

# **SISTEMAS INFORMÁTICOS**

## **GITT, GIT, GIST Y GIEC**

### **PRÁCTICA 4: Estructuras**

UAH, Departamento de Automática, ATC-SOL

#### **OBJETIVOS**

- Conocer el concepto y el uso de estructuras
- Aprender a realizar programas que utilicen arrays de estructuras
- Utilizar y controlar cadenas de caracteres

#### **TEMPORIZACIÓN**

- Inicio de la práctica: Semana del 6 de noviembre
- Tiempo de desarrollo de la práctica: 3 semanas. Finalizarla antes de la prueba de evaluación parcial 3
- Evaluación: ver fechas en el aula virtual

## 1. Introducción

La práctica 4 se compone de dos ejercicios que engloban tanto los conceptos de arrays de estructuras como todos los conceptos de programación vistos a lo largo del curso. En el primer ejercicio debe realizar un diccionario inglés-español y español-inglés. Debe comprender el código mostrado, completar la función `main` y realizar las funciones que faltan. El segundo ejercicio maneja arrays de estructuras para almacenar los datos de los alumnos de una clase y visualizarlos por pantalla junto con las estadísticas de las notas obtenidas. Debe realizar el programa completo.

## 2. Ejercicio1. Diccionario

### 1. Explicación del ejercicio

Para la realización del diccionario inglés-español, español-inglés debe utilizar un array de tantos elementos como palabras contenga el diccionario (suponga que tendrá un máximo de 100 palabras). Cada elemento del array albergará una palabra en español y su correspondiente traducción al inglés. Por ejemplo: perro y dog, gato y cat, etc.

Como puede notar cada posición del array debe alojar dos variables, una para almacenar la palabra en español y otra para almacenar la palabra en inglés. Si revisa los conceptos vistos en teoría, existe un tipo de dato que permite agrupar más de una variable del mismo o diferente tipo. Este tipo de dato es la estructura. Por lo tanto el diccionario se va almacenar en un array de estructuras donde cada estructura tendrá dos miembros, uno almacenará la palabra en español y el otro la palabra en inglés. La siguiente figura aporta una idea gráfica del diccionario. Cada estructura está representada por un rectángulo.

#### Diccionario

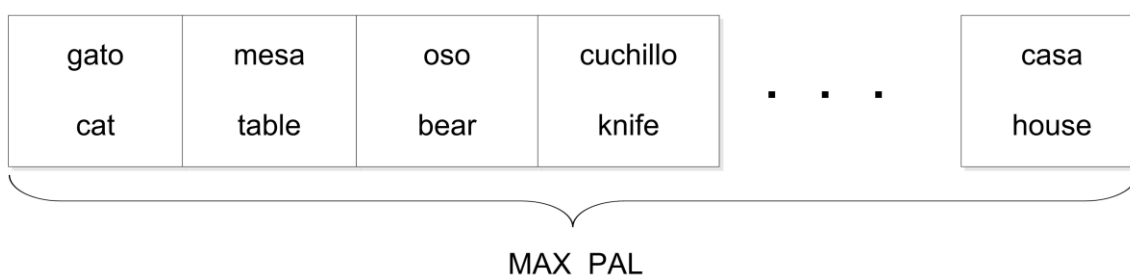


Figura 1. Representación gráfica del diccionario

Analice ahora el tipo de dato que necesita para guardar cada una de las palabras. Se trata de cadenas de caracteres que se almacenan en arrays de tipo `char`. Suponga que para este ejercicio las palabras que se van a introducir desde teclado en el momento de la creación del diccionario, tienen una longitud máxima de 19 caracteres. Esto significa que necesita un array de 20 caracteres, 19 son para almacenar la palabra y uno más para almacenar el carácter fin de cadena: `'\0'`. La variable que albergará cada una de las cadenas de caracteres se define de la siguiente forma:

```
char identificador_de_la_cadena[MAX_CAD];
```

Donde `MAX_CAD` se ha definido como una constante de valor 20.

