1º ASI – Fundamentos de Programación

Ejercicios del Tema 4 - Archivos Secuenciales: LIGA DE FÚTBOL

Completar el ejercicio de la liga de fútbol programando un sencillo menú de opciones que tenga este aspecto en pantalla:

```
PROGRAMA DE LA LIGA DE FÚTBOL
MENÚ DE OPCIONES
1. Introducir datos
2. Mostrar datos
3. Ordenar datos
4. Buscar un equipo
5. Borrar un equipo
6. Modificar un equipo
7. Salir del programa
```

Después, y según la opción elegida por el usuario, se debe llamar a una función por cada opción.

- **Introducir datos**: esta función permitirá al usuario introducir por teclado nuevos datos, que serán añadidos al fichero de datos (liga.dat)
- **Mostrar datos**: leerá el archivo de datos (liga.dat) y mostrará su contenido en la pantalla.
- **Ordenar datos**: sirve para ordenar el archivo de datos por orden decreciente de puntuación.
- **Buscar un equipo**: pide al usuario un nombre de equipo y lo busca en el archivo. Si lo encuentra, muestra sus datos por la pantalla.
- **Borrar un equipo**: pide al usuario un nombre de equipo y, si existe, lo borra del archivo.
- **Modificar un equipo**: pide al usuario un nombre de equipo y, si existe, muestra por la pantalla sus datos actuales y pide al usuario unos datos nuevos. Después, sustituye los datos del equipo por los nuevos y lo guarda todo en el archivo.

SOLUCIÓN 1

La primera solución que proponemos consiste en mantener los datos del programa en la memoria secundaria (es decir, en un archivo). En la memoria principal sólo se cargará un registro cada vez: el que se esté procesando en ese momento.

Ventajas de esta solución:

- Todas las operaciones que hacemos sobre los datos quedan **guardadas en el archivo de disco casi instantáneamente**. Si el programa "se cuelga", o si se va la luz o algo parecido, no perderemos los datos.
- Como los datos se almacenan en la memoria secundaria, tenemos una **capacidad de almacenamiento muy grande**. Es decir, el programa puede manejar cantidades enormes de información (tanta como capacidad tenga el disco duro o el dispositivo donde se almacene el archivo de datos)

Inconvenientes de esta solución:

