Archivos - Input / Output

Pablo R. Ramis

Universidad Nacional de Rosario, Instituto Politécnico, Dto. de Informática,
prramis@ips.edu.ar,
WWW home page: http://informatica.ips.edu.ar

Resumen Hasta el momento, todo lo que hemos hecho se basa en procesar datos entrados por teclaso o sea la entrada estándar y mostrados por pantalla, salida estandar, además de esto, es importante y útil, que en algunos casos haya una persistencia de la información, para esto se utilizaran archivos en disco.
Nos fijaremos en los siguientes temas:

1. Tipos de archivos y sus diferencias.
2. Acceso a archivos secuenciales.
3. Abordaje de archivos de acceso aleatorio.

Cabe mencionar que los temas no estarán agotados en su explicación, se presenta aquí un resumen de las funciones principales o más comunes y ejemplos de su uso.

1. Archivos

Tenemos dos posibles modos, el secuencial y el aleatorio. El usar una forma u otra dependerá de lo que se quiera hacer. Si se es prolijo en la estructura en que se guarda la información, nada impide que un archivo sea abierto de un modo u otro en diferentes momentos.

El modo de acceso al archivo condicionará la forma de sus operaciones: lectura, guardado, modificación y eliminación de los datos.

En caso en el acceso secuancial, como se desprende del nombre, se hará en el mismo orden en que se guardó los datos. Se podrá avanzar, retroceder y saltear datos, pero será muy dificil o muy costoso el agregar en forma intercalada información nueva, una forma seria armar un nuevo archivo que combine la información vieja con la nueva en el lugar adecuado.

En los archivos de acceso aleatorio a diferencia de lo que se expresó antes, no requiere recorrer todo el archivo para llegar a un sector de lectura determinado. Esto da mas flexibilidad aunque puede ser más laboriosa su programación.

2. Archivos de Acceso Secuencial

Todas las funciones que se nombren o usen se encuentran en la libreria *stdio.h*.

2.1. Apertura y cierre del archivo

Cuando se pretende manipular archivos se debe informar lo que se pretende, se requerirá el nombre del archivo, y que se quiere hacer con él: guardar o escribir datos para inicializarlo, guardar datos agregándolos al final, o leer datos. C se encargará de la comunicación con el sistema operativo para que se garanticen las operaciones. Al cerrar el archivo, C guardará en el todos los datos remanentes que no se hayan almacenados hasta el momento, libera el control del archivo y actualiza el directorio en que se trabajaba.

Para abrir el archivo se utiliza la función **fopen()** y para cerrarlo **fclose()**.

$$FILE * puntero\_a\_archivo = fopen(nombre\_de\_archivo, modo\_de\_acceso)$$

y para su cierre:

$$fclose(puntero\_a\_archivo)$$

El valor de *modo_de_acceso* puede ser uno de los siguientes:

Modo	Descripción
"r"	Abre un archivo para lectura
"w"	Abre un archivo para escribirlo (lo crea)
"a"	Abre un archivo para añadir datos al final (append)
"r+"	Abre un archivo para actualización (lectura y escritura)
"w+"	Abre un archivo para actualización (primero lo crea, luego permite lectura y escritura)
"a+"	Abre un archivo para actualización (lee todo el archivo o escribe al final)

Hay algunas variantes que se pueden utilizar, por ejemplo se podría agregar, aunque sea una redundancia, la letra *"t"* que significa *texto* o sea que el archivo es de tipo *ASCII*. Ejemplo: *rt"*. Esto haría que el archivo sea compatible con cualquier aplicación que pueda abrir y leer este tipo de codificación.

La otra opción es utilizar la letra *"b"*. Ejemplo: *rb"*. Esto indica que el modo de acceso es *binario*.

Modos binarios Cuando se utiliza el archivo en formato binario, los datos están çomprimidos". El archivo por lo tanto será de menor tamaño. Este formato es específico de cada sistema, esto atenta contra la portabilidad del archivo el cual dependerá de la plataforma para su correcta lectura. Los modos quedarían de la siguiente forma: *rb", "wb", ."ab", ."+b", "wb+", "w+b", ."b+", r+b"*

Algunos consejos Algo a tener en cuenta, al utilizar el modo "w."en cualquiera de sus variantes, el archivo se crea. Si existiese un archivo con el mismo nombre, C lo va a sobreescribir sin dar alguna advertencia (los datos que existían en el se perderían).

Si existiera un error en la apertura del archivo, C no retornará un puntero a un archivo dañado o inexistente, sino que dicho puntero será NULL. Por lo tanto, siempre que se haga la apertura o creación de un archivo se deberá chequear la validez del puntero antes de manipular al archivo.

