```
/*  Curso de C,              */
/*    Nacho Cabanes          */
/*---------------------------*/

#include <stdio.h>

main()
{
  char mensajeError[5][80] = {
    "Fichero no encontrado",
    "El fichero no se puede abrir para escritura",
    "El fichero está vacío",
    "El fichero contiene datos de tipo incorrecto"
    "El fichero está siendo usado"
    };

  printf("El segundo mensaje de error es: %s",
    mensajeError[1]);
}
```

Ejercicios propuestos:

- Un programa guarde los nombres de los meses. El usuario deberá indicar un número de mes (por ejemplo, 3) y se le mostrará el nombre de dicho mes (por ejemplo, Marzo).
- Usar un array de 3 dimensiones para guardar los nombres de los meses en español e inglés. El usuario deberá indicar un número de mes (por ejemplo, 3) y se le mostrará el nombre de dicho mes en español (Marzo) y en inglés (March).

5.4. Arrays indeterminados.

Si damos un valor inicial a un array, no será necesario que indiquemos su tamaño, porque el compilador lo puede saber contando cuantos valores hemos detallado, así:

```
int punto[] = {10, 0, -10};
char saludo[ ] = "hola";
char mensajes[][80] = {"Bienvenido", "Hasta otra"};
```

Ejercicios propuestos:

- Un programa que pida 10 nombres y los memorice. Después deberá pedir que se teclee un nombre y dirá si se encuentra o no entre los 10 que se han tecleado antes. Volverá a pedir otro nombre y a decir si se encuentra entre ellos, y así sucesivamente hasta que se teclee "fin".
- Un programa que prepare espacio para un máximo de 100 nombres (de un máximo de 80 letras cada uno). El usuario deberá ir introduciendo un nombre cada vez, hasta que se pulse Intro sin teclear nada, momento en el que dejarán de pedirse más nombres y se mostrará en pantalla la lista de los nombres que se han introducido hasta entonces.

5.5. Estructuras

5.5.1. Definición y acceso a los datos

Un **registro** es una agrupación de datos, los cuales no necesariamente son del mismo tipo. Se definen con la palabra "**struct**".

Para acceder a cada uno de los datos que forman el registro, tanto si queremos leer su valor como si queremos cambiarlo, se debe indicar el nombre de la variable y el del dato (o campo) separados por un punto:

```
/*---------------------------*/
/*  Ejemplo en C nº 50:      */
/*  c050.c                   */
/*                           */
/*  Registros (struct)       */
/*                           */
/*  Curso de C,              */
/*    Nacho Cabanes          */
/*---------------------------*/

#include <stdio.h>

main()
{
  struct
  {
    char  inicial;
    int   edad;
    float nota;
  } persona;

  persona.inicial = 'J';
  persona.edad = 20;
  persona.nota = 7.5;
  printf("La edad es %d", persona.edad);
}
```

Como es habitual en C, para declarar la variable hemos indicado primero el tipo de datos (struct { ...}) y después el nombre que tendrá esa variable (persona).

También podemos declarar primero cómo van a ser nuestros registros, y más adelante definir variables de ese tipo:

```
/*---------------------------*/
/*  Ejemplo en C nº 51:      */
/*  c051.c                   */
/*                           */
/*  Registros (2)            */
/*                           */
/*  Curso de C,              */
/*    Nacho Cabanes          */
/*---------------------------*/

#include <stdio.h>

struct datosPersona
{
  char  inicial;
  int   edad;
  float nota;
};

main()
{
  struct datosPersona ficha;
```

```
  ficha.inicial = 'J';
  ficha.edad = 20;
  ficha.nota = 7.5;
  printf("La edad es %d", ficha.edad);
}
```

Ejercicios propuestos:

- Un "struct" que almacene datos de una canción en formato MP3: Artista, Título, Duración (en segundos), Tamaño del fichero (en KB). Un programa debe pedir los datos de una canción al usuario, almacenarlos en dicho "struct" y después mostrarlos en pantalla.

5.5.2. Arrays de estructuras

Hemos guardado varios datos de una persona. Se pueden almacenar los de **varias personas** si combinamos el uso de los "struct" con las tablas (arrays) que vimos anteriormente. Por ejemplo, si queremos guardar los datos de 100 alumnos podríamos hacer:

```
struct
{
  char  inicial;
  int   edad;
  float nota;
} alumnos[100];
```

La inicial del primer alumno sería "alumnos[0].inicial", y la edad del último sería "alumnos[99].edad".