- El funcionamiento es **más lento**, ya que se accede continuamente a la memoria secundaria.
- Las operaciones de **borrado, modificación y ordenación plantean problemas** de implementación con archivos secuenciales. Cualquier solución que adoptemos será siempre muy lenta.

```
#include <stdio.h>
#define ARCHIVO_DATOS "liga.dat"    // Nombre del archivo de datos

// Prototipos de funciones
void introducir_datos();
void mostrar_datos();
void ordenar_datos();
int contar_equipos();
void buscar_equipo();
void borrar_equipo();
void modificar_equipo();

// Estructura de datos de cada equipo
struct s_equipo
{
   char nombre[50];
   int jug, gan, per, emp;
   int puntos;
};

int main(void)
{
```

```
   int opc;
   char txt[50];

   do
   {
      // Mostrar el menú de opciones
      printf("\n\nPROGRAMA DE LA LIGA DE FUTBOL\n\n");
      printf("MENU DE OPCIONES\n\n");
      printf("1 - Introducir datos\n");
      printf("2 - Mostrar datos\n");
      printf("3 - Ordenar datos\n");
      printf("4 - Buscar un equipo\n");
      printf("5 - Borrar un equipo\n");
      printf("6 - Modificar un equipo\n");
      printf("7 - Salir del programa\n");

      // Leer la opción seleccionada por el usuario
      do
      {
         printf("\nElija una opción (1-7): ");
         gets(txt);
         opc = atoi(txt);
      }
      while ((opc < 1) || (opc > 7));

      // Llamar a la función correspondiente según la opción elegida
      switch (opc)
      {
         case 1: introducir_datos(); break;
         case 2: mostrar_datos(); break;
         case 3: ordenar_datos(); break;
         case 4: buscar_equipo(); break;
         case 5: borrar_equipo(); break;
         case 6: modificar_equipo(); break;
      }
   }
   while (opc != 7);

   return 0;
}


// Leer por teclado los datos de un equipo y añadirlo al archivo de datos
void introducir_datos()
{
   FILE *f;
   struct s_equipo equipo;
   char aux[50];


   // Leer datos del equipo por teclado
   printf("Introduzca los datos del equipo.\n");
   printf("  Nombre: ");
   gets(equipo.nombre);
   printf("  Jugados: ");
   gets(aux);
   equipo.jug = atoi(aux);
   printf("  Ganados: ");
   gets(aux);
   equipo.gan = atoi(aux);
   printf("  Perdidos: ");
   gets(aux);
   equipo.per = atoi(aux);
   equipo.emp = equipo.jug - equipo.gan - equipo.per;
   equipo.puntos = equipo.gan * 3 + equipo.emp * 1;

   // Abrir el archivo en modo "ab" (añadir/binario) y grabar los datos del equipo
   f = fopen(ARCHIVO_DATOS, "ab");
   if (f == NULL)
      printf("Error al abrir el archivo %s. No se pueden guardar los datos\n", ARCHIVO_DATOS);
   else {
      fwrite(&equipo, sizeof(struct s_equipo), 1, f);
      fclose(f);
   }
}

// Lee los datos del archivo de datos y los muestra por la pantalla
void mostrar_datos()
{
   FILE *f;
```

```
   struct s_equipo equipo;
   int result;

   // Abrir el archivo para lectura
   f = fopen(ARCHIVO_DATOS, "rb");
   if (f == NULL) {
     printf("Error al abrir el archivo %s\n", ARCHIVO_DATOS);
     return;
   }

   // Leer datos y mostrarlos en la pantalla, hasta que se alcance el EOF
   printf("\nEQUIPO              JUG GAN EMP PER PUNTOS\n\n");
   while (!feof(f))
   {
      fread(&equipo, sizeof(struct s_equipo), 1, f);
      printf("%-20s %2i  %2i  %2i  %2i   %2i\n", equipo.nombre, equipo.jug, equipo.gan,
                equipo.emp, equipo.per, equipo.puntos);
   }
   fclose(f);
}


// Ordena el archivo de datos por puntuación. Usaremos el método de la burbuja
// Habrá que recorrer el archivo N veces intercambiando los equipos adyacentes que
// estén desordenados, siendo N el nº de equipos que hay en el archivo.
void ordenar_datos()
{
    FILE *f, *f_temp;
    int N, i;
    struct s_equipo eq1, eq2;

    printf("Ordenando el archivo...\n");
