5. ARCHIVOS

5.1. INTRODUCCIÓN

Llegados hasta este punto, todos los programas y ejemplos anteriores tienen en común que los datos que manejan son procesados en memoria RAM la cual no garantiza el almacenamiento de dicha información en forma permanente, sino mientras el programa en cuestión se ejecuta.

Si lo que se quiere es que los datos que trabaja el programa se almacenen de forma permanente, la solución consiste en utilizar otros medios de almacenamiento que garanticen dicha persistencia como por ejemplo discos duros, memorias flash, cd-roms, etc., (es decir, uso de almacenamiento secundario).

El almacenamiento de la información o datos en dispositivos de almacenamiento secundario se lleva a cabo a través del concepto de *archivo*, que se define como una secuencia de datos organizados como bytes (flujos o streams) o como una secuencia de registros (estructurados de alguna forma). Para cualquier caso para tener el control sobre un archivo se necesitan llevar a cabo 3 operaciones fundamentales:

a) Creación y/o apertura de un archivo
b) Escritura y/o lectura del o al archivo
c) Cierre del archivo

El control de un archivo para saber su límite, es decir, saber cuando hemos llegado al final de este, se lleva a cabo a través del uso de una marca especial denominada *"fin de archivo"* EOF (End Of File).

El término **archivo** se define entonces como una colección de datos almacenados en un dispositivo secundario externo, ya sea como archivos de texto o ASCII (formato texto) o como archivos binarios o estructurados (formato binario). La información que está contenida en un archivo puede procesarse de forma:

- **Secuencial** desde el inicio hasta el final del archivo en secuencia (dato por dato) o bien,
- **Directa** accediendo de manera directa a un registro o a un byte específico mediante un índice, similar al acceso que se realiza con los arreglos.

En C la manipulación de archivos se lleva a cabo bajo el estándar ANSI a través de un conjunto de funciones de entrada y salida para archivos ya sea de texto o binarios.

5.2. OPERACIONES PARA ARCHIVOS

En C, los archivos están organizados como secuencias de bytes o flujos (en principio sin estructura predeterminada). Estos flujos se conocen como streams y es el medio de abstracción que utiliza el lenguaje para acceder a cualquier dispositivo externo. Los streams pueden ser secuencias binarias o de texto ASCII.

Cada secuencia que se asocia con un archivo está representada en C con el tipo FILE definido dentro de <stdio.h> y que es en donde se encuentran también todas las funciones de entrada y salida para archivos junto con algunos otros tipos de datos importantes que también se utilizan como:

```
size_t   /* entero sin signo */
fpos_t  /* entero sin signo */
FILE    /* tipo estructura para manejar archivos */
```

Además se tienen otros elementos llamados “macros” (que son una especia de constantes) utilizados también cuando se manipulan archivos:

- EOF
- SEEK_SET
- SEEK_CUR
- SEEK_END

EOF como ya comentamos, se utiliza para saber cuando hemos llegado al final de un archive, mientras que SEEK_SET, SEEK_CUR y SEEK_END son utilizadas por una función llamada *fseek()* que se utiliza para el acceso directo de datos de un archivo.

En un programa en C, cada archivo que trabajemos lo tendremos que asociar con un apuntador a archivos para que a través de él podamos identificarlo, escribir y/o leer de él y finalmente cerrarlo. La sintaxis es:

```
FILE *f;
```

5.2.1. Apertura de un archivo

Antes de hacer cualquier otra cosa con un archivo, se tiene que aperturar (independientemente si existe o es un archivo nuevo). La operación de apertura de un archivo lleva consigo dos cosas: la asignación de memoria al archivo asociado al apuntador y la manipulación del archivo físico a través de su representación lógica en el programa, es decir, llevar a cabo operaciones con el apuntador asociado.