El tipo de dato que ha sido definido para la creación del diccionario es el siguiente:

```
typedef struct
{
    char ingles[MAX_CAD];
    char espanyol[MAX_CAD];
} tPalabra;
```

Observe que los identificadores elegidos para nombrar las variables, expresan de forma certera el contenido de estas. Por ejemplo, la cadena que contendrá la palabra en español recibe el nombre `espanyol`, la que almacenará la palabra en inglés se llama `ingles`, el array de estructuras que albergará todo el diccionario se llama `Diccionario`. Esta práctica es fundamental para conseguir una buena técnica de programación, pues facilita el entendimiento del código cuando sea leído por otras personas y por usted mismo/a una vez que haya pasado algún tiempo.

No confunda el array de estructuras donde se almacenarán todas las palabras del diccionario junto con su traducción, con los arrays de caracteres donde se almacenarán las palabras en español o en inglés. Observe la siguiente imagen donde se diferecian gráficamente ambos arrays.

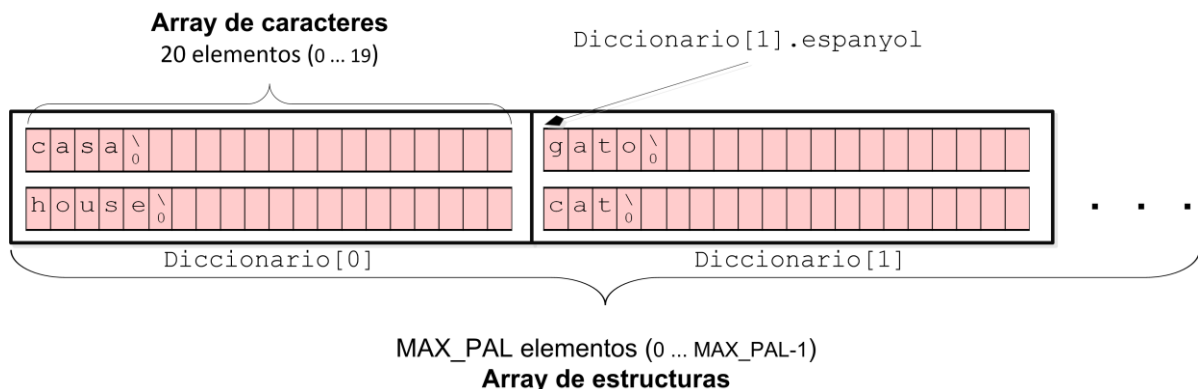


Figura 2. Representación del array de estructuras y los arrays de caracteres

## 2. Acciones a realizar

Junto con el enunciado de la práctica se proporcionan dos ficheros: un fichero de código (`main_Dicc.c`) formado por la función principal `main()`, la función `menu()` y la función `LeeCadena()`, y un fichero cabecera (`diccionario.h`) que contiene las directivas al preprocesador necesarias. **Debe crear un proyecto y añadir estos ficheros al proyecto. Además, debe añadir otro fichero de código donde debe incluir las funciones que va a programar.** A modo de repaso recuerde el tema 3, donde se habló de programas formados por múltiples ficheros que incluyen uno o varios ficheros de encabezado y de

código. Solo un fichero de código debe contener la función `main()`. El resto de ficheros de código contendrán el código fuente de las funciones.

Note que el tipo de dato mencionado ya está definido en el fichero cabecera junto con los prototipos de las funciones y las directivas `include` correspondientes. Observe que la función `menu()`, ya codificada, imprime un menú que ofrece las siguientes acciones:

1. Añadir una nueva palabra al diccionario.
2. Traducir una de las palabras que haya sido introducida previamente.
3. Mostrar el diccionario entero (todas sus palabras junto con su traducción).
4. Salir del programa.

La función `LeeCadena()` lee una cadena de caracteres desde teclado, recibe el array de tipo `char` donde se va a almacenar la cadena y el número máximo de caracteres que puede contener, incluido el carácter fin de cadena (`'\0'`). Si analiza el código verá que utiliza la función `fgets()` y transforma el último carácter de la cadena (`'\n'`) en `'\0'`. Esto debe ser así, puesto que para el caso de que la cadena leída tenga una longitud menor al número máximo de caracteres, `fgets()` no realiza este cambio de forma automática. Haga una llamada a esta función cada vez que necesite leer una cadena desde teclado.

