ARCHIVOS

1. Introducción

Bienvenidos al maravilloso mundo de los archivos en C. He intentado meter en este cursillo todo lo relacionado con la creación y gestión de archivos, tanto binarios como de texto. Los archivos son una de las más importantes, por no decir la más importante, de las partes de un programa, ya que nos ayudan a almacenar datos necesarios sin que se pierdan al apagar el equipo.

Veremos muchas de las funciones de manejo de archivos, aunque no todas, pero si las más importantes y relevantes, así como las formas de recorrer un archivo de la cabeza a los pies.

2. Definición de archivo

Un archivo en sentido global puede ser desde un monitor hasta una impresora, pasando por un archivo en disco.

La idea más común del concepto de archivo es un conjunto de posiciones de memoria situadas en un disco de los dispositivos externos de almacenamiento del sistema, en las cuales podemos almacenar y recuperar información.

El lenguaje C nos proporciona un acceso secuencial y un acceso directo a los registros de un archivo, pero no soporta el acceso indexado a un registro dado.

Los archivos en C los podemos clasificar, según la información que contengan, en dos grupos: **archivos de texto** y **archivos binarios**.

Los archivos de texto se caracterizan por estar compuestos por una serie de caracteres organizados en líneas terminadas por un carácter de nueva línea (carácter '\n'). Esto nos hace pensar en la idea de usar la impresora como si fuese un archivo de texto.

Por otro lado, los archivos binarios constan de una secuencia de bytes. Podemos decir que cualquier archivo que no sea de texto, será binario.

A la hora de trabajar con archivos, tendremos que especificar antes de usarlos, si serán de texto o binarios.

Podemos establecer una segunda clasificación de los archivos, atendiendo al modo de acceso a su información. De este modo, distinguiremos entre archivos secuenciales y archivos directos.

Los archivos de acceso secuencial se basan en el hecho de que, para acceder a una determinada posición de los mismos, hemos de recorrer desde el principio todas las posiciones hasta llegar a la deseada. Las impresoras son un claro ejemplo de acceso secuencial, así como las cintas magnéticas.

Con el uso de archivos de acceso directo podemos acceder de forma directa a la posición que queramos sin tener que pasar por las posiciones anteriores. El dispositivo de acceso directo por excelencia es el dico magnético.

El lenguaje C trata a los archivos como punteros. En realidad un archivo es un puntero a una estructura de nombre predefinido FILE, cuyas componentes son las características de la variable archivo declarada. Cada archivo deberá tener una estructura FILE asociada.

La estructura FILE se encuentra definida en el archivo de cabecera stdio.h, con lo cual es necesario incluirla en todos los programas que trabajen con archivos mediante la directiva #include <stdio.h>

La forma de declarar variables de tipo FILE es la siguiente:

FILE *f, *f1, ...

Como podemos observar se trata de punteros que apuntan a estructuras de tipo FILE. En realidad lo que ocurre con el tipo FILE, es que redefine a una estructura de la siguiente forma:

```
typedef struct
   {
   short         level;
   unsigned      flags;
   char          fd;
   unsigned char  hold;
   short         bsize;
   unsigned char *buffer, *curp;
   unsigned      istemp;
   short         token;
   } FILE;
```

Esta estructura la podrás ver editando el archivo <stdio.h>. El programador no tendrá que definirla puesto que ya existe.

A continuación vamos a ver las operaciones que podemos realizar con archivos secuenciales y directos.

3. Archivos de acceso secuencial

La primera operación después de declarar un archivo y antes de cualquier otra operación con él es abrirlo.

3.1 # fopen

<variable_archivo> = fopen (<nombre_archivo>, <modo_acceso>);

Donde:

* variable_archivo es la variable declarada de la forma: FILE *<variable_archivo>
* nombre_archivo es el nombre que tendrá el archivo en el dispositivo usado. Podrá ser variable o constante.
* modo_acceso es el modo de apertura del archivo, mediante el cual le diremos al compilador si se trata de un archivo de texto o binario, y si vamos a leer o escribir en el archivo.

Los distintos modos de abrir un archivo son los siguientes:

Modo: Significado

r : Abre un archivo de texto para solo lectura. Si el archivo no existe, devuelve un error
w : Abre el archivo de texto para solo escritura. Si el archivo no existe, lo crea, y si existe lo machaca.
a : Abre el archivo de texto para añadir al final. Si el archivo no existe, lo crea.
r+: Abre el archivo de texto para lectura/escritura. Si el archivo no existe, da un error
w+: Abre el archivo de texto para lectura/escritura. Si el archivo no existe, lo crea, y si existe lo machaca.
a+: Abre el archivo de texto para añadir al final, con opción de lectura. Si el archivo no existe, lo crea.

rb: Abre un archivo binario para solo lectura. Si el archivo no existe, devuelve un error
wb: Abre el archivo binario para solo escritura. Si el archivo no existe, lo crea, y si existe lo machaca.
ab: Abre el archivo binario para añadir al final. Si el archivo no existe, lo crea.
rb+: Abre el archivo binario para lectura/escritura. Si el archivo no existe, da un error
wb+: Abre el archivo binario para lectura/escritura. Si el archivo no existe, lo crea, y si existe lo machaca.
ab+: Abre el archivo binario para añadir al final, con opción de lectura. Si el archivo no existe, lo crea.