Un archivo se apertura usando la operación *fopen(…)* la cual retorna una dirección de memoria asociada con el archivo (esto es, con el apuntador asociado). Si el archivo no se puedo abrir o no se pudo crear (en caso de que no exista) la función retornará NULL. La apertura de un archivo lleva consigo el indicar dentro de *fopen(…)* con un segundo argumento, si el archivo es: de sólo escritura, sólo lectura, lectura y escritura, etc. El primer argumento es el nombre del archivo que físicamente se encuentra almacenado en nuestro dispositivo externo o bien se va a escribir a él.

```
FILE *f;

/*
Apertura para solo escritura del archivo con nombre físico
“MiArchivo.txt”. Si no se especifica lo contrario, el archivo es de texto
*/
f=fopen(“MiArchivo.txt”,”w”);
if (f==NULL)
   {
    printf(“ERROR: No se pudo abrir o crear el archivo…”);
    return 1;
   }
```

A continuación se muestra una tabla con los posibles valores que puede tomar el Segundo argumento de la función *fopen(…)*.

Modo	Significado
“r”	Abre un archivo de texto para lectura
“w”	Abre un archivo de texto para escritura. Si existe previamente, se crea uno vacío
“a”	Abre o crea un archivo para escritura al final de éste
“r+”	Abre u archivo de texto para lectura y escritura
“w+”	Crea un archivo de texto para lectura y escritura. Si existe previamente, se crea uno vacío.
“a+”	Abre o crea un archivo para lectura o escritura al final de éste
“rb”	Abre un archivo binario para lectura
“wb”	Crea un archivo binario para escritura. Si existe previamente, se crea uno vacío.
“ab”	Abre o crea un archivo binario para escritura al final de éste
“rb+”	Abre un archivo binario para lectura y escritura
“w+”	Crea un archivo binario para lectura y escritura. Si existe previamente, se crea uno vacío.
“ab+”	Abre o crea un archivo binario para lectura y escritura al final.

5.2.2. Cierre de un archivo

Un archivo debe cerrarse cuando se decide que ya no será utilizado. La operación de cierre de un archivo se lleva a cabo con la función *fclose(…)* y es importante hacerlo para la liberación de los recursos asignados al archivo, pero sobre todo porque fuerza la escritura al almacenamiento externo de los datos que podrían haber quedado en el buffer de memoria asignado previamente cuando se utilizó *fopen(…)*. Su sintaxis es:

```
FILE *f;
. . .
fclose(f);
```

5.2.3. Funciones para el procesamiento de archivos

FUNCIÓN	SIGNIFICADO / USO
`FILE *fopen(const char *nomarch, const char *modo);`	Abre un archivo y lo asocia con un dispositivo físico externo. Requiere de un nombre de archivo y un modo de uso
`int fclose(FILE *f)`	Cierra un archivo que fue previamente abierto mediante *fopen(…)*. Requiere del apuntador asociado al archivo.
`int putc(int car, FILE *f);` `int fputc(int car, FILE *f);`	Escriben caracteres en un archive que se haya abierto con anterioridad para su escritura
`int getc(FILE *f);` `int fgetc(FILE *f);`	Regresa una marca de EOF si se ha llegado al final del archivo o un carácter almacenado en el archivo.
`int fputs(const char *cad, FILE *f);`	Escribe la cadena *cad* en el archivo asociado a *f. Si se produce un error devuelve EOF
`char *fgets(char *cad, int longitud, FILE *f);`	Lee una cadena de la secuencia especificada hasta longitud-1 caracteres. La cadena resultante termina con el carácter de fin de línea '\0'. La función retorna una apuntador a la cadena o bien '\0' en caso de error.
`void rewind(FILE *f);`	Inicializa el indicador de posición del archivo asociado al principio de éste, es decir, rebobina el archivo.
`int ferror(FILE *f);`	Determina si se produjo un error en una operación sobre un archivo. Si hay un error devuelve un valor distinto de cero, en caso contrario devuelve cero.
`int remove(char *nomArch);`	Borra el archivo especificado. Si tiene éxito regresa cero, sino un valor distinto de cero.
`int fflush(FILE *f);`	Escribe todos los datos almacenados en el buffer de la memoria en el archivo asociado a *f. Si tiene éxito regresa cero, sino regresa EOF.
`size_t fread(void *buf, size_t numbytes, size_t cuenta, FILE *f);`	Lee bloques de bytes cualquier tipo de datos. Devuelve el número de elementos leídos. *buf* representa un apuntador a

	una región de memoria donde escribir los datos leídos del archivo. *numbytes* es el número de bytes a leer. *cuenta* determina cuantos elementos se van a leer. *f* es el apuntador al archivo asociado.
size_t write(void *buf, size_t numbytes, size_t cuenta, FILE *f);	Escribe al archivo bloques de bytes cualquier tipo de datos. Devuelve el número de elementos escritos.
int fprintf(FILE *f, const char *cadenaControl, …);	Se comporta igual que como *printf()* pero sobre el archivo asociado a *f.*
int fscanf(FILE *f, const char *cadenaControl, …);	Se comporta igual que como *scanf()* pero sobre el archivo asociado a *f.*
FILE freopen(const char *nomArch, const char *modo, FILE *secuencia)	Asocia una secuencia con un archivo nuevo. Se puede utilizar para asociar una secuencia estándar (stdin, stdout, stderr) con un archivo nuevo.

