Unidad Temática N°3

~ Archivos ~

"Tenemos objetos persistentes, se llaman archivos".

~ Ken Thompson

Archivos

Antes de comenzar cabe aclarar que un archivo es un concepto, cuyo contenido es una agrupación lógica de datos y, mediante el cual podemos restringir el acceso a dichos datos a diferentes usuarios

Antes de empezar a trabajar con archivos en C, debemos saber todas las operaciones que se realizan (creación, apertura, lectura, escritura, cierre) se realizan utilizando *flujos de datos,* también llamados *stream*.

Un flujo de datos es una abstracción que permite la circulación (envío y/o recepción) de datos desde un origen hacia un destino. La apertura de un archivo, supone establecer la conexión del programa con el dispositivo que contiene el archivo y por el canal que comunica el archivo con el programa transitan secuencias de datos.

Hay 3 flujos o canales abiertos de forma automática cuando se ejecuta un programa:

- FILE *stdin: asocia la entrada estándar (teclado) con el programa.
- FILE *stdout: asocia la salida estándar (pantalla) con el programa.
- FILE *stderr: asocia la salida estándar (pantalla) con los errores del programa.

Es por estos flujos que cuando utilizamos `printf("Hola mundo!!");` se escribe en stdout y cuando hacemos `scanf("%d", numero);` se capturan los datos desde el teclado.

Respecto de los tipos de archivos hay 2:

- Texto: archivos cuyo contenido son caracteres de texto. Una de sus característica es que puede ser leído por una persona, otra es que naturalmente ocupan más espacio que un archivo binario, son más portables.
- Binarios: archivos cuyo contenido son secuencias binarias. No pueden ser leídos por una persona pero su eficiencia es mayor respecto de los archivos de texto. Además ocupan menos espacio que un archivo de texto (sobre todo si la mayoría de los datos son números). Son menos portables que un archivo de texto.

Puntero a FILE

Para poder acceder a un archivo y trabajar con el mismo desde un programa hecho en C, vamos a utilizar la estructura FILE. Esta estructura contiene información sobre el archivo, como la dirección, el modo de apertura y varios detalles más que salvo raras excepciones se utilizan.

La estructura FILE está declarada en el archivo stdio.h por lo que para utilizarla debemos antes incluirlo en nuestro archivo fuente.

Recuerde que para trabajar con archivos debemos declarar un puntero a FILE.

```
#include <stdio.h>

int main(){
  FILE *ptrArchivo = NULL;
  ...
}
```

Operaciones con archivos

Apertura

La primer operación a realizar cuando trabajamos con archivos es abrirlo. En esta operación vamos a conectar el programa con el archivo e indicar cómo vamos a tratar el archivo, si como un archivo de texto o como un archivo binario. También debemos indicar el modo de apertura.

Para abrir un archivo utilizamos la función *fopen*.

Ejemplo:

```
#include <stdio.h>

int main(){
  FILE *ptrArchivo = NULL;
  ptrArchivo = fopen(“datos.txt”, “r”);
  ...
}
```

Como vemos en el ejemplo, *fopen* posee 2 parámetros, el primero es la ruta (path) del archivo, mientras que el segundo es el modo de apertura. Ambos son de tipo *const char **.

La función *fopen* devuelve un puntero a FILE.

Siempre debemos verificar el valor retornado por *fopen*, ya que en caso de no poder abrir el archivo devolverá NULL.

Modos de apertura de un archivo

Al abrir un archivo, *fopen*, espera como segundo argumento el modo. El modo indicará cómo tratar el archivo si como un archivo de texto o como un archivo binario y además si el mismo se está abriendo para lectura, escritura o modificación.

Para indicar que el archivo a abrir es binario, debemos agregar una ‘b’ en el modo.

Modo	Descripción
“r”	Abre un archivo de texto para lectura (el archivo debe existir).
“w”	Abre un archivo de texto para escritura. Si el archivo no existe lo crea y si existe se pierden sus datos.
“a”	Abre un archivo para escritura al final (el archivo debe existir).

En todos los casos agregando una ‘b’ se indica que el archivo es un archivo binario.

El señalador + permite que se puedan ambas operaciones, es decir, operaciones de lectura y escritura; de otro modo únicamente se podría ejecutar la acción por defecto.

Ejemplo:

```
#include <stdio.h>      // printf, fopen
#include <stdlib.h>     // exit, EXIT_FAILURE

int main ()
{
  FILE * pFile;
  pFile = fopen ("myfile.dat","rb");
  if (pFile == NULL)
  {
    printf ("Error al abrir el archivo");
    exit (EXIT_FAILURE);
  }
  ...
  return 0;
}
```

Cierre

Para cerrar la conexión establecida entre el programa y el archivo utilizamos la función *fclose*. Esta función recibe el puntero a FILE que obtuvimos en la operación de apertura.