    N = contar_equipos();                    // Cuenta en nº de equipos que hay en el archivo

    for (i=0; i<N; i++)                     // Repetiremos el proceso N veces
    {
        // Abrir archivo de datos para lectura
        f = fopen(ARCHIVO_DATOS, "rb");
        if (f == NULL) {
          printf("No se puede abrir el archivo de datos %s\n", ARCHIVO_DATOS);
          return;
        }

        // Crear archivo auxiliar para realizar el intercambio de elementos
        f_temp = fopen("temporal", "wb");
        if (f_temp == NULL) {
          printf("Error al crear archivo temporal\n");
          fclose(f);
          return;
        }

        // Recorrer el archivo intercambiando elementos adyacentes (si están desordenados)
        fread(&eq1, sizeof(struct s_equipo), 1, f);
        while (!feof(f))
        {
           // Leemos el siguiente registro (eq2) para compararlo con el anterior (eq1)
           fread(&eq2, sizeof(struct s_equipo), 1, f);

           if (eq1.puntos > eq2.puntos)          // Están ordenados (ORDENACIÓN POR PUNTOS)
           {                                     // Copiamos el equipo eq1 al archivo temporal
               fwrite(&eq1, sizeof(struct s_equipo), 1, f_temp);
               eq1 = eq2;                        // Preparamos la próxima iteración
           }
           else       // Están desordenados. Hay que intercambiarlos
           {          // Escribimos eq2 en lugar de eq1 en el archivo temporal
               fwrite(&eq2, sizeof(struct s_equipo), 1, f_temp);
           }
        }
        fwrite(&eq2, sizeof(struct s_equipo), 1, f_temp);
        fclose(f);
        fclose(f_temp);

        // Convertimos el archivo temporal en el nuevo archivo de datos
        remove(ARCHIVO_DATOS);
        rename("temporal", ARCHIVO_DATOS);
   } // Fin del "for" (se repite N veces)

   printf("Ordenación terminada.\n");
}
```

```
// Devuelve el número de equipos que hay en el archivo de datos
int contar_equipos()
{
   int cont;
   FILE* f;
   struct s_equipo eq;

   // Abrir archivo de datos para lectura
   f = fopen(ARCHIVO_DATOS, "rb");
   if (f == NULL) {
      printf("No se puede abrir el archivo de datos %s\n", ARCHIVO_DATOS);
      return -1;
   }

   cont = 0;
   while (!feof(f))
   {
       fread(&eq, sizeof(struct s_equipo), 1, f);
       cont++;
   }

   return cont;
}

// Pide por teclado un nombre de equipo y lo busca en el archivo de datos
// Si lo encuentra, muestra sus datos en la pantalla
void buscar_equipo()
{
   FILE *f;
   struct s_equipo equipo;
   char nom_eq[50];
   int encontrado;

   // Leer nombre del equipo que se pretende buscar
   printf("Introduzca nombre del equipo: ");
   gets(nom_eq);

   // Abrir archivo de datos para lectura
   f = fopen(ARCHIVO_DATOS, "rb");
   if (f == NULL) {
      printf("No se puede abrir el archivo de datos %s\n", ARCHIVO_DATOS);
      return;
   }

   // Recorrer el archivo buscando el equipo
   encontrado = 0;
   while (!feof(f))
   {
      fread(&equipo, sizeof(struct s_equipo), 1, f);
      if (strcmp(equipo.nombre, nom_eq) == 0)      // Hemos encontrado el equipo
      {
         encontrado = 1;
         break;
      }
   }
   fclose(f);

   // Mostrar resultado en la pantalla
   if (encontrado == 1)
   {
      printf("Nombre: %s\n", equipo.nombre);
      printf("Puntos: %i\n", equipo.puntos);
      printf("Ganados: %i - Empatados: %i - Perdidos: %i\n", equipo.gan, equipo.emp, equipo.per);
   }
   else
      printf("Equipo no encontrado");
}


// Pide por teclado un nombre de equipo y lo busca en el archivo de datos
// Si lo encuentra, lo borra del archivo.
void borrar_equipo()
{
   FILE *f, *f_temp;
   struct s_equipo equipo;
   char nom_eq[50];
   int encontrado;
```

```
   // Leer nombre del equipo que se pretende borrar