Algo a tener en cuenta es que la apertura del archivo es conveniente realizarla en el momento que se utilizará y una vez hecho esto cerrarlo. De esta forma se preserva el riesgo de que ante un corte de luz el archivo tenga menos posibiloidad de corroperse.

2.2. Escribir en un archivo

En C, cualquier función de Entrada/Salida que en su sintaxis requiera un dispositivo efectúa entrada y salida de archivos. por ejemplo: *getc(), putc(), fprintf(), fgets(), fputs().*

Son varias las funciones que existen. las más comunes son *fscanf() y fprintf()* de uso similar a *scanf() y printf()* respectivamente, con la diferencia que el primer parámetro el cuál será el puntero al archivo (o cualquier dispositivo reconocido por C. Ej. STDIN).

En el siguiente ejemplo vemos como leemos tres numeros enteros y se graban en un archivo:

fscanf(puntero_a_archivo, "%d%d%d ", &numero1, &numero2, &numero3);

Las características y el uso son similares al *scanf().*

Son varias las funciones que se pueden utilizar, cada una de ellas podrías ser mas cómoda dependiendo que se pretende hacer, pero tanto *fputs()* y *fprintf()* son totalmente eficientes para hacerlo y su uso es similar a cuando se utilizan para la salida estandar.

2.3. Cerrar un archivo

Como se ha dicho antes, una vez abierto y usado el archivo corresponde cerrarlo. Es la forma de preservarlo ya que sino el archivo correría riesgos de corromperse.

La sintaxis seria:

fclose(puntero_de_archvio);

La función retornará 0 si el archivo se ha cerrado bien o EOF en caso de errores (que será un número negativo).

2.4. Ejemplo de escritura de archivo

El programa que sigue es básico pero mostrará el uso de las funciones que mencionamos. Este caso solo escribiremos en el archivo.

```c
// archivo1.c

// El programa escribe los valores del 1 a 100 en un archivo
    en disco */

#include<stdio.h>

int main()
{

    FILE* pArchivo;

    int contador;

    pArchivo = fopen("./NUMEROS.1", "wt");
    /*valida que el archivo este creado correctamente*/
    if (pArchivo != NULL){
        for (contador = 1; contador <= 100; contador++){
            /* se escriben los numeros */
            fprintf(pArchivo, "%d ", contador);
        }

        /* Se cierra el archivo */
        if (!fclose(pArchivo))
            printf("El archivo ha sido cerrado.\n");
        else
            printf("Error al cerrar el archivo.\n");
    }else{
        printf("Error al crear el archivo\n");
    }
    return 0;
}
```

Compilamos y probamos.

```
$ gcc -o archivos1 archivo1s.c
$ ls
 archivos1  archivos1.c
$ ./archivos1
 El archivo ha sido cerrado.
$ ls
 NUMEROS.1  archivos1  archivos1.c
$ cat NUMEROS.1
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51
52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67
68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83
84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99
100 $
```

2.5. Ejemplo de escritura añadiendo datos

Como hemos mencionado antes, abriendo el archivo en modo *append* agregaremos al final datos en el archivo. Recuerde que si el archivo no existe lo crea y luego escribe.

Para el ejemplo ya disponemos de un archivo NOMBRES.DAT

```
$ cat NOMBRES.DAT
 Pablo Ramis
 Fernando Villar
 Guido Macchi
$
```

Vemos que el archivo ya posee datos. El programa seria el siguiente:

```
1  // archivoAppend.c
2
3  /* El programa abrira al archivo NOMBRES.DAT
4     y agregara datos al final */
5
6  #include<stdio.h>
7
8  int main ()
9  {
10
11     FILE* pArchivo;
12
13     pArchivo = fopen("./NOMBRES.DAT", "a");
14
15     if(pArchivo != NULL){
16         fputs("Alejandro Rodriguez\n", pArchivo);
17         fputs("Diego Bottallo\n", pArchivo);
18         fputs("Federico Severino\n", pArchivo);
19
20         if (fclose(pArchivo))
21             printf("Error al cerrar el archivo\n");
22
23     }else
24         printf("Error al abrir archivo\n");
25
26     return 0;
27 }
```

```
$ gcc -oarchivoAppend archivoAppend.c
$ ./archivoAppend
$ cat NOMBRES.DAT
 Pablo Ramis
 Fernando Villar
 Guido Macchi
 Luciano Diamand
 Alejandro Rodriguez
```

```
 Diego Bottallo
 Federico Severino
$
```

2.6. Lectura de un archivo

Las formas de lectura pueden ser varias, por ejemplo, leer letra por letra o cadenas en forma completas. Si el archivo es una colección de números grabados con *fprintf()* sería natural el usar *fscanf()*.