Ejemplo:

```
#include <stdio.h>

int main(){
  FILE *ptrArchivo = NULL;
  ptrArchivo = fopen("datos.txt", "r");
  …
  fclose(ptrArchivo);
}
```

Los archivos en C trabajan con una memoria intermedia, un *buffer* con el objetivo de reducir las operaciones de E/S. Cuando este buffer se llena se vuelcan los datos al disco o se vuelve a realizar una lectura de los datos, según qué operación estemos realizando.

Puede ocurrir que al finalizar el procesamiento de los datos, en el caso de una operación de escritura de archivos, queden datos en el buffer. La función *fclose* también vuelca el buffer al archivo, por esto es importante no olvidar cerrar el archivo.

Caso particular entrada estándar: fflush

Como mencionamos en el párrafo anterior, las operaciones de entrada / salida en C trabajan con un buffer de datos, una suerte de array podríamos decir.

En ocasiones cuando estamos leyendo datos de la entrada estándar (stdin), pueden quedar en el buffer ciertos caracteres que podrían complicar la lectura de los datos siguientes. El caso más típico es mezclar el uso de *scanf* y *gets.*

Ejemplo:

"Caso A"	"Caso B"

`int edad = 0;`
`char nombre[81];`
 	
`printf("Ingrese la edad: \n");`
`scanf("%d", &nro);`
`printf("Ingrese el nombre: \n");`
`gets(nombre);`
	`int edad = 0;`
	`char nombre[81];`
	
	`printf("Ingrese la edad: \n");`
	`scanf("%d", &nro);`
	`fflush(stdin);`
	`printf("Ingrese el nombre: \n");`
	`gets(nombre);`

En el caso “A” puede ocurrir que el \n que se inserta cuando el usuario presiona la tecla enter quede en el buffer de stdin ya que *scanf* no lo lee. Bajo ese escenario, *gets* tomará el caracter ‘\n’ y no leerá nada más, con lo cual nos dará la sensación que no se está ejecutando o que no está guardando los datos en la variable nombre.

La solución es utilizar la función *fflush(stdin)* que vacía el buffer, eliminando en este caso el ‘\n’ y permitiendo que *gets* lea la entrada correctamente.

Si bien lo expuesto en estos ejemplos funciona en algunas plataformas, carece de portabilidad, es decir, no funciona en todas las plataformas. Una solución portable sería la siguiente:

```
#include <stdio.h>
#include <stdlib.h>
#define LONG_PAL 81;

void discardChars(){
	char c;
	while((c = getchar()) != '\n' && c != EOF);
	return;
}

int main(){

	int nro = 0;
	char pal[LONG_PAL];

	printf("ingrese nro: ");
	scanf("%d", &nro);
	printf("\ningrese nombre: ");
	discardChars();
	fgets(pal, LONG_PAL, stdin);
	printf("\nNumero: %d\tNombre: %s\n", nro, pal);

	return 0;
}
```

Escritura y lectura de archivos

Veremos a continuación un resúmen de las funciones utilizadas para leer o escribir archivos tanto de texto como binarios. Luego analizaremos algunos ejemplos y buenas prácticas para operar con archivos.

Archivos de Texto

Escritura

int fputc(int character, FILE * stream): escribe un caracter en el archivo apuntado por stream. Avanza el puntero del archivo en 1 posición. En caso de éxito devuelve el caracter en caso de error EOF.

```
#include <stdio.h>

int main(){
  FILE *ptrArchivo = NULL;
  ptrArchivo = fopen(“letras.txt”, “w”);
  for(char letra = ‘a’; letra <= ‘c’; letra++){
    fputc(letra, ptrArchivo);
  }
  fclose(ptrArchivo);
}
```

int fputs (const char * str, FILE * stream): escribe la cadena *str* en el archivo apuntado por stream. Avanza el puntero del archivo N posiciones (tantas como caracteres haya escrito). En caso de éxito devuelve la cantidad de caracteres escritos y en caso de error EOF. La función fputs no agrega el caracter ‘\0’.