Decir que los archivos de texto se pueden referenciar también como "rt", "wt", "at", "rt+", "wt+" y "at+", pero para facilitar más las cosas, cuando no especificamos si queremos abrir un archivo de texto o binario, por defecto se supone que es de texto, es por ello que los modos "r", "w" y "a" se refieren a archivos de texto.

Hemos de tener cuidado con los modos de apertura de los archivos por que al abrir con cualquier modo "w", si ya existía el archivo, se borrará la información que contuviese. Si no existe, lo creará. Por el contrario con el modo "a" también lo creará si no existe el archivo, pero si ya existe, añadirá al final del mismo los datos que escribamos.

Podemos saber si un archivo ha sido abierto correctamente o no. Debido a que los archivos son punteros a estructuras tipo FILE, podemos conocer su valor al hacer que apunten a ellas. La forma de hacerlo es comparando el puntero con la constante predefinida NULL. De esta forma, cuando el puntero valga NULL (no apunta a ninguna variable o posición de memoria), es porque ha habido algún problema al abrir el archivo.

Este problema se puede deber a que no existe el archivo que intentamos abrir para lectura, que no hay espacio suficiente en el disco al tratar de crear el archivo, que la unidad de disco no está preparada, que el disco está dañado físicamente o que la impresora no está conectada. Veamos en ejemplo del testeo de la existencia de un archivo de texto.

```
FILE *f;

f = fopen ("texto.txt", "r");
if (f == NULL)
  printf ("Error, el archivo no existe\n");
else     ...
```

O de esta otra manera

```
FILE *f;

f = fopen ("texto.txt", "r");
if (!f)
  printf ("Error, el archivo no existe\n");
else     ...
```

O bien, de forma abreviada

```
FILE *f;

if ((f = fopen ("texto.txt", "r")) == NULL)
  printf ("Error, el ficnero no existe\n");
else     ...
```

3.2 # fclose

<valor> = fclose (<variable_archivo>);

Donde:

* valor es el valor que nos dirá si ocurre algún error cerrando el archivo. 0 indica que todo ha ido bien.
* variable_archivo es la variable declarada de la forma: FILE *<variable_archivo>

Cuando terminemos de trabajar con un archivo hemos de realizar la operación de cierre del archivo. Si no lo hacemos podemos ocasionar la pérdida de todos los datos del mismo. Si se necesitan cerrar varios archivos a la vez, nos podemos ahorrar varios fclose utilizando la funcion fcloseall.

3.3 # fcloseall

<numero_archivos> = fcloseall();

Donde:

* numero_archivos es una variable entera que indica el número de archivos cerrados, o EOF si ha ocurrido un error.

Ahora veamos unas funciones para escribir y leer datos de un archivo de texto.

3.4 # putc

Esta función escribe un carácter en un archivo de texto. Devuelve un entero si la escritura es correcta. De otra forma devuelve el valor EOF que indica que ha habido un error. Su formato es:

putc (<carácter>, <var_fich>);

Donde:

* carácter es el carácter que se desea escribir en el archivo.
* var_fich es la variable declarada como FILE.

Ejemplos:

* putc ('a', f1);
* c = 'G'; putc (c, f1);

Ni que decir tiene que cuando usemos funciones para escribir en archivos, estos tendrán que haber sido abiertos en modo escritura o para añadir datos.

3.5 # getc

Esta función lee un carácter de un archivo de texto abierto en modo lectura. Devuelve un entero si la lectura es correcta. De otra forma devuelve el valor EOF que indica que hemos llegado al final del archivo. Su formato es:

<carácter> = getc (<var_fich>);

Donde:

* carácter es la variable que contendrá el carácter leído del archivo.
* var_fich es la variable declarada como FILE.

