Archivos

Programación I – Laboratorio I.

Tecnicatura Superior en Programación.

UTN-FRA

Autores: *Pablo Gil*

Hector Farina

Revisores: *Ing. Ernesto Gigliotti*

Lic. Mauricio Dávila

Versión : 1

Índice de contenido

1 Archivos

Todos los programas que se ejecutaron hasta el momento requerían de ingreso de datos para realizar determinadas tares 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 como el disco rígido. Los datos que se ingresan se deberán guardar en un archivo que se almacena en el disco de forma de disponer de la información en cualquier momento.

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

1.1 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 esta guardada en formato texto se esta haciendo referencia a como 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 caracter 1 y el caracter 2 , entonces si analizamos el equivalente ASCII de los caracteres , en el archivo queda:

Caracter	A	n	a	1	2
ASCII	33	110	97	49	50
Binario	0b00100001	0b01101110	0b01100001	0b00110001	0b00110010

Lo que figura en la fila que dice binario es lo que queda en el disco rígido.

Si por ejemplo se quiere guardar en el archivo de texto el número **45678** se va a guardar como caracteres , es decir que en el disco queda:

Caracter	4	5	6	7	8
ASCII	52	53	54	55	56
Binario	0b00110100	0b00110101	0b00110110	0b00110111	0b00111000

Se observa que para guardar un número como texto la cantidad de bytes a usar es igual a la cantidad de dígitos, que en este caso en particular estamos usando 5 bytes. En el caso de usar un tipo de variable int solo se necesitan 2 bytes pero para ello deberíamos hacerlo en un archivo binario.

1.2 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 45678 , 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:

```
0b1011001001101110
```

Normalmente el hecho de trabajar con archivos binarios esta asociado con el uso de estructuras, dado que la forma mas 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 mas consistente.

1.3 Trabajo sobre archivos

En C, todas las operaciones que se realizan sobre archivos son hechas a travez 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.

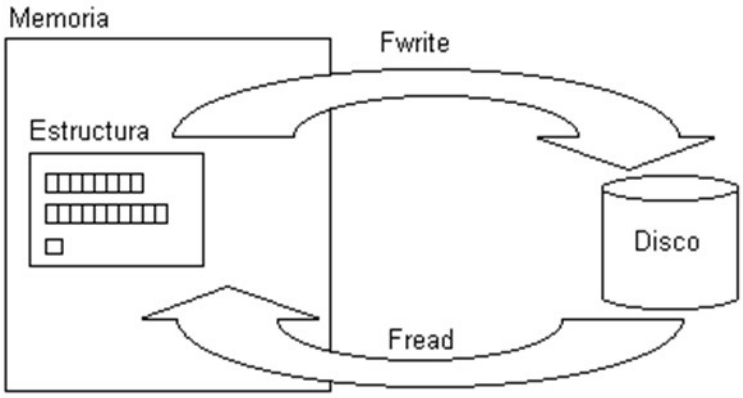
Durante el transcurso del apunte se usarán solamente la primer categoría , o sea aquellas que usan "buffer". El hecho de utilizar un buffer significa que no se tiene acceso directo al archivo y que cualquier operación que se desee realizar (lectura o escritura) va a ser hecha sobre el buffer. Cuando 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:

```
FILE *fopen(char *nombre,char *modo);
int fclose(FILE *fp);
int fcloseall(void);
int fread(void *memoria,int num,int cont,FILE *fp);
int fwrite(void *memoria,int num,int cont,FILE *fp);
int fclose(FILE *f);
int feof(FILE *f);
int ferror(FILE *fp);
void rewind(FILE *fp);
int fseek(FILE *fp,long num,int origen);
long ftell(FILE *f);
int getc(FILE *fp);
int putc(int ch,FILE *fp);
char* fgets(char *str,int n,FILE *fp);
int fputs(const char *str,FILE *fp);
int fscanf(FILE *fp,const char *formato[,dirección,...]);
int fprintf(FILE *fp,const char *formato[,argumento,...]);
```

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, por lo tanto 'fread()' y 'fwrite()' toman o dejan esos datos en el archivo.

Gráficamente podemos verlo de la siguiente forma.



1.3.1 El tipo de dato 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;            /* fill/empty level of buffer */
      unsigned          flags;            /* File status flags          */
      char              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.

1.4 Apertura de un 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 permite la apertura del archivo es '**fopen()**' y su formato es el siguiente:

```
FILE * fopen (const char *Nombre_de_archivo , const char *Modo)
```

Donde:

Nombre_de_archivo: 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 estan 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 binario para operaciones de lectura.
wb	Abre un archivo binario para operaciones de escritura.
ab	Abre un archivo binario para añadir datos.
r+b	Abre un archivo binario para operaciones de lectura escritura.
w+b	Abre un archivo binario para operaciones de lectura escritura.
a+b	Abre un archivo 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 o el disco esta lleno, 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 o wb solamente se puede escribir. Si se intenta leer datos del archivo no va a aparecer ningún error, simplemente lo que aparezca como dato leido 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 o 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 traves 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

A tener en cuenta:

- 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.
- No se debe modificar el valor del puntero devuelto por '**fopen()**'.