   printf("Introduzca nombre del equipo que quiere borrar: ");
   gets(nom_eq);

   // Abrir archivo de datos para lectura
   f = fopen(ARCHIVO_DATOS, "rb");
   if (f == NULL) {
      printf("No se puede abrir el archivo de datos %s\n", ARCHIVO_DATOS);
      return;
   }

   // Crear archivo auxiliar para realizar el borrado
   f_temp = fopen("temporal", "wb");
   if (f_temp == NULL) {
       printf("Error al crear archivo temporal\n");
       fclose(f);
       return;
   }

   // Recorrer el archivo copiando todos los registros, menos el que se quiere borrar
   encontrado = 0;
   while (!feof(f))
   {
      fread(&equipo, sizeof(struct s_equipo), 1, f);
      if (strcmp(equipo.nombre, nom_eq) == 0)      // Hemos encontrado el equipo
         encontrado = 1;
      else
         fwrite(&equipo, sizeof(struct s_equipo), 1, f_temp);
   }
   fclose(f);
   fclose(f_temp);

   // Mostrar resultado y eliminar archivo temporal
   if (encontrado == 1)
   {
      printf("Equipo borrado");
      remove(ARCHIVO_DATOS);
      rename("temporal", ARCHIVO_DATOS);
   }
   else
   {
      printf("Equipo no encontrado");
      remove("temporal");
   }
}


// Pide por teclado un nombre de equipo y lo busca en el archivo de datos
// Si lo encuentra, muestra sus datos en la pantalla y pide al usuario
// que los modifique. Luego guarda los datos modificados en el archivo.
void modificar_equipo()
{
   FILE *f, *f_temp;
   struct s_equipo equipo, nuevo;
   char nom_eq[50], aux[50];
   int encontrado;

   // Leer nombre del equipo que se pretende modificar
   printf("Introduzca nombre del equipo que quiere modificar: ");
   gets(nom_eq);

   // Abrir archivo de datos para lectura
   f = fopen(ARCHIVO_DATOS, "rb");
   if (f == NULL) {
      printf("No se puede abrir el archivo de datos %s\n", ARCHIVO_DATOS);
      return;
   }

   // Crear archivo auxiliar para realizar la modificación
   f_temp = fopen("temporal", "wb");
   if (f_temp == NULL) {
       printf("Error al crear archivo temporal\n");
       fclose(f);
       return;
   }

   // Recorrer el archivo copiando todos los registros, menos el que se quiere modificar
   encontrado = 0;
   while (!feof(f))
   {
```

```
      fread(&equipo, sizeof(struct s_equipo), 1, f);
      if (strcmp(equipo.nombre, nom_eq) == 0)      // Hemos encontrado el equipo
      {
        encontrado = 1;
        // Mostramos sus datos actuales
        printf("Equipo encontrado. Sus datos actuales son:");
        printf("Nombre: %s\n", equipo.nombre);
        printf("Jugados = %i, Puntos = %i\n", equipo.jug, equipo.puntos);
        printf("Ganados = %i, Empatados = %i, Perdidos = %i\n\n",
                                                    equipo.gan, equipo.emp, equipo.per);
        // Pedimos al usuario que introduzca los datos nuevos
        printf("Introduzca los datos nuevos:\n");
        printf("Nombre: ");
        gets(nuevo.nombre);
        printf("Partidos jugados: ");
        gets(aux); nuevo.jug = atoi(aux);
        printf("Partidos ganados: ");
        gets(aux); nuevo.gan = atoi(aux);
        printf("Partidos perdidos: ");
        gets(aux); nuevo.per = atoi(aux);
        nuevo.emp = nuevo.jug - nuevo.per - nuevo.gan;
        nuevo.puntos = nuevo.gan * 3 + nuevo.emp;
        // Grabamos los datos nuevos
        fwrite(&nuevo, sizeof(struct s_equipo), 1, f_temp);
      }
      else
        fwrite(&equipo, sizeof(struct s_equipo), 1, f_temp);
   }
   fclose(f);
   fclose(f_temp);

   // Mostrar resultado y eliminar archivo temporal
   if (encontrado == 1)
   {
      printf("Equipo modificado");
      remove(ARCHIVO_DATOS);
      rename("temporal", ARCHIVO_DATOS);
   }
   else
   {
      printf("Equipo no encontrado");
      remove("temporal");
   }

}
```