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

int main(){
  FILE *archivo = NULL;
  if( (archivo=fopen("pruebafputs.txt", "w")) == NULL ){
    exit(EXIT_FAILURE);
  }

  char *palabras[] = {"linea 1", "linea 2", "linea 3"};
  for(int i = 0; i < 3; i++){
    fputs(palabras[i], archivo);
  }

  fclose(archivo);
  exit(EXIT_SUCCESS);
}
```

```
Archivo:

linea 1linea 2linea 3
```

int fprintf (FILE * stream, const char * format, ...): permite escribir varias variables (de tipos de datos nativos) en el archivo apuntado por stream utilizando algún formato particular. Avanza el puntero del archivo N posiciones (tantas como caracteres haya escrito). En caso de éxito devuelve la cantidad de caracteres escritos y en caso de error un número negativo. Para ver la especificación completa sobre los formatos que se pueden utilizar, visitar la página: http://www.cplusplus.com/reference/cstdio/fprintf/

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

int main(){
  FILE *archivo = NULL;
  if( (archivo=fopen("pruebafputs.txt", "w")) == NULL ){
    exit(EXIT_FAILURE);
  }

  char *palabras[] = {"linea 1", "linea 2", "linea 3"};
  for(int i = 0; i < 3; i++){
    fprintf(archivo, "%d)_ %s\n", i, palabras[i]);
  }

  fclose(archivo);
  exit(EXIT_SUCCESS);
}
```

```
Salida:

0)_ linea 1
1)_ linea 2
2)_ linea 3
```

Lectura

int fgetc (FILE * stream): devuelve el caracter que se encuentra en la posición actual del archivo apuntado por stream. Puede devolver EOF en caso de error o si se alcanzó el final del archivo

```
#include <stdio.h>

int main(){
  FILE *ptrArchivo = NULL;
  ptrArchivo = fopen(“letras.txt”, “r”);
  char letra = ‘a’
  while((letra = fgetc(ptrArchivo))!= EOF){
    printf(“Letra leída: %c\n”, letra);
  }
```

```
  fclose(ptrArchivo);
}
```

char * fgets (char * str, int num, FILE * stream): lee y devuelve una cadena de caracteres del archivo apuntado por stream. La función devuelve un puntero a la cadena leída o NULL en caso de error. La lectura de la cadena finaliza cuando encuentra el caracter de fin de línea, fin de archivo o bien cuando leyó num-1 caracteres

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

int main(){
       FILE *archivo = NULL;
       if(  (archivo=fopen("pruebafputs.txt", "r"))==NULL  ){
           exit(EXIT_FAILURE);
       }
       char palabra[7];

       while(fgets(palabra, 8, archivo) != NULL){
           printf("Palabra leída: %s\n", palabra);
       }

       fclose(archivo);
       exit(EXIT_SUCCESS);
}
```

Note que al invocar a fgets, la longitud debe ser la longitud de la línea a leer + 1.

int fscanf (FILE * stream, const char * format, ...): permite leer varios datos desde el archivo apuntado por stream en distintas variables, especificando el formato de los datos leídos. Tener en cuenta que este formato debe coincidir en orden y cantidad con las variables que se especifican en la llamada a fscanf. En caso de éxito, devuelve la cantidad de items leídos y en caso de error devuelve EOF.

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

int main(){
  FILE *archivo = NULL;
  if(  (archivo=fopen("pruebafputs.txt", "r"))==NULL  ){
    exit(EXIT_FAILURE);
  }
```

```
  int num = -1;
  char palabra[8];

  while(fscanf(archivo, "%d)_ %s\n", &num, palabra) != EOF){
      printf("Número leído: %d\n", num);
      printf("Palabra leída: %s\n", palabra);
  }

  fclose(archivo);
  exit(EXIT_SUCCESS);
}
```

Archivos Binarios

Escritura

size_t fwrite (const void * ptr, size_t size, size_t count, FILE * stream): escribe un buffer de cualquier tipo de dato en un archivo binario.

- ptr: puntero a los datos a escribir.
- size: tamaño del tipo de elemento a escribir, en bytes.
- count: cantidad de elementos a escribir.
- stream: puntero al archivo.

El puntero al archivo avanza la cantidad de bytes escritos.

La función devuelve la cantidad de elementos que pudo escribir. En caso de éxito deberán coincidir con lo indicado en el parámetro count.

Ejemplo:

```
#include <stdio.h>

int main ()
{
  FILE *pFile;
  char buffer[] = { 'x' , 'y' , 'z' };
  pFile = fopen ("myfile.bin", "wb");
  fwrite (buffer , sizeof(char), sizeof(buffer), pFile);
  fclose (pFile);
  return 0;
}
```

Tener en cuenta que la extensión del archivo no define si es de texto o binario, de hecho podría no estar ej: myfile; lo importante es el modo de apertura del archivo.

Lectura

size_t fread (void * ptr, size_t size, size_t count, FILE * stream): lee datos almacenados en un archivo binario y los devuelve en un buffer de cualquier tipo de dato.

- ptr: puntero al buffer donde alojar los datos leídos.
- size: tamaño de los datos en bytes.
- count: cantidad de datos a leer.
- stream: puntero al archivo.

El puntero al archivo avanza la cantidad de bytes leídos.

La función devuelve la cantidad de elementos que pudo leer. En caso de éxito deberán coincidir con lo indicado en el parámetro count.