5.3. ARCHIVOS DE TEXTO

A continuación se muestra un programa donde se crea un archivo de texto. Los caracteres ingresados por el usuario a través de la entrada estándar (stdin) o teclado se escriben al archivo. Posteriormente el archivo se vuelve a abrir para su lectura. Se lee todo su contenido y se muestra en pantalla:

```
#include <stdio.h>

int main()
{
 FILE *f; //Apuntador a un archivo
 char car;

 // Se crea un archivo de texto para escritura
 f=fopen("MiArchivo.txt", "w");
 if (f==NULL)
    {
     printf("ERROR: No se pudo crear el archivo...");
     return 1;
    }

/* Se leen caracteres desde stdin y se escriben al archivo */
 do{
     car=getc(stdin);
     fprintf(f,"%c",car);
   }while(car!='.');
 fclose(f);
```

```
// Se vuelve a abrir el archivo pero para su lectura
 f=fopen("MiArchivo.txt", "r");
 if (f==NULL)
    {
     printf("ERROR: No se pudo crear el archivo...");
     return 1;
    }

 system("cls");
 printf("EL ARCHIVO QUE CREASTE TIENE EL SIGUIENTE CONTENIDO:\n");

 // Se lee el archivo y se muestra en pantalla su contenido
 do{
     car=getc(f);
     printf("%c",car);
   }while(car!=EOF);
 fclose(f);

 printf("\n\n");
 system("pause");
 return 0;
}
```

En este otro ejemplo se muestra el código de un programa en C donde se utiliza el mismo archivo de texto creado en el ejemplo anterior pero como parámetro de una función que muestra su contenido.

```
#include <stdio.h>

void mostrar_archivo(FILE *arch);

int main()
{
 FILE *f;
 char car;

 f=fopen("MiArchivo.txt", "r");
 if (f==NULL)
    {
     printf("ERROR: No se pudo crear el archivo...");
     return 1;
    }

 mostrar_archivo(f);
 fclose(f);

 printf("\n\n");
 system("pause");
 return 0;
```

```
}

void mostrar_archivo(FILE *arch)
{
 char c;
 rewind(arch);
 while (!feof(arch))
    {
     c=fgetc(arch);
     printf("%c",c);
    }
}
```

5.4. ARCHIVOS DE ACCESO DIRECTO O BINARIOS

En determinadas ocasiones nos interesa accesar a un punto específico del archivo para procesar un dato en particular y no procesar todos los elementos desde el inicio hasta el final de a uno a la vez. El lenguaje C proporciona este tipo de acceso "directo" a los archivos.
La declaración de un archivo de acceso directo es idéntica a la de otros archivos y sólo se distingue de ellos por las funciones que se manejan las cuales son funciones de posicionamiento.

Cuando un archivo se abre, su apuntador apunta al primer registro o elemento. Después de cada operación de lectura o escritura, el apuntador se incrementa en uno y apunta al siguiente registro o byte. Cabe hacer mención que sólo pueden accederse de manera directa los archivos binarios y no los de texto los cuales siempre son de acceso secuencial.

Para poder posicionarnos en un registro o byte de un archivo de acceso directo, utilizaremos la función *fseek()* que tiene la siguiente sintaxis:

```
int fseek(FILE *f, long numbytes, int origen);
```

Con esta fución nos posicionamos en un registro particular dentro del archivo asociado, pues lo que hace es cambiar el valor del apuntador y moverlo directamente a un registro o byte en particular.

**f* es el apuntador al archivo asociado y devuelto por una llamada a *fopen()*
numbytes es un entero largo, que representa el número de bytes a partir de *origen* que supondría la nueva posición. Este valor podría ser negativo si lo que se quiere es retroceder al apuntador.