**Su tarea consiste en codificar la función `main` y las funciones: `anyadir_palabra()`, `traducir_palabra()` y `mostrar_diccionario()`.** Observe que para crear el diccionario, el usuario debe pulsar la opción 1 cada vez que desee añadir una nueva palabra.

**IMPORTANTE:** realice primero la opción 1 del menú, compile, ejecute y corrija los posibles errores de ejecución antes de continuar con la opción 2. A continuación repita el proceso para la opción 2 y finalmente para la opción 3. Si codifica las tres opciones a la vez y compila por primera vez después de haber codificado todo el programa, la dificultad para corregir los errores es mucho mayor y es muy probable que no pueda llegar a una solución final, teniendo que comenzar el programa de nuevo desde el principio.

### 3. Descripción de las funciones a desarrollar

```
int anyadir_palabra (tPalabra Dicc[], int num);
```

La función recibe el diccionario y el número de palabras que tiene. Debe añadir una palabra y su traducción y devuelve el nuevo número actual de palabras, que será el que ya tenía incrementado en uno. Tenga cuidado de no sobrepasar el límite máximo del diccionario.

Para entender mejor el modo de operar de esta función analice la tarea que debe realizar esta, junto con la llamada a la función, realizada desde `main`, y mostrada a continuación:

```
num_pal = anyadir_palabra(Diccionario, num_pal);
```

Como puede ver, se le pasan como argumentos el diccionario y el número de palabras que contiene. Observe que la variable `num_pal` de la función `main` es la encargada de almacenar el número de palabras del diccionario. Por esta razón inicialmente debe valer 0. El número actual de palabras que ha sido retornado por la función `anyadir_palabra` es recogido de nuevo en la variable `num_pal`. Se destruye, por tanto, el valor anterior y almacena el nuevo número de palabras una vez que una nueva palabra ha sido añadida. Esto permite que la variable `num_pal` contabilice todas las palabras que se van añadiendo al diccionario.

Sitúe la llamada a la función en el lugar de `main` que considere adecuado.

Por otro lado, la función `anyadir_palabra` debe realizar los siguientes tareas:

1. Comprobar que el diccionario no está lleno y tiene suficiente espacio para añadir una palabra más (preguntar si el número de palabras es mayor o igual que 100). Si el diccionario está lleno debe imprimir un mensaje indicándolo y retornar el mismo número de palabras que recibió puesto que no se ha añadido ninguna palabra.
2. En caso de que el diccionario no esté lleno:
  - 2.1 Leer desde teclado la palabra en español y almacenarla en la cadena de caracteres `espanyol`, en la posición siguiente a la ocupada por la última palabra del diccionario.
  - 2.2 Leer la palabra en inglés y almacenarla en la cadena `ingles`, en la misma posición donde se añadió la palabra en español. El orden de lectura de las palabras es indiferente. Simplemente debe asegurarse de almacenar la palabra española en la cadena `espanyol` y la palabra inglesa en la cadena `ingles`.
  - 2.3 Retornar el número de palabras más 1, puesto que, en este caso, una nueva palabra ha sido introducida en el diccionario.

Pero, ¿cómo sabe la función `anyadir_palabra()` la posición dentro del array en la que tiene que ser almacenada la nueva palabra? Es sencillo: esta posición está marcada por el número de palabras que tiene el diccionario y que la función lo recibe en uno de sus parámetros.

Para comprobar que todos los conceptos han sido entendidos y afianzar estos, así como para comprobar que todo el código funciona correctamente (una vez que lo haya escrito), debe pensar como se ejecutaría el programa para todos los posibles casos de uso. En este ejercicio podemos considerar 3 casos de uso:

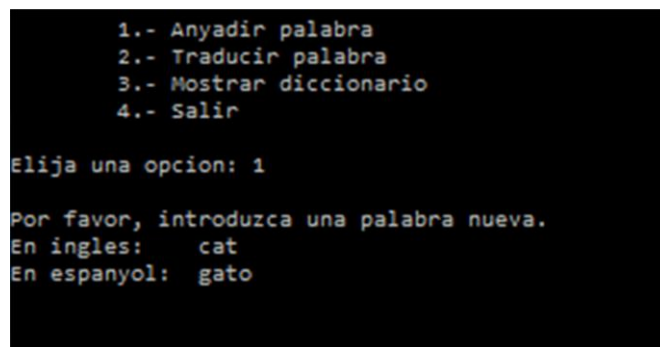
1. Introducción del primer elemento en el diccionario
2. Introducción de un elemento intermedio