Ejemplo:

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

int main () {
  FILE * pFile;
  char * buffer;
  size_t result;

  pFile = fopen ( "myfile.bin" , "rb" );
  if (pFile==NULL) {
    fputs ("File error",stderr);
    exit (EXIT_FAILURE);
  }

  // allocate memory to contain the whole file:
  buffer = (char*) malloc (sizeof(char) * 3);
  if (buffer == NULL) {
    fputs ("Memory error",stderr);
    exit (EXIT_FAILURE);
  }

  result = fread (buffer, sizeof(char), 3, pFile);
  if (result != 3) {
    fputs ("Reading error",stderr);
    exit (EXIT_FAILURE);
  }

  fclose (pFile);
  free (buffer);
  return 0;
}
```

Función feof()

La función feof (acrónimo de “**f**ile **e**nd **o**f **f**ile”) nos indica cuando se alcanzó el final de un archivo que estamos leyendo.

Esta función puede ser utilizada en conjunto con cualquiera de las funciones de lectura que vimos anteriormente.

La “regla de oro” para utilizar esta función es que **SIEMPRE antes de ser invocada debemos haber intentado leer el archivo**.

La función recibe un puntero a FILE como parámetro y devuelve true si se alcanzó el final del archivo o false en caso de que no se haya alcanzado el fin del archivo.

Ejemplo:

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

int main () {
  FILE * pFile;
  char * buffer;
  size_t result;

  pFile = fopen ( "myfile.bin" , "rb" );
  if (pFile==NULL) {
    fputs ("File error",stderr);
    exit (EXIT_FAILURE);
  }

  // allocate memory to contain the whole file:
  buffer = (char*) malloc (sizeof(char) * 3);
  if (buffer == NULL) {
    fputs ("Memory error",stderr);
    exit (EXIT_FAILURE);
  }

  // siempre realizar una lectura antes de utilizar feof
  result = fread (buffer, sizeof(char), 3, pFile);
  while ( !feof(pFile) ) {

    // procesar los datos leídos.

    result = fread (buffer, sizeof(char), 3, pFile); // Volver a leer
    if (result != 3) {
      fputs ("Reading error",stderr);
      exit (EXIT_FAILURE);
    }
  }
```

```
  fclose (pFile);
  free (buffer);
  return 0;
}
```

Funciones para acceso aleatorio

Cuando trabajamos con archivos binarios es frecuente tratar de acceder a los datos a partir de su posición dentro del archivo sin tener que recorrer registro por registro (secuencial).

Para lograr este objetivo, podemos utilizar las funciones *ftell* y *fseek*.

int fseek (FILE * stream, long int offset, int origin): permite situar el puntero del archivo en una posición determinada.

- stream: puntero al archivo
- offset: desplazamiento expresado en bytes
- origin: indica desde donde se realizará el desplazamiento. Para esto podemos utilizar cualquiera de estas 3 constantes
    - SEEK_SET: indica que el desplazamiento debe hacerse desde el principio del archivo
    - SEEK_CUR: indica que el desplazamiento debe hacerse desde la posición actual del archivo
    - SEEK_END: indica que el desplazamiento debe hacerse desde el final del archivo

Mayormente utilizaremos SEEK_SET

Ejemplo:

```
#include <stdio.h>

typedef struct {
  char nombre[80];
  int dni;
} ST_PERSONA;

int main ()
{
  ST_PERSONA persona;
  FILE *pFile;
  pFile = fopen ("agenda.bin", "rb");
  // posiciona el puntero en el 4to registro del archivo
  fseek(pFile, 3 * sizeof(persona), SEEK_SET);
```

```
  fread (&persona , sizeof(persona), 1, pFile);
  imprimir(persona);
  fclose (pFile);
  return 0;
}
```

long int ftell (FILE * stream): devuelve la posición actual del archivo. Recibe por parámetro un puntero a FILE.

Esta función es útil si deseamos saber la cantidad de registros de un archivo:

Ejemplo:

```
#include <stdio.h>

typedef struct {
  char nombre[80];
  int dni;
} ST_PERSONA;

int main ()
{
  ST_PERSONA persona;
  FILE *pFile;
  pFile = fopen ("agenda.bin", "rb");

  // tomo la posicion actual
  long actualPos = ftell(pFile);

  // muevo el puntero al final del archivo
  fseek(pFile, 0, SEEK_END);

  // tomo la posicion actual (ubicado al final)
  long ultimo = ftell(pFile);

  // vuelvo a donde estaba al principio
  fseek(pFile, actualPos, SEEK_SET);

  printf(“Cantidad de registros: %d\n”, (int)(ultimo / sizeof(persona));

  fclose (pFile);
  return 0;
}
```