Los archivos abiertos en modo lectura, sea texto o binario, deben existir sino C lanzará un error.

Lo último a tener en cuenta es cuando detener la lectura. Si el archivo se tiene que leer en forma completa, C provee una solución para identificar el final de la lectura. Es una marca de final de archivo o *EOF* (End Of File). Si se ignorara este final la lectura sería errónea.

```
1  // lecturaArchivo.c
2
3  /* Abre el archivo para leer el contenido */
4
5  #include<stdio.h>
6
7  int main (){
8
9      FILE* pArchivo;
10
11     char nombre[25];
12     char apellido[25];
13
14     pArchivo = fopen("./NOMBRES.DAT", "rt");
15
16     if (pArchivo != NULL){
17         fscanf(pArchivo, "%s %s", nombre, apellido);
18         while ( !feof(pArchivo)){
19             printf("Nombre leido: %s Apellido leido: %s \n",
                   nombre, apellido);
20             fscanf(pArchivo, "%s %s", nombre, apellido);
21         }
22         if (fclose(pArchivo))
23             printf("Error al cerrar el archivo\n");
24     }else
25         printf("Error al abrir archivo\n");
26
27     return 0;
28 }
```

```
$ gcc -olecturaArchivo lecturaArchivo.c
$ ./lecturaArchivo
Nombre leido: Pablo Apellido leido: Ramis
Nombre leido: Fernando Apellido leido: Villar
Nombre leido: Guido Apellido leido: Macchi
Nombre leido: Luciano Apellido leido: Diamand
Nombre leido: Alejandro Apellido leido: Rodriguez
Nombre leido: Diego Apellido leido: Bottallo
Nombre leido: Federico Apellido leido: Severino
$
$
```

En la línea 18 del ejemplo encontramos una función hasta ahora nueva *feof()*. Como *fscanf()* leerá en bloques hasta un espacio, tab o fin de linea nos resultará imposible encontrar manualmente el fin del archivo, por eso utilizamos la función de librería *feof()*. En cada lectura del archivo, el puntero del mismo avanza, esta función irá chequeando cuando se llega al final, retornará 0.

2.7. Ejercicio

1. Realizar un programa en C que realice la lectura del archivo del ejemplo utilizando la función *getc()*.
2. Realice un programa en C que leyendo el archivo NOMBRES.DAT cree uno nuevo llamado APELLIDOS.DAT y se escriba apellido y nombre en vez de nombre y apellido como estaba en el original.
3. Haga una copia exacta del archivo NOMBRES.DAT pero el nuevo que sea binario.

3. Archivos de Acceso Aleatorio

Los archivos de acceso aleatorio muestran el poder de cómputo que posee C. Si solo se trabajara con archivos secuenciales la performance sería muy baja exceptuando que se cargue la totalidad del archivo en un array y se lo procese en memoria.

En el uso de los archivos de acceso aleatorio tendremos en cuenta que en ellos escribiremos registros, que a fines prácticos para nosotros serán las *struct*, o sea, grabaremos de a bloques una colección de datos los cuales pueden ser de diferentes tipos.

C conforma sus archivos con cadenas de caracteres (*stream*) la cual luego se podrá acceder secuencial o aleatoriamente para guardarlos en variables o estructuras.

El guardado de bloques hace que el espacio de uso por cada escritura sea el mismo sin importar el contenido interno, si bien eso podria significar o interpretarse ineficiente no es significativo el desperdicio de espacio y permite luego que el recorrido en modo aleatorio nos de una mejor performance. Las funciones que se utilizarán para esto son diferentes a las de modo secuencial.

3.1. Apertura y escritura de archivos

La apertura es idéntica a lo que se vio con anterioridad. Lo que suele ser recomendable es que el archivo se habra siempre con la opción de lectura y escritura y de forma binaria (ya que esta compresión ayuda a que si el volumen de lo grabado es grande, el archivo este comprimido.

fopen(puntero_a_archivo, "w+b");

Para la escritura podemos disponer de una función útil: *fwrite()*.
El prototipo de la primera es el siguiente:

*t_size fwrite(const void *ptr, size_t tamaño, size_t nmiemb, FILE *pArchivo);*

En donde:

ptr	es el puntero al registro de elementos a escribir.
tamaño	es el tamaño en bytes de cada elemento a escribir.
nmiemb	es el número de elementos, cada uno con un tamaño en bytes.
pArchivo	es el puntero al objeto archivo que será la salida.