### 3. Introducción de un elemento en caso de que el diccionario esté lleno.

Suponga por ejemplo que el diccionario está vacío y vamos a añadir la primera palabra. En ese caso la variable `num_pal` de la función `main` vale 0. La función `anyadir_palabra` recibe el diccionario vacío y el número de palabras (0). A continuación chequeará si el diccionario está lleno. Como no es el caso, pasará a leer las palabras desde teclado y a almacenarlas en el array. La posición donde se almacenan es aquella que ha sido pasada en el argumento `num_pal` (0) que coincide con la primera posición del array. La función, para terminar, retorna el número de palabras más 1, puesto que se ha añadido una palabra. Finalmente este valor es recogido por `main()` en la variable `num_pal`, que ahora ya tiene el valor 1 y que corresponde con el número de palabras que tiene el diccionario.

Como un segundo caso, imagine que el diccionario tiene 3 palabras (`num_pal` vale 3). La función recibe el diccionario con las tres palabras y el número de palabras. Como el diccionario no está lleno leerá desde teclado las dos palabras y las almacenará en la posición 3 del array de estructuras que corresponde al lugar de la cuarta palabra en el array. Finalmente retornará el número 4 como número actual de palabras que tien el diccionario y que queda almacenado en la variable `num_pal` de la función `main`.

Como último caso, imagine que el diccionario está lleno y vamos a introducir una palabra. La función recibe el diccionario y el valor 100 como número de palabras, detecta que el diccionario está lleno y envía un mensaje de aviso a la pantalla retornando el mismo número de palabras que recibió (100), sin añadir ninguna palabra nueva. La figura 3 muestra la ejecución del programa cuando se añade una nueva palabra al diccionario.



```
1.- Anyadir palabra
2.- Traducir palabra
3.- Mostrar diccionario
4.- Salir

Elija una opcion: 1

Por favor, introduzca una palabra nueva.
En ingles:      cat
En espanyol:    gato
```

Figura 3. Introducción de una palabra en el diccionario

```
void traducir_palabra (tPalabra Dicc[], int num);
```

Esta función recibe en sus parámetros el diccionario y el número de palabras que lo componen. En primer lugar, la función `traducir_palabra` debe leer desde teclado la palabra que se desea traducir (en inglés o en español). A continuación debe buscarla en el diccionario y si la encuentra mostrará su traducción por pantalla. En concreto debe realizar las siguientes tareas:

1. Leer desde teclado la palabra a traducir.
2. Recorrer el diccionario comparando los dos campos de cada estructura con la palabra a traducir (utilice la función `strcmp`). Si la palabra buscada es encontrada en el campo `espanyol` de una posición genérica (`i`) del diccionario, su traducción será el campo `ingles` de la misma posición (`i`), y viceversa. La figura 4 representa gráficamente la búsqueda de la palabra `perro` en el diccionario.

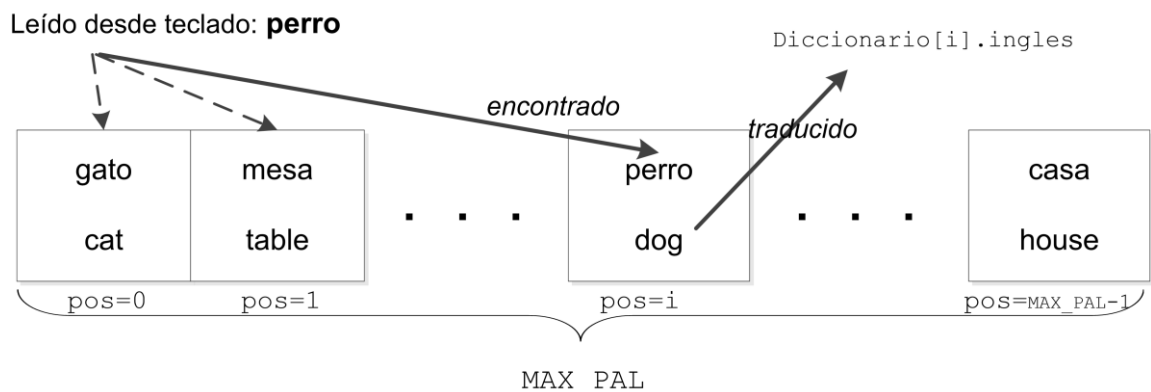


Figura 4. Búsqueda en el diccionario y traducción de una palabra leída desde teclado

3. Una vez encontrada la palabra, mostrar por pantalla su traducción.
4. Si la palabra a traducir no se encuentra en el diccionario, imprimir un mensaje informando que la palabra buscada no existe.