Origen es alguna de las siguientes macros:

- SEEK_SET (*numbytes* es relativo al principio del archivo)
- SEEK_CUR (*numbytes* es relativo a la posición actual)
- SEEK_END (*numbytes* es relativo al fin de archivo)

El siguiente ejemplo muestra un código que lee un archivo de texto carácter por carácter y lo convierte en mayúsculas. Cuando lee un carácter, hacemos que el apuntador a registro avance automáticamente a la posición siguiente. Por último se lleva el apuntador al inicio y se muestra el contenido del archivo por la salida estándar.

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

int main()
{
 FILE *f;
 char car;

 f=fopen("MiArchivo.txt","r+");
 if(f==NULL)
   {
    printf("\nERROR: No se pudo abrir el archivo...");
    system("pause");
    return 1;
   }

 while((car=getc(f))!=EOF)
    {
    if (car!='\n')
    {
     car=toupper(car);
     fseek(f,-1,SEEK_CUR);
     putc(car,f);
     fseek(f,0,SEEK_CUR);
     }
    }

 fseek(f,0,SEEK_SET);
 while((car=getc(f))!=EOF)
    {
     printf("%c",car);
    }
 fclose(f);

 printf("\n");
 system("pause");
 return 0;
}
```

Finalmente otro ejemplo en el que el programa crea un archivo binario, cuyos registros contienen los siguientes campos:

- Código (entero)
- Descripción (cadena de caracteres).
- Precio unitario (real).

El programa debe permitir la carga de datos en el archivo. Luego, leer y mostrar el último registro válido del archivo (anterior al EOF). Con ulterioridad se debe posicionar sobre la marca de fin de archivo (EOF) y permitir agregar datos nuevos al final. Por último, mostrar el contenido del archivo completo.

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

#define N 5

typedef struct {
      int codigo;
      char descripcion[50];
      float precio_unit;
} t_producto;

/* Prototipos */
t_producto cargar_producto();
void imprimir_producto(t_producto);

int main()
{
      FILE *stock;
      t_producto buffer;
      int i;

      if ((stock=fopen("stock.dat","wb+")) == NULL)
      {
            printf("No se puede abrir el archivo.\n");
            return 1;

      } /* Carga de datos en el archivo */
      for (i=0; i<N; i++)
      {
            buffer = cargar_producto();
            fwrite(&buffer, sizeof(t_producto), 1, stock);
      }

      /* Lectura del último registro (registro N-1) */
      fseek(stock, (N-1)*sizeof(t_producto), SEEK_SET);
      fread(&buffer, sizeof(t_producto), 1, stock);
```

```
	imprimir_producto(buffer);

	/* Posicionamiento sobre EOF (para agregar un registro al final) */
	fseek(stock, 0, SEEK_END);
	buffer = cargar_producto();
	fwrite(&buffer, sizeof(t_producto), 1, stock);

	/* Listado secuencial del archivo */
	rewind(stock);
	while (!feof(stock))
	{
		fread(&buffer, sizeof(t_producto), 1, stock);
		imprimir_producto(buffer);
	}
	fclose(stock);

	return 0;
	}
	t_producto cargar_producto()
	{
		t_producto buffer;
		printf("DATOS DEL PRODUCTO\n");
		printf(" Código: ");
		scanf("%d", &(buffer.codigo));
		while (getchar() != '\n'); /* Limpieza buffer de teclado */

		printf(" Descripción: ");
		fgets(buffer.descripcion, 50, stdin);
		if (buffer.descripcion[strlen(buffer.descripcion)-1] == '\n')
		   buffer.descripcion[strlen(buffer.descripcion)-1] = '\0';
		else
		  while (getchar() != '\n'); /* Limpieza buffer de teclado */

		printf(" Precio unitario: ");
		scanf("%f",&(buffer.precio_unit));
		while (getchar() != '\n'); /* Limpieza buffer de teclado */
		printf("\n");
	return buffer;
}

void imprimir_producto(t_producto producto)
{
	printf("DATOS DEL PRODUCTO\n");
	printf(" Código: %d\n", producto.codigo);
	printf(" Descripcion: %s\n", producto.descripcion);
	printf(" Precio unitario: %f\n", producto.precio_unit);
	printf("\n");
}
```