Veamos un ejemplo:

```
#include <stdio.h>
void main (void)
{
	FILE *parch;
	if((parch=fopen("banco.dat","rb"))==NULL)
	{
		printf("\nEl archivo no puede ser abierto");
		exit (1);
	}
	fclose(parch);
}
```

En el ejemplo se utiliza un if para detectar la correcta apertura del archivo. Se debe recordar 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.

```
void main (void)
{
	FILE *parch;
	//Se abre en modo lectura
	if((parch=fopen("banco.dat","rb"))==NULL)
	{
		//Si el modo anterior dio error el archivo
		if((parch=fopen("banco.dat","wb"))==NULL)
		{
			printf("\nEl archivo no puede ser abierto");
			exit (1);
		}
		//no existe, por lo tanto se crea
		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:

```
void main (void)
{
	FILE *parch;
	char nombre[20];
	printf("\nIngrese el nombre y ruta del archivo que desea abrir: ");
	gets(nombre);
	if((parch=fopen( nombre ,"rb"))==NULL)
	{
		if((parch=fopen( nombre ,"wb"))==NULL)
		{
			printf("\nEl archivo %s no puede ser abierto",nombre);
			exit (1);
		}
		fclose(parch);
	}
}
```

1.4.1 Cierre de un archivo

Todo archivo que se abre debe ser cerrado antes de terminar el programa. El terminar el programa sin cerrar el o los archivos 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;

Veamos un ejemplo:

```
void main (void)
{
	FILE *parch;
	//Se abre en modo lectura
	if((parch=fopen("banco.dat","rb"))==NULL)
	{
		printf("\nEl archivo no puede ser abierto");
		exit (1);
	}
	if((fclose(parch))==-1) //Se cierra el archivo
	{
		printf("\nNo se pudo cerrar el archivo");
	}
	else
	{
		printf("\nEl archivo se cerro exitosamente");
	}
}
```

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

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

```
	int fcloseall ( void );
```

Si la operación es exitosa retorna la cantidad de archivos que se cerraron , en caso contrario devuelve –1.

Si modificamos el ejemplo anterior:

```
void main (void)
{
	FILE *parch1, *parch2;
	if((parch1=fopen("banco.dat","rb"))==NULL) //Se abre en modo lectura
	{
		printf("\nEl archivo no puede ser abierto");
		exit (1);
	}

	if((parch2=fopen("result.dat","wb"))==NULL) //Se abre en modo escritura
	{
		printf("\nEl archivo no puede ser abierto");
		exit (1);
	}

	fcloseall();
}
```

1.5 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);
```

Donde:

- **origen:** Es un puntero al lugar desde donde 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

Nota: size_t es un unsigned int definido en stdio.h

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:

```
FILE *parch;
char texto[ ]="Prueba de escritura";
int cant , longi;
if((parch=fopen("prueba.txt","w"))==NULL) //Se abre en modo escritura
{
	printf("\nEl archivo no puede ser abierto");
	exit (1);
}
longi=strlen (texto );
cant=fwrite ( texto , sizeof ( char ) , longi , parch ); //Se escribe al archivo
if (cant<longi)
	printf("\nError al escribir el archivo");
else
	printf("\nSe escribieron %d caracteres", cant);
fclose(parch);
```

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

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

void main (void)
{
	FILE  *bin;
	struct a pers;

	bin=fopen("bin.dat","wb");
	printf("\nIngrese el nombre: ");
	gets(pers.nombre);
	printf("Ingrese la edad: ");
	scanf("%d",&pers.edad);
	fflush(stdin);
	fwrite(&pers,sizeof(pers),1,bin);
	fclose(bin);
}
```

Antes de escribir el archivo se debe cargar el dato que se desea guardar , en nuestro caso debemos cargar la esctructura. 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.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.

1.6 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 leido 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 leidos (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 leidos 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 de lectura de archivo binario:

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

void main (void)
{
	FILE  *bin;
	struct a pers;
	int cant;

	if ((bin=fopen("bin.dat","rb"))==NULL)
	{
		printf("No se pudo abrir el archivo");
		exit(1);
	}
	while(!feof(bin))
	{
			cant=fread(&pers,sizeof(pers),1,bin);
			if(cant!=1)
			{
				if(feof(bin))
					break;
				else
				{
					error("No leyo el ultimo registro");
					break;
				}
			}
			printf("\n%s\t%d",pers.nombre,pers.edad);
	}
	fclose(bin);
	getch();
}
```

Despues de hacer la lectura se debe verificar que se haya leido 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 llego al final del archivo.