Ejemplos:

* c = getc (f1);

Un ejemplo del uso de estas dos funciones sería leer todos los caracteres de un archivo de texto y escribirlos en otro archivo de texto y por pantalla a la vez para comprobar la correcta ejecución:

```
#include <stdio.h>

main ()
{
  FILE *f_in, *f_out;
  char c;

  clrscr();
  if ((f_in = fopen ("prueba.c", "r")) == NULL)
    {
    printf ("Error de apertura del archivo\n");
    exit (1);
    }
  if ((f_out = fopen ("salida.c", "w")) == NULL)
    {
    printf ("Error de creación del archivo\n");
    exit (1);
    }
  do
    {
    c = getc(f_in);
    putchar(c)       /* escritura en pantalla */
    putc(c, f_out);  /* escritura en el archivo */
    }
  while (c != EOF);
  fclose (f_in);
  fclose (f_out);
}
```

Aparece en este ejemplo una nueva función, la función exit(). Esta función provoca que acabe el programa con lo que la función main() devuelve un valor, en este caso el valor 1. Recordamos que toda función devuelve un valor.

3.6 # fscanf

Esta función trabaja de la misma manera que scanf, pero leyendo los datos formateados de un archivo. Su sintaxis es:

```
fscanf (<var_fich>, <cadena_de_control>, <lista_variables>);
```

Donde:

* var_fich es la variable declarada como FILE.
* cadena_de_control son las cadenas de control que se desean leer, tales como %d, %s, %f, %x, etc...
* lista_variables son las variables que contendrán los valores de la lectura del archivo que deben coincidir con sus respectivas cadenas de control.

Ejemplos:

* fscanf (fich, "%s%d%f", nombre, &edad, &altura);

3.7 # fprintf

Esta función trabaja de la misma manera que printf, pero escribiendo los datos formateados sobre un archivo. Su sintaxis es:

fprintf (<var_fich>, <cadena_de_control>, <lista_variables>);

Donde:

* var_fich es la variable declarada como FILE.
* cadena_de_control son las cadenas de control que se desean escribir, tales como %d, %s, %f, etc...
* lista_variables son las variables que contendrán los valores para la escritura en el archivo que deben coincidir con sus respectivas cadenas de control.

Ejemplos:

* fprintf (fich, "%s%d%f", nombre, edad, altura);

4. Gestión de un archivo secuencial de estructuras

Los componentes o registros de un archivo pueden ser de cualquier tipo de datos, desde tipos básicos (enteros, float, char, etc...) hasta estructuras de datos.

Las operaciones en cualquiera de los casos son siempre iguales. Veamos un ejemplo de un archivo de estructuras:

* Apertura:

```
...
struct Tcli
  {
  char nombre[30];
  char dirección[30];
  ...
  };
```

```
void main()
{
  FILE *f;
  struct Tcli cliente;
  char archivo[13];

  strcpy (archivo, "cliente.dat\0");
  if ((fopen(archivo, "ab") == NULL)
    {
    printf ("Error al abrir el archivo\n");
    exit (1);
    }
  ...
}
```

* Escritura:

```
fwrite (&cliente, sizeof(cliente), 1, f);
```

* Lectura:

```
fread (&cliente, sizeof(cliente), 1, f);
```

Las operaciones básicas en el mantenimiento o gestión de un archivo secuencial son las siguientes:

* Comprobación de la existencia del archivo: Se abrirá el archivo en modo lectura, si no existe, se permitirá la creación del mismo, volviéndolo a abrir en modo escritura.
* Altas: Se abrirá el archivo en modo añadir.
* Bajas: Utilizaremos dos archivos: el maestro y otro auxiliar. El maestro lo abriremos para lectura, y el auxiliar para escritura. Iremos transfiriendo todos los registros del maestro al auxiliar excepto el que queramos borrar. A continuación, borraremos el maestro, y asignaremos al auxiliar el nombre del maestro.
* Modificaciones: Procederemos de igual forma que para las bajas, salvo que transferiremos todos los registros al auxiliar, los no modificados y el modificado. Borraremos el maestro y renombraremos el auxiliar para darle el nombre del maestro.
* Consultas: Se abrirá el archivo en modo lectura.
* Listado por pantalla: Se abrirá el archivo en modo lectura. Se irá leyendo información mientras no se llegue al final del archivo.
* Listado por impresora: Utilizaremos dos archivos, el maestro y el de impresora. El primero lo abriremos para lectura, y el segundo para escritura, del siguiente modo:

```
FILE *f, *fimp;

f = fopen ("clientes.dat", "r");
fimp = fopen ("prn", "w");
```

De tal forma que todo lo que escribamos en fimp saldrá por impresora. Por tal motivo, debemos pensar que se trata de un archivo de texto, por lo cual, usaremos funciones de escritura en archivos de texto.

5. Acceso directo a archivos

Ya hemos visto como acceder secuencialmente a un archivo, sin embargo también se puede hacer de forma directa.