SOLUCIÓN 2

La segunda solución que proponemos consiste en **cargar todos los datos en la memoria principal** al comenzar el programa, y operar exclusivamente con esos datos. Al finalizar el programa, volveremos a escribir todos los datos en el archivo de memoria secundaria.

Ventajas de esta solución:

- La ejecución es **más rápida**, ya que los datos que se manipulan están en la memoria principal. Sólo se accede a memoria secundaria dos veces: al empezar y al terminar la ejecución.
- Las operaciones **de borrado, modificación y ordenación son muy sencillas** de realizar, ya que se trata de operaciones sobre vectores. El borrado lo haremos añadiendo una marca a cada registro.

Inconvenientes de esta solución:

- **Si el programa se detiene** por alguna razón antes de terminar (corte de luz, colapso del sistema, etc), **perderemos todas las modificaciones** que hayamos hecho sobre los datos, ya que no se habrán escrito en el archivo.
- Al cargar los datos en un vector tenemos una importante **limitación de espacio**. Si el archivo es muy grande, los datos sencillamente **no cabrán en la memoria principal**. Si utilizásemos memoria dinámica la situación mejoraría algo, pero también podría desbordarse.

```
#include <stdio.h>
#define ARCHIVO_DATOS "liga.dat"    // Nombre del archivo de datos
#define MAX_EQUIPOS 20              // Nº máximo de equipos que puede manejar el programa

// Estructura de datos de cada equipo
struct s_equipo
{
   char nombre[50];
   int jug, gan, per, emp;
```

```
   int puntos;
   char borrado;
};

// Prototipos de funciones
int cargar_datos(struct s_equipo equipos[MAX_EQUIPOS]);
void grabar_datos(struct s_equipo equipos[MAX_EQUIPOS], int num_eq);
void introducir_datos(struct s_equipo equipos[MAX_EQUIPOS], int num_eq);
void mostrar_datos(struct s_equipo equipos[MAX_EQUIPOS], int num_eq);
void ordenar_datos(struct s_equipo equipos[MAX_EQUIPOS], int num_eq);
void buscar_equipo(struct s_equipo equipos[MAX_EQUIPOS], int num_eq);
void borrar_equipo(struct s_equipo equipos[MAX_EQUIPOS], int num_eq);
void modificar_equipo(struct s_equipo equipos[MAX_EQUIPOS], int num_eq);

int main(void)
{
   int opc;
   char txt[50];
   int num_eq;                              // Nº de equipos que hay almacenados
   struct s_equipo equipos[MAX_EQUIPOS];    // Array donde almacenaremos los datos

   num_eq = cargar_datos(equipos);          // Cargar datos desde archivo al array

   do
   {
      // Mostrar el menú de opciones
      printf("\n\nPROGRAMA DE LA LIGA DE FUTBOL\n\n");
      printf("MENU DE OPCIONES\n\n");
      printf("1 - Introducir datos\n");
      printf("2 - Mostrar datos\n");
      printf("3 - Ordenar datos\n");
      printf("4 - Buscar un equipo\n");
      printf("5 - Borrar un equipo\n");
      printf("6 - Modificar un equipo\n");
      printf("7 - Salir del programa\n");

      // Leer la opción seleccionada por el usuario
      do
      {
         printf("\nElija una opción (1-7): ");
         gets(txt);
         opc = atoi(txt);
      }
      while ((opc < 1) || (opc > 7));

      // Llamar a la función correspondiente según la opción elegida
      switch (opc)
      {
         case 1: if (num_eq < MAX_EQUIPOS-1) {
                     introducir_datos(equipos, num_eq);
                     num_eq++;
                 }
                 else
                     printf("No hay memoria para añadir más equipos\n");
                 break;
         case 2: mostrar_datos(equipos, num_eq);
                 break;
         case 3: ordenar_datos(equipos, num_eq);
                 break;
         case 4: buscar_equipo(equipos, num_eq);
                 break;
         case 5: borrar_equipo(equipos, num_eq);
                 break;
         case 6: modificar_equipo(equipos, num_eq);
                 break;
      }
   }
   while (opc != 7);

   grabar_datos(equipos, num_eq);         // Grabar datos del array en el archivo

   return 0;
}


// Lee los datos grabados en el archivo de disco y los carga en la memoria
// principal (en un array de estructuras). Devuelve el número de equipos leidos del archivo.
int cargar_datos(struct s_equipo equipos[MAX_EQUIPOS])
{
   int num_regs;
```

```
   FILE *f;

   // Abrimos el archivo para lectura
   f = fopen(ARCHIVO_DATOS, "rb");
   if (f == NULL) {
      // Si el archivo no existe, lo creamos
      printf("El archivo %s no existe. Se creará uno vacío.\n", ARCHIVO_DATOS);
      f = fopen(ARCHIVO_DATOS, "wb");
      if (f == NULL) {
          printf("Error al crear el archivo. El programa no puede continuar.\n");
          exit(1);
      }
   }

   // Leemos del archivo de datos un máximo de MAX_EQUIPOS equipos
   // fread() nos devolverá el número de registros leídos realmente
   // Si el archivo se acaba de crear, ese número será 0
   num_regs = fread(equipos, sizeof(struct s_equipo), MAX_EQUIPOS, f);
   fclose(f);

   return num_regs;   // Devolvemos el nº de registros leídos
}

// Escribe los datos de la memoria principal (almacenados en un array de estructuras)
// en el archivo de datos
void grabar_datos(struct s_equipo equipos[MAX_EQUIPOS], int num_eq)
{
   FILE *f;

   // Abrir el archivo para escritura, sobreescribiendo lo que hubiera antes
   f = fopen(ARCHIVO_DATOS, "wb");
   if (f == NULL)
      printf("Error al abrir el archivo %s. ¡No se pueden guardar los datos!\n", ARCHIVO_DATOS);
   else {
      // Escribimos "num_eq" registros
      fwrite(equipos, sizeof(struct s_equipo), num_eq, f);
      fclose(f);
   }
}

// Leer por teclado los datos de un equipo y añadirlo al array de datos
void introducir_datos(struct s_equipo equipos[MAX_EQUIPOS], int num_eq)
{
   char aux[50];

   // Leer datos del equipo por teclado y almacenarlos en la posición "num_eq" del array