La figura 5 muestra la ejecución del programa cuando se busca una palabra.

```

1.- Anyadir palabra
2.- Traducir palabra
3.- Mostrar diccionario
4.- Salir

Elija una opcion: 2

Por favor, introduzca una palabra.
Palabra a traducir:  perro
Traducción: dog
Presione una tecla para continuar . . .

```

Figura 5. Búsqueda de una palabra en el diccionario

```
void mostrar_diccionario (tPalabra Dicc[], int num);
```

Esta función recibe el diccionario y el número de palabras en sus parámetros. No devuelve ningún valor. La función debe recorrer el diccionario e imprimir todas las palabras que lo componen junto con su traducción. La figura 6 muestra la ejecución del programa cuando se ha pulsado la opción 3.

```
1.- Anyadir palabra
2.- Traducir palabra
3.- Mostrar diccionario
4.- Salir

Elija una opcion: 3

DICCIONARIO

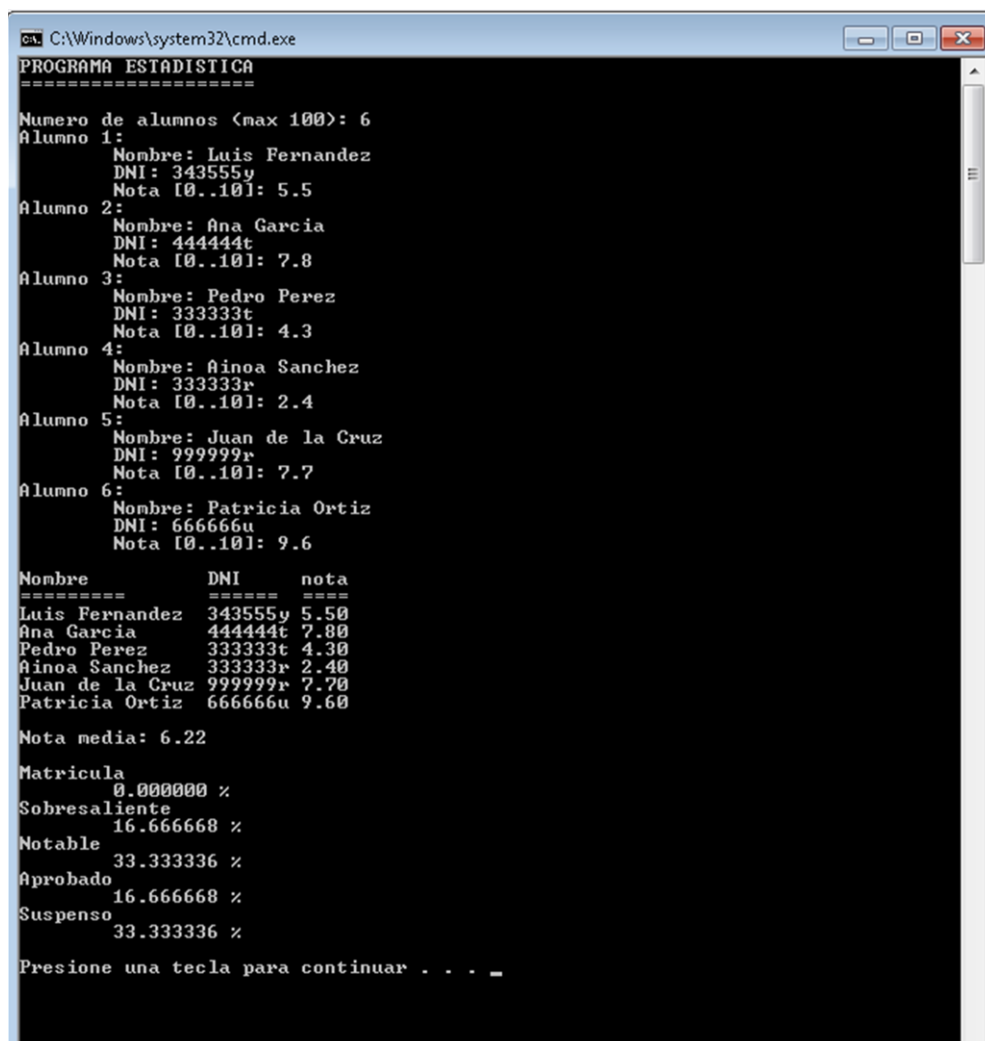
    ESPANYOL      INGLES
    -----
    gato          cat
    mesa          table
    perro         dog
    loro          parrot
Presione una tecla para continuar . . .
```

