Archivos 3 - Archivos de Acceso Aleatorio

Los archivos de acceso aleatorio son más versátiles, permiten acceder a cualquier parte del fichero en cualquier momento, como si fueran arrays en memoria. Las operaciones de lectura y/o escritura pueden hacerse en cualquier punto del archivo. Éste tipo de archivos de acceso aleatorio se denominan binarios y no podemos ver el contenido que hay en él a través de un bloc de notas, ya que los datos se encuentran expresados en lenguaje máquina.

Características de su implementación en C:

1. Apertura en modo binario (b).
2. Uso de las funciones pertinentes al trabajo con archivos binarios (acceso aleatorio), algunas son:
    a. fread y fwrite, que permiten leer y escribir registros de longitud constante desde y hacia un fichero binario.
    b. fseek para situar el puntero de lectura/escritura en el lugar apropiado de tu archivo.
3. Se puede abrir el archivo en un modo que te permita leer y escribir sobre ese mismo puntero a la estructura de archivo (FILE*). Esto no es imprescindible, es posible usar archivos de acceso aleatorio sólo de lectura o de escritura

Por ejemplo, supongamos que nuestros registros tienen la siguiente estructura:

```
typedef struct {
    char Nombre[34];
    int dato;
}stRegistro;
```

Teniendo en cuenta que los registros empiezan a contarse desde el cero, para hacer una lectura del registro número 6 usaremos:

```
FILE * fichero;
stRegistro reg;
fseek (fichero, 5*sizeof (stRegistro), SEEK_SET);
fread (&reg, sizeof (stRegistro), 1, fichero);
```

Análogamente, para hacer una operación de escritura, usaremos:

```
fseek (fichero, 5 * sizeof (stRegistro), SEEK_SET);
fwrite (&reg, sizeof (stRegistro), 1, fichero);
```

Muy importante: después de cada operación de lectura o escritura, el cursor del fichero se actualiza automáticamente a la siguiente posición, por lo tanto si estamos haciendo inserciones o lecturas de manera ordenada es buena idea hacer siempre un fseek() antes de un fread() o un fwrite().

Calcular la longitud de un fichero

Para calcular el tamaño de un fichero, ya sea en bytes o en cantidad de registros se suele usar el siguiente procedimiento:

```
int nRegistros;
int nBytes;
fseek (fichero, 0, SEEK_END); // Colocar el cursor al final del fichero
nBytes = ftell (fichero); // Tamaño en bytes
nRegistros = ftell (fichero) / sizeof (stRegistro); // Tamaño en nº
registros
```

Borrar registros

Borrar registros puede ser complicado, ya que no hay ninguna función de librería estándar que lo haga. Es su lugar, se suele usar uno de estos dos métodos para eliminar registros:

... Para eliminar un registro de un archivo deberías copiar todo el contenido del archivo en otro auxiliar, sin incluir el registro a borrar. Una vez hayas hecho esto copias el archivo auxiliar sobre el original y ya habrás eliminado el registro que deseabas quitar. De todos modos este proceso puede ser algo lento. Por eso existe el "borrado lógico", que consiste en añadir un campo a cada registro que indique si está borrado o no. De este modo, solo debes trabajar con los "no borrados". Así, borrar uno es algo rápido, aunque se desaprovecha espacio de disco. Esto se resolvería añadiendo al programa una opción de "compactado" del archivo, que elimina todos los que están marcados según el algoritmo indicado al principio.

De tal manera, el usuario escoge el momento de borrar los registros... cosa que para grandes ficheros podría ser muy útil, ya que dicho proceso podría activarse al comenzar el trabajo del sistema... por ejemplo.

Como idea, para evitar al usuario tener que pulsar esa opción sin saber muy bien por qué... puedes hacer que el borrado físico (real) se haga cada vez que el programa arranque, por ejemplo.

Primer Método:

Marcar el registro como borrado o no válido, para ello hay que añadir un campo extra en la estructura del registro:

```
typedef struct {
    char valido; // Campo que indica si el registro es válido
    char nombre[34];
    int dato;
}stRegistro;
```

Si el campo **valido** tiene un valor prefijado, por ejemplo 'S' o ' ', el registro es válido. Si tiene un valor prefijado, por ejemplo 'N' o '*', el registro será inválido o se considerará borrado.

Esto funciona como una “bandera ó flag”, le podríamos asignar valores 1 y 0 (uno y cero), lo que nos resulte más amigable.

De este modo, para borrar un registro sólo se debe cambiar el valor de ese campo.

Pero hay que tener en cuenta que será el programa el encargado de tratar los registros del modo adecuado dependiendo del valor del campo **valido**, el hecho de marcar un registro no lo borra físicamente.

Si se quiere elaborar más, se puede mantener un fichero auxiliar con la lista de los registros borrados. Esto tiene un doble propósito:

- Que se pueda diseñar una función para sustituir a fseek() de modo que se tengan en cuenta los registros marcados.
- Que al insertar nuevos registros, se puedan sobrescribir los anteriormente marcados como borrados, si existe alguno.

Segundo Método:

Hacer una copia del fichero en otro fichero, pero sin copiar el registro que se quiere borrar. Este sistema es más tedioso y lento, y requiere cerrar el fichero y borrarlo o renombrarlo, antes de poder usar de nuevo la versión con el registro eliminado.

Lo normal es hacer una combinación de ambos, durante la ejecución normal del programa se borran registros con el método de marcarlos, y cuando se cierra la aplicación, o se detecta que el porcentaje de registros borrados es alto o el usuario así lo decide, se "empaqueta" el fichero usando el segundo método.