   printf("Introduzca los datos del equipo.\n");
   printf("  Nombre: ");
   gets(equipos[num_eq].nombre);
   printf("  Jugados: ");
   gets(aux);
   equipos[num_eq].jug = atoi(aux);
   printf("  Ganados: ");
   gets(aux);
   equipos[num_eq].gan = atoi(aux);
   printf("  Perdidos: ");
   gets(aux);
   equipos[num_eq].per = atoi(aux);
   equipos[num_eq].emp = equipos[num_eq].jug - equipos[num_eq].gan - equipos[num_eq].per;
   equipos[num_eq].puntos = equipos[num_eq].gan * 3 + equipos[num_eq].emp * 1;

   // Ponemos el campo "borrado" a "N" (o sea, "no")
   equipos[num_eq].borrado = 'N';
}

// Recorre los datos del array de datos y los muestra por la pantalla
void mostrar_datos(struct s_equipo equipos[MAX_EQUIPOS], int num_eq)
{
   int i;

   printf("\nEQUIPO              JUG GAN EMP PER PUNTOS\n\n");
   // Recorremos el array de datos y los mostramos en la pantalla
   for (i = 0; i < num_eq; i++)
   {
      // Mostramos los datos del equipo sólo si no ha sido borrado
      if (equipos[i].borrado != 'S')
         printf("%-20s %2i  %2i  %2i  %2i   %2i\n", equipos[i].nombre, equipos[i].jug,
equipos[i].gan,
              equipos[i].emp, equipos[i].per, equipos[i].puntos);
   }
```

```
}


// Ordena el array de datos por puntuación. Usaremos el método de la burbuja.
void ordenar_datos(struct s_equipo equipos[MAX_EQUIPOS], int num_eq)
{
    int i, j;
    struct s_equipo aux;

    for (i = 1; i < num_eq; i++)
    {
        for (j = num_eq - 1; j >= i; j--)
        {
           if (equipos[j-1].puntos < equipos[j].puntos)     // Intercambiar
           {
               aux = equipos[j-1];
               equipos[j-1] = equipos[j];
               equipos[j] = aux;
           }
        }
    }
}


// Pide por teclado un nombre de equipo y lo busca en el array de datos
// Si lo encuentra, muestra sus datos en la pantalla
void buscar_equipo(struct s_equipo equipos[MAX_EQUIPOS], int num_eq)
{
   char nombre_buscado[50];
   int i, encontrado;

   // Leer nombre del equipo que se pretende buscar
   printf("Introduzca nombre del equipo: ");
   gets(nombre_buscado);

   // Recorrer el array buscando el equipo
   encontrado = 0;
   for (i=0; i<num_eq; i++)
   {
      if ((strcmp(equipos[i].nombre, nombre_buscado) == 0) &&     // Hemos encontrado el equipo
          (equipos[i].borrado != 'S'))                            // y no ha sido borrado
      {
         encontrado = 1;
         break;
      }
   }

   // Mostrar resultado en la pantalla
   if (encontrado == 1)
   {
      printf("Nombre: %s\n", equipos[i].nombre);
      printf("Puntos: %i\n", equipos[i].puntos);
      printf("Ganados: %i - Empatados: %i - Perdidos: %i\n",
                           equipos[i].gan, equipos[i].emp, equipos[i].per);
   }
   else
      printf("Equipo no encontrado\n");
}


// Pide por teclado un nombre de equipo y lo busca en el array de datos
// Si lo encuentra, lo marca como borrado para luego no escribirlo en el archivo.
void borrar_equipo(struct s_equipo equipos[MAX_EQUIPOS], int num_eq)
{
   char nombre_buscado[50];
   int i, encontrado;

   // Leer nombre del equipo que se pretende borrar
   printf("Introduzca nombre del equipo que quiere borrar: ");
   gets(nombre_buscado);

   // Recorrer el array buscando el equipo que se quiere borrar
   encontrado = 0;
   for (i = 0; i < num_eq; i++)
   {
      if ((strcmp(equipos[i].nombre, nombre_buscado) == 0) &&      // Hemos encontrado el equipo
          (equipos[i].borrado != 'S'))                             // y no ha sido borrado
      {
         encontrado = 1;
         equipos[i].borrado = 'S';                          // Lo marcamos como borrado
```

```
         printf("Equipo borrado\n");                        // Informamos al usuario
         break;
      }
   }

   if (encontrado == 0)
      printf("Equipo no encontrado\n");
}


// Pide por teclado un nombre de equipo y lo busca en el array de datos
// Si lo encuentra, muestra sus datos en la pantalla y pide al usuario
// que los modifique.
void modificar_equipo(struct s_equipo equipos[MAX_EQUIPOS], int num_eq)
{
   char nombre_buscado[50], aux[50];
   int i, encontrado;

   // Leer nombre del equipo que se pretende modificar
   printf("Introduzca nombre del equipo que quiere modificar: ");
   gets(nombre_buscado);

   // Recorrer el array para localizar al equipo que se quiere modificar
   encontrado = 0;
   for (i=0; i < num_eq; i++)
   {
      if ((strcmp(equipos[i].nombre, nombre_buscado) == 0) &&   // Hemos encontrado el equipo
          (equipos[i].borrado != 'S'))                          // y no ha sido borrado
      {
        encontrado = 1;
        // Mostramos sus datos actuales
        printf("Equipo encontrado. Sus datos actuales son:\n");
        printf("Nombre: %s\n", equipos[i].nombre);
        printf("Jugados = %i, Puntos = %i\n", equipos[i].jug, equipos[i].puntos);
        printf("Ganados = %i, Empatados = %i, Perdidos = %i\n\n",
                                    equipos[i].gan, equipos[i].emp, equipos[i].per);
        // Pedimos al usuario que introduzca los datos nuevos
        printf("Introduzca los datos nuevos:\n");
        printf("Nombre: ");
        gets(equipos[i].nombre);
        printf("Partidos jugados: ");
        gets(aux); equipos[i].jug = atoi(aux);
        printf("Partidos ganados: ");
        gets(aux); equipos[i].gan = atoi(aux);
        printf("Partidos perdidos: ");
        gets(aux); equipos[i].per = atoi(aux);
        equipos[i].emp = equipos[i].jug - equipos[i].per - equipos[i].gan;
        equipos[i].puntos = equipos[i].gan * 3 + equipos[i].emp * 1;

        break;
      }
   }

   // Mostrar resultado al usuario
   if (encontrado == 1)
      printf("Equipo modificado\n");
   else
      printf("Equipo no encontrado");

}
```