Figura 6. Listado de todas las palabras del diccionario



### 3. Ejercicio 2. Estadísticas de notas de una clase

Realice un programa que lea los nombres, DNI's y las notas de todos los alumnos de una clase (considere un máximo de 100 alumnos) y calcule la nota media y el porcentaje de matrículas, sobresalientes, notables, aprobados y suspensos. Una vez leídos los datos desde el teclado, debe mostrarlos por pantalla junto con los cálculos estadísticos obtenidos. La figura 7 muestra la ejecución del programa. En este ejercicio no dispone de ningún fichero de código, debe realizar el **programa completo**.



```
ca. C:\Windows\system32\cmd.exe
PROGRAMA ESTADISTICA
=====
Numero de alumnos (max 100): 6
Alumno 1:
    Nombre: Luis Fernandez
    DNI: 343555y
    Nota [0..10]: 5.5
Alumno 2:
    Nombre: Ana Garcia
    DNI: 444444t
    Nota [0..10]: 7.8
Alumno 3:
    Nombre: Pedro Perez
    DNI: 333333t
    Nota [0..10]: 4.3
Alumno 4:
    Nombre: Ainoa Sanchez
    DNI: 333333r
    Nota [0..10]: 2.4
Alumno 5:
    Nombre: Juan de la Cruz
    DNI: 999999r
    Nota [0..10]: 7.7
Alumno 6:
    Nombre: Patricia Ortiz
    DNI: 666666u
    Nota [0..10]: 9.6

Nombre      DNI      nota
=====
Luis Fernandez 343555y 5.50
Ana Garcia    444444t 7.80
Pedro Perez   333333t 4.30
Ainoa Sanchez 333333r 2.40
Juan de la Cruz 999999r 7.70
Patricia Ortiz 666666u 9.60

Nota media: 6.22

Matricula
0.000000 %
Sobresaliente
16.666668 %
Notable
33.333336 %
Aprobado
16.666668 %
Suspenso
33.333336 %

Presione una tecla para continuar . . . _
```

Figura 1. Introducción de los datos de 6 alumnos y visualización de los resultados

Para encarar un programa de esta magnitud se aconseja actuar de forma ordenada dividiendo el problema en distintas partes. Se propone plantearse y resolver las siguientes cuestiones:

1. ¿Cómo organizo la información? ¿Qué tipo de datos y variables necesito para almacenar los datos de los alumnos?
2. ¿Cómo organizo el código en diferentes funciones?
3. ¿En qué ficheros escribo los distintos elementos de código que voy a desarrollar?

### 3.1. Organización de la información, tipos de datos y variables

Los datos que debe manejar por cada alumno son nombre, dni y nota. ¿Qué tipo de dato conoce usted que permita albergar distintos campos del mismo o diferente tipo? La respuesta, como se vio en el primer ejercicio, es la estructura. Piense una estructura que incluya los 3 miembros mencionados. Una vez decidido el tipo de estructura, es fácil suponer que la información de todos los estudiantes es sólo posible almacenarla en un array de estructuras, cuyo número de elementos vendrá dado por el número máximo de estudiantes. La figura 8 refleja gráficamente el array.