La función feof determina si se ha llegado al final de el 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 es 0 si no se llegó al final del archivo y distinto de cero si se llegó al final del archivo.

1.7 Búsqueda y modificación

En la lectura y escritura de archivos el indicador de posición se actualiza automáticamente , pero existen casos , por ejemplo en las búsquedas y modificaciones sobre archivos , en los cuales se necesita mover el indicador de posición a algún lugar en particular. Para ello se cuenta con 2 funciones que permiten realizar tal operación , ellas son fseek y rewind. Por otra parte se cuenta con una función que permite obtener el lugar donde se encuentra el indicador de posición del archivo , esa función es ftell. A continuación se detallan las 3 funciones.

1.7.1 rewind

Esta función permite llevar el indicador de posición al comienzo del archivo. El prototipo es el siguiente:

```
void  rewind (FILE* arch);
```

La función rewind ubica el indicador de posición del archivo referenciado por el puntero arch al principio y limpia los indicadores de fin de archivo y error que se encuentran el la estructura FILE.

Si se utiliza fread luego de ejecutar rewind, se podra leer el archivo desde el comienzo como cuando se abre con fopen.

1.7.2 fseek

Esta función permite desplazar el indicador de posición del archivo a la posición que se le indique. El prototipo es:

```
int  fseek ( FILE *arch , long desplazamiento , int  origen);
```

Donde:

- **arch:** Puntero a la estructura FILE asociada con el archivo
- **desplazamiento:** Es la cantidad de bytes que se desplazará el indicador de posición a partir de origen
- **origen:** Es una constante que determina el punto de referencia a partir del cuál se realiza el desplazamiento.

Los valores que se le pueden dar a origen figuran en la siguiente tabla. Dichos valores se encuentran definidos en stdio.h:

SEEK_SET	A partir del comienzo del archivo
SEEK_CUR	A partir de la posición actual del archivo
SEEK_END	A partir de el final del archivo

Valor retornado: Si la operación es exitosa devuelve cero , caso contrario retorna un valor distinto de cero.

La función fseek mueve el indicador de posición del archivo desplazamiento bytes a partir de la posición indicada por origen.

Ejemplos:

```
fseek ( ptr , 0L , SEEK_SET );
```

Mueve el indicador de posición al comienzo del archivo. El origen es SEEK_SET que indica el comienzo del archivo y se desplaza 0 bytes , por lo tanto queda al principio. Es aconsejable que cuando se desee llevar el indicador de posición al comienzo del archivo se utilice rewind ya que fseek no limpia el indicador de error ni el de fin de archivo , por lo tanto cuando en determinada situación se use fseek no va a dar los resultados esperados.

```
fseek ( ptr , 0L , SEEK_END );
```

Mueve el indicador de posición al final del archivo. El origen es SEEK_END que indica el final del archivo y a partir de alli se desplaza 0 bytes , por lo tanto esta al final del archivo. Si se desea en algún momento agregar datos , simplemente se debe usar esta función para enviar el indicador de posición al final del archivo.

```
fseek ( ptr , 20L , SEEK_SET );
```

Mueve el indicador de posición 20 bytes a partir del comienzo del archivo.

```
fseek ( ptr , (long) (-1)*sizeof (struct x) , SEEK_CUR );
```

Mueve el indicador de posición una estructura para atras a partir de la posición actual. Normalmente esta forma se utiliza cuando se estan editando datos del archivo. Al realizar una búsqueda se va leyendo cada uno de los datos del archivo por medio de fread, pero cuando encontramos el dato el indicador de posición del archivo esta en el dato siguiente al que queremos modificar , con lo cual al hacer fwrite para escribirlo se modificará otro dato. Por lo tanto antes de escribir se debe mover el indicador de posición una estructura para atrás.

1.7.3 ftell

La función ftell me permite obtener la posición actual del indicador de posición. El prototipo es el siguiente:

```
long  ftell (FILE  * arch );
```

Donde arch es el puntero a la estructura FILE asociada al archivo.

Valor retornado: Si la operación es exitosa devuelve la cantidad de bytes que hay desde el commienzo del archivo hasta el lugar en que se encuentra el indicador de posición del archivo , en caso contrario devuelve –1L (-1 como tipo long).

Ejemplo: Obtener el tamaño de un archivo en bytes:

```
void main (void)
{
	FILE  *bin;
	long int cant;
	if ((bin=fopen("bin.dat","rb"))==NULL)
	{
		printf("No se pudo abrir el archivo");
		exit(1);
	}
	fseek (bin , 0L , SEEK_END ); //Se envía la posición al final del archivo
	cant=ftell (bin);
	printf("\nEl archivo tiene %ld bytes",cant);
	fclose(bin);
	getch();
}
```