SOLUCIÓN IDEAL

Estas dos son **soluciones extremas** que se pueden aplicar a casi todos los programas que usan archivos. Podemos resumir ambas soluciones de esta manera:

- **Solución 1**: usar directamente los registros en la memoria secundaria, cargando en la mempria principal sólo el registro al que estemos accediendo en cada momento.
- **Solución 2**: cargar todos los registros en la memoria principal, usando ésta durante todo el programa, para volver a grabar todos los registros en memoria secundaria justo antes de terminar la ejecución.

Como hemos visto, las dos soluciones tienen ventajas y desventajas. **Lo ideal**, por lo tanto, sería **una combinación de ambas**. Por ejemplo, se pueden cargar en memoria principal un conjunto de registros (ni todos, ni sólo uno) y operar con ellos. Cuando haga falta otro conjunto de registros, el primero se guarda en el archivo y el nuevo conjunto se lee. Esto, como es lógico, complica mucho la labor de programación.

Además, ten en cuenta que **las soluciones planteadas son mejorables** en muchos aspectos. Por ejemplo, en la segunda solución, los registros borrados siguen ocupando memoria hasta que el programa termina. Se podría hacer un volcado

periódico de datos desde el array hacia el archivo, de manera que se eliminaran los registros borrados y, además, no se perdiera toda la información en caso de que el programa terminase abruptamente.