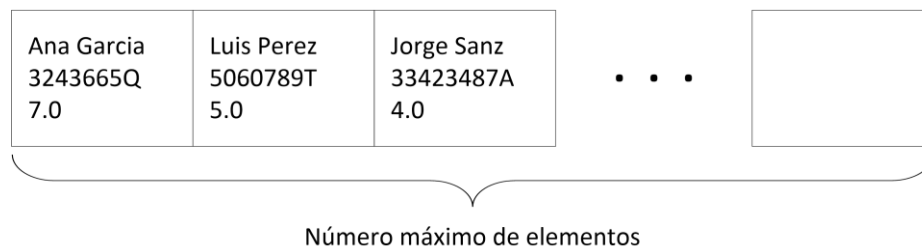


Figura 8. Representación gráfica del array de estructuras

### 3.2. Organización del código en distintas funciones

Una vez que ha pensado donde va a almacenar la información de la que dispone, piense cómo organizar el código en distintas funciones que ejecuten las tareas solicitadas de la forma más eficiente posible. Para ello puede plantearse los siguientes interrogantes:

#### ¿Cuántas funciones necesito?

El número de líneas de código de las funciones no debe ser excesivamente largo ni excesivamente corto. Un número de líneas de código entre 3 y 9 puede considerarse adecuado para una función. Piense también que la función debe realizar una tarea lógica dentro del programa.

Procure que no haya código fuente repetido. Si esto ocurriera significaría que podría crearse una nueva función y el código repetido sustituirse por llamadas a ésta.

Para este ejercicio el código puede ser organizado en tres bloques: lectura de los datos desde teclado, visualización de la información por pantalla y realización y gestión de los

cálculos estadísticos: media y porcentaje de suspensos, aprobados, notables, sobresalientes y matrículas.

### ¿Cuáles son los parámetros que deben tener mis funciones para realizar su tarea?

Piense cómo va a realizar la tarea dentro de la función. En dicho proceso se dará cuenta de que necesita variables de la función `main` para llevarla a cabo. Esas variables son las que deben ser pasadas desde `main`<sup>1</sup> (en la llamada a la función) como argumentos. Tenga cuidado de no pasar argumentos a las funciones, que no vayan a ser utilizados por éstas. Tal práctica no tiene ningún sentido.

### ¿Deben retornar algún valor que sea necesario para la función `main`?

Piense si `main` necesita alguno de los cálculos realizados por la función. En tal caso debe ser retornado el dato en cuestión.

En definitiva, debe emplear un tiempo antes de escribir el código para asegurarse de que no existe otra forma más eficiente (más rápida, con menos líneas de código, etc.) de proceder.

Ateniéndose a la información anterior cabe la posibilidad de dividir los tres bloques mencionados en varias funciones cada uno.

## 3.2.1 Bloque de lectura de datos desde teclado

Considere la realización de tres funciones para este bloque: `PedirDatos()`, `LeeAlumno()` y `LeeCadena()`.

La función `PedirDatos()` soportará el grueso de la tarea y se apoyará en las funciones `LeeAlumno()` y `LeeCadena()` para llevar a cabo con éxito la lectura de todos los datos. Es decir, las funciones `LeeAlumno()` y `LeeCadena()` serán llamadas por la función `PedirDatos()`. `PedirDatos()` preguntará al usuario el número de alumnos (valor de 0 a 100), y, para cada alumno, llamará a la función `LeeAlumno()`. Además, debe retornar a la función `main()` el número de alumnos leído. Su prototipo es el siguiente:

```
int PedirNotas (tficha Fichas[]);
```

Parámetros:

<code>Fichas</code>	Array de fichas con nombre, dni y nota
---------------------	--

Valor retornado:

<code>int</code>	Número de fichas leídas.
------------------	--------------------------

---

<sup>1</sup> Nota: Para una mayor simplicidad de la explicación, se está considerando que las funciones son llamadas desde `main`. No obstante, debe tener en cuenta, tal y como ocurre en este mismo ejercicio, que una función también puede ser llamada desde otras funciones.

La misión de la función `LeeAlumno()` es leer los datos de un alumno y retornar una estructura con los datos leídos. No recibe ningún argumento.

La función `LeeCadena()` ya es conocida para usted, puesto que trabajó con ella en el ejercicio anterior. Debe ser llamada cada vez que necesite realizar la lectura de un cadena de caracteres.