Ejemplo: A continuación se incluye un ejemplo de un programa que trabaja con registros de acceso aleatorio, es un poco largo, pero bastante completo:

```
//######## aleator.c → Ejemplo de ficheros de acceso aleatorio. ##########

//----------------------- Headers o Cabeceras ----------------------------
```

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

//-------------------- Declaración de estructuras -------------------------
typedef struct {
    char valido; // Campo que indica si el registro es válido 'S' → 
Válido, 'N' → Inválido
    char nombre[34];
    int dato;
} stRegistro;

//--------------------- Prototipado de las funciones ----------------------
char Menu();
void Leer(stRegistro * reg);
void Mostrar(stRegistro * reg);
void Listar(int n, stRegistro * reg);
void Empaquetar(FILE **fa); // Uso un puntero a puntero (doble puntero), 
para no retornar un puntero desde la función (otra solución)

//--------- Cuerpo principal o función principal del programa ------------
int main() {
    FILE * fa;
    stRegistro reg;
    char opcion;
    int numero;
    fa = fopen ("alea.dat", "a+b"); // Este modo permite leer y escribir
    do {
        switch (opcion = Menu ()) {
            case '1': // Añadir registro
                Leer (&reg); fseek (fa, 0, SEEK_END); // Insertar al final
                fwrite (&reg, sizeof (stRegistro), 1, fa);
                break;
            case '2': // Mostrar registro
                system ("cls");
                printf ("Mostrar registro: ");
                scanf ("%d", &numero);
                fseek (fa, numero * sizeof (stRegistro), SEEK_SET);
                fread (&reg, sizeof (stRegistro), 1, fa);
                Mostrar (&reg);
                break;
            case '3': // Eliminar registro
                system ("cls");
```

```
                printf ("Eliminar registro: ");
                scanf ("%d", &numero);
                fseek (fa, numero * sizeof (stRegistro), SEEK_SET);
                fread (&reg, sizeof (stRegistro), 1, fa);
                reg.valido = 'N';
                fseek (fa, numero * sizeof (stRegistro), SEEK_SET);
                fwrite (&reg, sizeof (stRegistro), 1, fa);
                break;
            case '4': // Mostrar todo
                rewind (fa);
                numero = 0;
                system ("cls");
                printf ("Numero Nombre Datos\n");
                while (fread(&reg, sizeof (stRegistro), 1, fa) > 0) {
                    Listar (numero++, &reg);
                }
                system ("pause");
                break;
            case '5': // Eliminar marcados
                Empaquetar (&fa);
                break;
        }
    }while (opcion != '0');
    fclose (fa);
    return 0;
}

//---------- Muestra un menú y captura una opción del usuario ------------
char Menu (){
    char resp;
    do {
        system ("cls");
        printf ("\n\n\n\t\t\tMENU PRINCIPAL\n");
        printf ("\n\t[ 1 ] - Insertar registro");
        printf ("\n\t[ 2 ] - Mostrar registro");
        printf ("\n\t[ 3 ] - Eliminar registro");
        printf ("\n\t[ 4 ] - Mostrar todo");
        printf ("\n\t[ 5 ] - Eliminar registros marcados");
        printf ("\n\t[ 0 ] - Salir");
        fflush (stdin);
        resp = getche ();
    } while (resp < '0' && resp > '5');
    return resp; }
```

```c
//------ Permite que el usuario introduzca un registro por pantalla -------
void Leer (stRegistro * reg) {
    system ("cls");
    printf ("Leer registro:\n\n");
    reg -> valido = 'S'; // Trabajo con un puntero a un registro, entonces cambio el operador punto (.) por el operador flecha (->) para acceder a los campos de la estructura.
    printf ("Nombre: ");
    fflush (stdin);
    gets ((*reg).nombre);
    printf ("Dato: ");
    scanf ("%d", &(*reg).dato);
}

//--- Muestra un registro por pantalla, si no está marcado como borrado ---
void Mostrar (stRegistro * reg) {
    system ("cls");
    if ((*reg).valido == 'S') {
        printf ("Nombre: %s\n", (*reg).nombre);
        printf ("Dato: %d\n", (*reg).dato);
    } system ("PAUSE");
}

// Muestra los registros por pantalla, en forma de listado, si no están marcados como borrados
void Listar (int n, stRegistro * reg) {
    if ((*reg).valido == 'S') {
        printf (" %4d ", n);
        printf (" %-34s", (*reg).nombre);
        printf (",%4d\n", (*reg).dato);
    }
}

//---------- Elimina los registros marcados como borrados ----------------
void Empaquetar (FILE **fa) { // Uso un puntero a puntero (doble puntero), para no retornar un  puntero desde la función (sería la otra manera de hacerlo).
    FILE *ftemp;
    stRegistro reg;
    ftemp = fopen ("alea.tmp", "wb");
    rewind (*fa);
    while (fread (&reg, sizeof (stRegistro), 1, *fa) > 0) {
```

```
        if (reg.valido == 'S') {
            fwrite (&reg, sizeof (struct stRegistro), 1, ftemp);
        }
    }
    fclose (ftemp);
    fclose (*fa); // Lo cierro para poder trabajar en:
    remove ("alea.bak"); // Un poquito de administración de archivos desde
consola DOS
    rename ("alea.dat", "alea.bak");
    rename ("alea.tmp", "alea.dat");
    *fa = fopen ("alea.dat", "r+b"); // y lo abro, pues debo devolver la
ejecución con el archivo abierto.
}
//________ F I N ________ Fuente de inpiración: http://www.conclase.net
```