Si el modo anterior dio error, el archivo no
existe, por lo tanto se crea
           // puedo trabajar con un archivo nuevo
           fclose (parch);
        }
        else{
            printf("ERROR de datos - El archivo no pudo ser abierto");
        }
    else{
        // trabajo con el archivo existente
         fclose (parch);
    }
}
```

La idea es abrir un archivo para leer, en el caso de que exista se trabaja normalmente, pero si no existe lo abre el segundo fopen(). De esta forma nos evitamos borrar un archivo que existe y tiene datos.

En el caso de querer ingresar el nombre del archivo por teclado, el programa se modifica de la siguiente manera:

```
int main () {
    FILE *parch=NULL;
    char nombre[20];
    printf("\n Ingrese el nombre y ruta del archivo que desea abrir: ");
    fflush(stdin);
    gets(nombre);
    parch = fopen(nombre, "rb");
    if (parch == NULL) // Si el archivo no pudo ser abierto
        parch = fopen ( nombre , "wb" ); // se abre en modo escritura.
        if (( parch != NULL) {
           // puedo trabajar con un archivo nuevo
           fclose (parch);
        }
        else{
            printf("ERROR de datos - El archivo no pudo ser abierto");
        }
    else{
        // trabajo con el archivo existente
         fclose (parch);
    }
}
```

Cierre del archivo

Todo archivo que se abre debe ser cerrado antes de terminar el programa. **Terminar el programa sin cerrar el o los archivos abiertos puede causar pérdida de datos.**

La función **fclose( )** es la que se encarga de cerrar un archivo. El formato de la función es el siguiente:

```
int fclose ( FILE * parch ); //donde parch es el puntero a la estructura
FILE asociada con el archivo que se desea cerrar.
```

Valor retornado: Si el archivo es cerrado exitosamente se retorna un “0”, en caso contrario se devuelve “-1”

Ejemplo:

```
void main (void) {
    FILE * parch = fopen (" d:\\banco.dat" , "rb" );
    if (( parch != NULL) { //Se abre en modo lectura
        // se trabaja con el archivo abierto
        if (( fclose (parch)) == -1 )
            printf ( "\n No se pudo cerrar el archivo" );
        else
            printf ( "\n El archivo se cerro exitosamente" );
    }
}
```

En el ejemplo se chequea que se haya cerrado correctamente el archivo. En el caso de abrir más de un archivo, antes de terminar el programa se deben cerrar.

```
int main () {
    FILE *parch1 = fopen ( "D:\\banco.dat" , "rb" );
    FILE *parch2 = fopen ( "D:\\result.dat", "wb" );
    if (parch1 != NULL) && (parch2 != NULL){
        // puedo trabajar con ambos archivos
        fclose (parch1);
        fclose (parch2);
    }
    else{
            printf("ERROR de datos");
    }
```

```
}
```

Existe una función llamada fcloseall() que cierra todos los archivos que se encuentran abiertos. El formato de la función es:

```
int fcloseall();
```

Si la operación es exitosa retorna la cantidad de archivos que se cerraron, en caso contrario devuelve “-1 ”

Si modificamos el ejemplo anterior, en lugar de usar la función fclose ( ) para cerrar los archivos

```
fclose (parch1); //Se cierra el archivo banco.dat
fclose (parch2); //Se cierra el archivo result.dat
```

Podríamos usar la función fcloseall( ):

```
fcloseall( ); //Cierra los 2 archivos abiertos: banco.dat y result.dat
```

Escritura de un archivo

Una vez que el archivo se encuentra abierto se puede empezar a trabajar para leer o escribir. La función utilizada para realizar la escritura es **fwrite( )**. Esta función sirve para escribir archivos de texto o binarios. El prototipo de la función es el siguiente:

```
int fwrite ( void * origen , size_t tamaño , size_t cantidad , FILE *arch)
                                                    /* Nota: size_t es un unsigned int definido en stdio.h */
```

donde:

origen	Es un puntero al lugar desde dónde se obtienen los datos para escribir en el archivo
tamaño	Es el tamaño en bytes del dato que se va a escribir
cantidad	Es la cantidad de datos de longitud **tamaño** que se van a escribir
arch	Es el puntero a FILE asociado al archivo

Valor retornado: Devuelve el número de datos escritos (cantidad). Si el valor retornado es menor al que se indicó por medio de la variable cantidad, significa que hubo un error en la escritura.

La función **fwrite( )** toma cantidad de datos de longitud **tamaño** desde la dirección **origen** y los escribe en el archivo asociado al puntero **arch** comenzando desde la posición actual del indicador de posición del archivo. Una vez que se completó la operación de escritura el indicador de posición es actualizado (queda apuntando al lugar donde se puede escribir el próximo dato).

Ejemplo: Si queremos escribir en un archivo de texto