Ejercicios propuestos:

- Ampliar el programa del apartado 5.5.1, para que almacene datos de hasta 100 canciones. Deberá tener un menú que permita las opciones: añadir una nueva canción, mostrar el título de todas las canciones, buscar la canción que contenga un cierto texto (en el artista o en el título).
- Un programa que permita guardar datos de "imágenes" (ficheros de ordenador que contengan fotografías o cualquier otro tipo de información gráfica). De cada imagen se debe guardar: nombre (texto), ancho en píxeles (por ejemplo 2000), alto en píxeles (por ejemplo, 3000), tamaño en Kb (por ejemplo 145,6). El programa debe ser capaz de almacenar hasta 700 imágenes (deberá avisar cuando su capacidad esté llena). Debe permitir las opciones: añadir una ficha nueva, ver todas las fichas (número y nombre de cada imagen), buscar la ficha que tenga un cierto nombre.

5.5.3. Estructuras anidadas

Podemos encontrarnos con un registo que tenga varios datos, y que a su vez ocurra que uno de esos datos esté formado por varios datos más sencillos. Para hacerlo desde C, incluiríamos un "struct" dentro de otro, así:

```
/*---------------------------*/
/*  Ejemplo en C nº 52:      */
/*  c052.c                   */
/*                           */
```

```
/*  Registros anidados       */
/*                           */
/*  Curso de C,              */
/*    Nacho Cabanes          */
/*--------------------------*/

#include <stdio.h>

struct fechaNacimiento
{
  int  dia;
  int  mes;
  int anyo;
};


struct
{
  char  inicial;
  struct fechaNacimiento diaDeNacimiento;
  float nota;
} persona;

main()
{
  persona.inicial = 'I';
  persona.diaDeNacimiento.mes = 8;
  persona.nota = 7.5;
  printf("La nota es %f", persona.nota);
}
```

Ejercicios propuestos:

- Ampliar el programa del primer apartado de 5.5.2, para que el campo "duración" se almacene como minutos y segundos, usando un "struct" anidado que contenga a su vez estos dos campos.

5.6 Ejemplo completo

Vamos a hacer un ejemplo completo que use tablas ("arrays"), registros ("struct") y que además manipule cadenas.

La idea va a ser la siguiente: Crearemos un programa que pueda almacenar datos de hasta 1000 ficheros (archivos de ordenador). Para cada fichero, debe guardar los siguientes datos: Nombre del fichero (max 40 letras), Tamaño (en KB, número de 0 a 2.000.000.000). El programa mostrará un menú que permita al usuario las siguientes operaciones:

1- Añadir datos de un nuevo fichero
2- Mostrar los nombres de todos los ficheros almacenados
3- Mostrar ficheros que sean de más de un cierto tamaño (por ejemplo, 2000 KB).
4- Ver todos los datos de un cierto fichero (a partir de su nombre)
5- Salir de la aplicación (como todavía no sabemos almacenar los datos, éstos se perderán).

No debería resultar difícil. Vamos a ver directamente una de las formas en que se podría plantear y luego comentaremos alguna de las mejoras que se podría (incluso se debería) hacer.

Una opción que podemos a tomar para resolver este problema es la de contar el número de fichas que tenemos almacenadas, y así podremos añadir de una en una. Si tenemos 0 fichas, deberemos almacenar la siguiente (la primera) en la posición 0; si tenemos dos fichas, serán la 0 y la 1, luego añadiremos en la posición 2; en general, si tenemos "n" fichas, añadiremos cada nueva ficha en la posición "n". Por otra parte, para revisar todas las fichas, recorreremos desde la posición 0 hasta la n-1, haciendo algo como

```
for (i=0; i<=n-1; i++) { ... más órdenes ...}
```

o bien algo como

```
for (i=0; i<n; i++) { ... más órdenes ...}
```

El resto del programa no es difícil: sabemos leer y comparar textos y números. Sólo haremos tres consideraciones:

- Los textos (nombre del fichero, por ejemplo) pueden contener espacios, por lo que usaremos "gets" en vez de "scanf".
- Es **"peligroso" mezclar órdenes "gets" y "scanf"**: si leemos un número con "scanf", la pulsación de la tecla "Intro" posterior se queda en el buffer del teclado, lo que puede provocar que después intentemos leer con "gets" un texto, pero sólo leamos esa pulsación de la tecla "Intro". Para evitarlo, los números los leeremos "en dos etapas": primero leeremos una cadena con "gets" y luego la convertiremos a número con "sscanf".
- Hemos limitado el número de fichas a 1000, así que, si nos piden añadir, deberíamos asegurarnos antes de que todavía tenemos hueco disponible.

Con todo esto, nuestro fuente quedaría así:

```
/*---------------------------*/
/*  Ejemplo en C nº 53:      */
/*  c053.c                   */
/*                           */
/*  Tabla con muchos struct  */
/*  y menu para manejarla    */
/*                           */
/*  Curso de C,              */
/*    Nacho Cabanes          */
/*---------------------------*/

#include <stdio.h>
#include <string.h>

struct{
   char nombreFich[41];     /* Nombre del fichero */
   unsigned long tamanyo;   /* El tamaño en bytes */
} fichas[1000];

int numeroFichas=0;  /* Número de fichas que ya tenemos */
int i;               /* Para bucles */

int opcion;          /* La opcion del menu que elija el usuario */
```

```
char textoTemporal[40]; /* Para cuando preguntemos al usuario */
unsigned long numeroTemporal;

main()
{
  do {
    /* Menu principal */
    printf("Escoja una opción:\n");
    printf("1.- Añadir datos de un nuevo fichero\n");
    printf("2.- Mostrar los nombres de todos los ficheros\n");
    printf("3.- Mostrar ficheros que sean de mas de un cierto tamaño\n");
    printf("4.- Ver datos de un fichero\n");
    printf("5.- Salir\n");

    /* Para evitar problemas con datos mal introducidos,
       leemos con "gets" y luego lo filtramos con "sscanf" */
    gets (textoTemporal);
    sscanf(textoTemporal, "%d", &opcion);

    /* Hacemos una cosa u otra según la opción escogida */
    switch(opcion){
       case 1: /* Añadir un dato nuevo */
               if (numeroFichas < 1000) {  /* Si queda hueco */
                 printf("Introduce el nombre del fichero: ");
                 gets(fichas[numeroFichas].nombreFich);
                 printf("Introduce el tamaño en KB: ");
                 gets(textoTemporal);
                 sscanf(textoTemporal,"%ld",&fichas[numeroFichas].tamanyo);
                 /* Y ya tenemos una ficha más */
                 numeroFichas++;
               } else   /* Si no hay hueco para más fichas, avisamos */
                 printf("Máximo de fichas alcanzado (1000)!\n");
               break;
       case 2: /* Mostrar todos */
               for (i=0; i<numeroFichas; i++)
                  printf("Nombre: %s; Tamaño: %ld Kb\n",
                    fichas[i].nombreFich, fichas[i].tamanyo);
               break;
       case 3: /* Mostrar según el tamaño */
               printf("¿A partir de que tamaño quieres que te muestre?");
               gets(textoTemporal);
               sscanf(textoTemporal, "%ld", &numeroTemporal);
               for (i=0; i<numeroFichas; i++)
                 if (fichas[i].tamanyo >= numeroTemporal)
                    printf("Nombre: %s; Tamaño: %ld Kb\n",
                      fichas[i].nombreFich, fichas[i].tamanyo);
               break;
       case 4: /* Ver todos los datos (pocos) de un fichero */
               printf("¿De qué fichero quieres ver todos los datos?");
               gets(textoTemporal);
               for (i=0; i<numeroFichas; i++)
                 if (strcmp(fichas[i].nombreFich, textoTemporal) == 0)
                    printf("Nombre: %s; Tamaño: %ld Kb\n",
                      fichas[i].nombreFich, fichas[i].tamanyo);
               break;
       case 5: /* Salir: avisamos de que salimos */
               printf("Fin del programa\n");
               break;
       default: /* Otra opcion: no válida */
               printf("Opción desconocida!\n");
               break;
       }
   } while (opcion != 5);  /* Si la opcion es 5, terminamos */
```