El retorno de la función un elemento de tipo *t_size* que será el total de los elementos escritos y que deberá coincidir con *nmiemb* para que no sea error
Un breve ejemplo sería:

```
1  #include <stdio.h>
2
3  int main ( )
4  {
5      FILE *fp;
6
7      char cadena[] = "Hola Mundo!\n";
8
9      fp = fopen ( "./HM.txt", "w+" );
10
11     if (fp == NULL)
12         printf("Error!\n");
13     else
14         fwrite( cadena, sizeof(char), sizeof(cadena), fp );
15
16     fclose ( fp );
17
18     return 0;
19 }
```

```
$ gcc -oescritura1 escritura1.c
$ ./escritura1
$ cat HM.txt
Hola Mundo!
$
```

3.2. Lectura de archivo. Función fseek()

fseek() permite leer un archivo hasta un punto específico. Su formato es simple:

fseek(puntero_a_archivo, numero_entero_long, origen);

Los valores que puede tomar *origen* son:

Descripción	Origen	Constante
Comienzo del archivo	SEEK_SET	0
Posición corriente del puntero archivo	SEEK_CUR	1
Fin del archivo	SEEK_END	2

El *origen* indica a C desde donde debe a comenzar a saltearse el número de de bytes que se especifica con *número_entero_largo*.

Tenemos que tener en cuenta que cuando hablamos del *puntero_a_archivo* no hablamos simplemente de el nombre o el lugar del archivo a leer (sin importar del método que usemos) sino que apunta la ubicación exacta del siguiente byte que debe ser leído o escrito.

Veremos una secuencia de ejemplos para mostrar usos:

```
1  /* letras.c
2   Escribe el abecedario y lee dos letras
3   la I y la R */
4
5   #include<stdio.h>
6
7  int main (){
8      FILE *pArchivo;
9      int letra; /*Almacena las letras*/
10
11     if ((pArchivo = fopen("letras.txt", "w+")) == NULL){
12         printf("Error al abrir archivo\n");
13         return 0;
14     }
15     /*Escribe las letras usando los ASCII*/
16     for (letra = 65; letra <= 90 ; letra ++){
17         putc(letra, pArchivo);
18     }
19     /*Posiciona al puntero en le 8vo bytes*/
20     fseek(pArchivo, 8L, SEEK_SET);
21     letra = getc(pArchivo);
22
23     printf("Primera letra leida %c\n", letra);
24     /*Posiciona al puntero en el Bytes 17*/
25     fseek(pArchivo, 17L, SEEK_SET);
```

```
26      letra = getc(pArchivo);
27
28      printf("Segunda letra leida %c\n", letra);
29
30      fclose(pArchivo);
31
32      return 0;
33  }
```

```
$ gcc -oletras letras.c
$
$ ./letras
Primera letra leida I
Segunda letra leida R
$
```

En la línea 20 y 25 vemos que *fseek* recibe las constantes 8L y 17L respectivamente, esto se debe a que la función recibe un entero largo.

Como se ve, el puntero al archivo siempre esta en el inicio, lo avanzamos para leer los Bytes deseados con el entero largo que se da como argumento. Si usaramos SEEK_END, el puntero del archivo se posicionará al final de este y podriamos retroceder en la lectura.

```
1   /*lectura.c
2     Lee el archivo letras.txt de atras hacia adelante*/
3
4   #include<stdio.h>
5
6   int main()
7   {
8       int contador; /*recorrera en sentido inverso*/
9       int letra;
10      FILE *pArchivo;
11
12      if ((pArchivo = fopen("letras.txt", "r")) == NULL ){
13          printf("Error al abrir el archivo\n");
14          return 0;
15      }
16
17      fseek(pArchivo, 1L, SEEK_END);
18
19      for(contador = 0; contador <= 25; contador++){
20          fseek(pArchivo, -2L, SEEK_CUR);
21          letra = getc(pArchivo);
22          printf("Letra leida: %c\n", letra);
23      }
24
25      fclose(pArchivo);
26      return 0;
```

```
27 }
```

```
$ gcc -olectura lectura.c
$
$ ./lectura
Letra leida: Z
Letra leida: Y
Letra leida: X
Letra leida: W
Letra leida: V
Letra leida: U
Letra leida: T
Letra leida: S
Letra leida: R
Letra leida: Q
Letra leida: P
Letra leida: O
Letra leida: N
Letra leida: M
Letra leida: L
Letra leida: K
Letra leida: J
Letra leida: I
Letra leida: H
Letra leida: G
Letra leida: F
Letra leida: E
Letra leida: D
Letra leida: C
Letra leida: B
Letra leida: A
$
```

Al utilizar SEEK_CUR el programa va retrocediendo 2 Bytes desde la posición del puntero actual, no desde el inicio como se venía usando en los ejemplos anteriores.