Supongamos que tenemos definido un archivo con la siguiente estructura de registro:

```
struct
   {
   int codigo;
   char nomart[31];
   float precio;
   } articulo;
```

Es evidente que la longitud de cada registro es de 37 bytes (2+31+4 bytes). De esta forma, la disposición de los registros dentro del archivo en disco se realiza en las siguientes posiciones:

NOTA: Este documento fue extraído de una publicación de la Universidad Tecnológica Nacional Facultad Regional Avellaneda.

EJEMPLOS Y EJERCICIOS

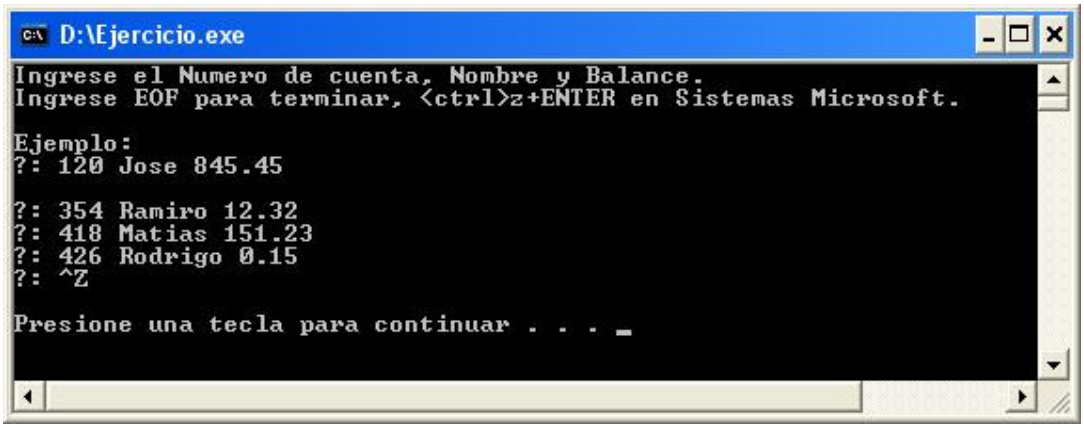
EJEMPLO 1: Escriba un programa que me permita cargar "Numero de Cuenta, Nombre, Balance" de un cliente de un banco. Esos datos serán guardados en la Unidad C, en un archivo llamado "clientes.dat".

```
#include <stdio.h>

int main()
{
int cuenta;
char nombre[30];
float balance;
FILE *P; // Puntero al archivo Clientes.dat

P = fopen("c:clientes.dat", "w");
if (P == NULL)
      printf ("El archivo no pudo abrirse!");
else
     {
     printf ("Ingrese el Numero de cuenta, Nombre y Balance.\n");
     printf ("Ingrese EOF para terminar, <ctrl>z+ENTER en Sistemas Microsoft.\n");
     printf ("\nEjemplo:\n?: 120 Jose 845.45\n\n?: ");
     scanf ("%d%s%f", &cuenta, nombre, &balance);

     while ( !feof(stdin) )
           {
           fprintf (P, "%d %s %.2f\n", cuenta, nombre, balance);
           printf ("?: ");
           scanf ("%d%s%f", &cuenta, nombre, &balance);
           }
     fclose (P);
     }
printf ("\n");
system("PAUSE");
return 0;
}
```

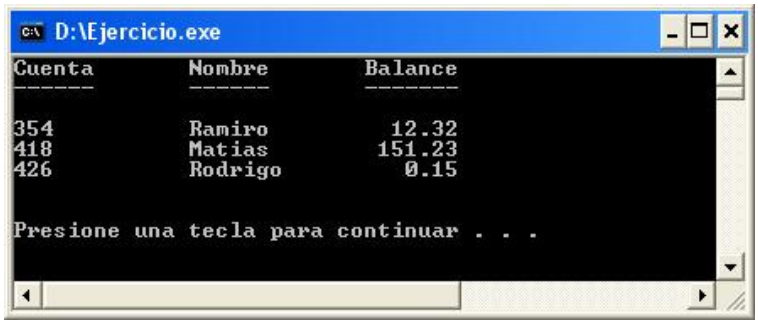


EJERCICIO 1: Escriba un programa que me permita leer del archivo "clientes.dat" los datos que tiene cargados, que son "Numero de Cuenta, Nombre, Balance" de un cliente de un banco.

```
#include <stdio.h>

int main()
{
int cuenta;
char nombre[30];
float balance;
FILE *P; // Puntero al archivo Clientes.dat

P = fopen("c:clientes.dat", "r");
.......
```



EJERCICIO 2: Escriba un programa que me permita, mediante un menú interactivo, listar los datos cargados en el archivo "clientes.dat", filtrando por balance (Cero, Negativo, Positivo).

```
#include <stdio.h>

int main()
{
int consulta, cuenta;
char nombre[30];
float balance;
FILE *P; // Puntero al archivo Clientes.dat

P = fopen("c:clientes.dat", "r");
.......
```