```
int main () {
    FILE * parch = fopen ( "D:\\prueba.txt" , "w" );
    char texto [ ] = "Prueba de escritura";
    int cant, long;
    if (( parch != NULL) { //Se abre en modo escritura
       longi = strlen (texto);
       cant = fwrite (texto , sizeof (char) , longi , parch) ; //Se escribe
al archivo
       if (cant < longi)
          printf ( "\n Error al escribir el archivo" );
       else
          printf ( "\n Se escribieron %d caracteres", cant);
       fclose (parch); //Se cierra el archivo
    }
    else{
        printf("ERROR de datos");
    }

    return 0;
}
```

Si se trata de un archivo binario el programa será el siguiente:

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

#define ARCH "c:\\bin.dat"
#define ESC 27

struct a {
    char nombre [10];
    int edad;
};
```

```c
int main () {
    FILE * bin = fopen ( ARCH , "wb" );
    struct a pers; /* una variable (pers) del tipo de la estructura a */
    printf ( "\n Se va a generar por primera vez el archivo \n" );
    if (( bin != NULL) {
        do {
            printf ( "\n Ingrese el nombre: ");
            fflush (stdin);
            gets (pers.nombre);
            printf ("Ingrese la edad: ");
            scanf ( "%d", &pers.edad);
            fflush (stdin);
            fwrite (&pers, sizeof (pers), 1, bin);
            printf ( "\n Presione ESC para terminar" );
        } while ((getche () ) != ESC );
    }
    else{
        printf("ERROR de datos");
    }
    fclose (bin);
    return 0;
}
```

Antes de escribir el archivo se debe cargar el dato que se desea guardar, en nuestro caso debemos cargar la estructura. Una vez que se cargaron todos los campos se llama a **fwrite( )** y se le pasa la dirección de comienzo de la estructura (&pers), el tamaño en bytes de la estructura (se puede escribir directamente o usar **sizeof( )**), la cantidad de estructuras que se van a escribir (se cargó solo una por lo tanto se escribe una) y finalmente el puntero que hace referencia al archivo **(bin)**. En el caso que no se desee escribir la estructura entera se deberán hacer 2 fwrite(), uno para la edad y el otro para el nombre.

Repasando:
*El operador sizeof( ) me permite obtener la cantidad de bytes de un determinado dato. Se puede usar con nombres de variables o con tipos de dato (**char, int, float,** etc.).*

Por ejemplo:
int i;
char c;
sizeof (char); sizeof (c); (= 1 byte) / son expresiones equivalentes */*
sizeof (int); sizeof (i); (= 2 byte) / son expresiones equivalentes */*

Lectura de un archivo

Para realizar la lectura de un archivo se utiliza la función **fread( )** que tiene el siguiente prototipo

```
int fread ( void * destino , size_t tamaño , size_t cantidad , FILE *arch )
```

donde:

destino	Es un puntero al lugar donde se va a dejar el dato leído con **fread()**
tamaño	Es el tamaño en bytes del dato a leer
cantidad	Es la cantidad de elementos de longitud **tamaño** que se van a leer
arch	Es el **puntero** a la **estructura FILE** asociada al archivo desde el que se va a leer

Valor retornado:

Devuelve el número de datos leídos **(cantidad)**. Si el valor retornado es menor al que se indicó por medio de la variable **cantidad**, significa que hubo un error en la lectura o que se llegó al final de archivo.

La función **fread( )** lee desde el archivo referenciado por **arch** a partir de la posición actual del indicador de posición, **cantidad** de elementos de longitud **tamaño** y deja los elementos leídos en la dirección de memoria indicada por **destino**. Una vez que se completó la operación de lectura se actualiza automáticamente el indicador de posición del archivo. A diferencia de lo que ocurre en la escritura, se debe verificar que se realice la lectura mientras no se haya llegado al final del archivo. Esta operación se realiza por medio de la función **feof( )**.
Ejemplo:

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

#define ARCH "c:\\bin.dat"
struct a {
    char nombre [10];
    int edad;
};
int main () {
    FILE * bin = fopen (ARCH , "rb" );
    struct a pers;
    int cant;
    if ( bin != NULL) {
        while ( !feof (bin)) {
            cant = fread ( &pers, sizeof (pers), 1, bin);
```

```
            if (cant != 1) {
                if ( !feof (bin)) {
                    printf ("\n %s \t %d ", pers.nombre, pers.edad);
                }
            }
        }
        fclose (bin);
        getch ( );
    }
    else{
        printf("ERROR de datos");
    }
    return 0;
}
```

Después de hacer la lectura se debe verificar que se haya leído la cantidad de datos que se indicó. Ocurre que cuando no se lee la cantidad de datos indicada puede haberse alcanzado el final de archivo o se pudo haber producido un error. Es por esto que cuando se entra al **if** que verifica la **cantidad**, debemos preguntar si se llegó al final del archivo.

Nos queda explicar el uso de la función: **feof ( )**

La función **feof( )** determina si se ha llegado al final del archivo, el prototipo es:

```
int feof ( FILE * arch )
```

donde: arch es el puntero a la estructura FILE asociada con el archivo.

El **valor retornado** por la función será:

== 0 /* (igual a cero) si no se llegó al final del archivo */
!= 0 /* (distinto de cero) si se llegó al final del archivo */

De todas maneras, la forma más usual de referirse a la función feof ( ) es mediante expresiones del tipo:

```
while ( ! feof (arch)) {
// mientras (sea distinto de fin de archivo (EOF))
/* donde arch es el puntero a la estructura FILE */
}
```

O algo así:

```
while (fread(&aux, sizeof(struct Algo) , 1 , archi) > 0 ) {
// si es mayor a cero es una lectura correcta, si es igual a
// cero significa que la lectura fue errónea o se ha llegado
```

```
// a la marca de fin de archivo (EOF).
}
```