El hecho de independizar la lectura de los datos de un solo alumno en la función `LeeAlumno()` es útil por dos motivos:

1. Facilita la reducción del número de líneas de código de la función `PedirDatos()`.
2. Hace el código más modular, posibilitando la lectura de un solo alumno con independencia del resto de alumnos. Si todos los alumnos se leyeran desde la función `PedirDatos()`, no se podría leer un solo alumno, en caso de ser necesario en otro punto del programa.

### 3.2.2 Bloque de visualización de la información

Este bloque debe visualizar los datos de los alumnos que han sido introducidos desde teclado. Realice este bloque en dos funciones: `MuestraAlumnos()` y `MuestraAlumno()`. Piense los prototipos de ambas funciones.

### 3.2.3 Bloque de realización de los cálculos estadísticos

El cometido de este bloque es realizar los cálculos estadísticos e imprimirlos por pantalla. Está formado por dos funciones: `CalculaEstadistica()` e `ImprimeEstadistica()`.

La función `CalculaEstadistica` calculará la nota media y el porcentaje de notas de cada intervalo. Para calcular los porcentajes se debe tener en cuenta la siguiente tabla:

Intervalo	Nota
$\geq 0$ y $<5$	<i>suspenso</i>
$\geq 5$ y $<7$	<i>aprobado</i>
7 y $<9$	<i>notable</i>
$\geq 9$ y $<10$	<i>sobresaliente</i>
$=10$	<i>matricula</i>

Para el cómputo del número de notas de cada intervalo debe utilizar un array de enteros de 5 posiciones inicializado a 0. La posición 0 del array contabilizará el número de matrículas, la posición 1 el número de sobresalientes y así sucesivamente. Tenga en cuenta la siguiente declaración:

```
int contadores[NUM_NOTAS]={0};
```

donde `NUM_NOTAS` se ha definido como una constante de valor 5.

La función `CalculaEstadistica()` llamará a la función `ImprimeEstadística()` pasándole los datos necesarios para la visualización por pantalla de los cálculos estadísticos.

La función `ImprimeEstadistica()` imprimirá la nota media de los estudiantes junto con los porcentajes de las notas obtenidas. Vea la figura 1 donde se muestra el formato de salida de la información. Considere la posibilidad de utilizar la siguiente variable global para la impresión de los porcentajes:

```
char Calificaciones[NUM_NOTAS][20]=
{
    "Matricula", "Sobresaliente", "Notable", "Aprobado", "Suspenso"
};
```

Analice el prototipo de la función `ImprimeEstadistica()`:

```
void ImprimeEstadistica(double suma, int cont[], int num)
```

Parámetros:

<code>suma</code>	Suma de las notas de todos los estudiantes
<code>cont</code>	Array de enteros con la cantidad de matriculas, sobresalientes, notables, aprobados y suspensos obtenidos por los estudiantes
<code>num</code>	Número de estudiantes

Valor retornado:

<code>void</code>	Ninguno
-------------------	---------

La función principal presenta el siguiente pseudocódigo:

```
Incluir fichero cabecera
main

    Declarar variables
    PedirNotas
    SI (Numero de alumnos >0) Entonces
        Mostrar datos de alumnos
        Calcular y mostrar estadística
    FIN SI
FIN main
```

### 3.3. Organización del código en varios ficheros

El código del programa se debe estructurar en varios ficheros de código de forma análoga al ejercicio anterior:

<code>main.c:</code>	Función <i>main()</i>
<code>estadistica.c:</code>	Función <i>CalculaEstadistica()</i>
<code>entrada.c:</code>	Funciones <i>PedirNotas()</i> , <i>LeeAlumno()</i> y <i>LeeCadena()</i>
<code>salida.c:</code>	Funciones <i>MuestraAlumnos()</i> , <i>MuestraAlumno()</i> e <i>ImprimeEstadistica()</i>
<code>estadistica.h:</code>	Fichero cabecera con las directrices al preprocesador, constantes, tipos de datos y prototipos de funciones

Finalmente, recordarle que para este ejercicio no se ha publicado ningún fichero de código. Se debe realizar todo el programa desde el principio, teniendo en cuenta que el pseudocódigo de la función `main()` está expuesto en el apartado anterior.