TEMA 7

CADENAS DE CARACTERES - STRINGS

1 INTRODUCCIÓN

Las cadenas de caracteres o *strings* son estructuras de datos iguales a las tablas numéricas o *arrays*, con la diferencia de que almacenan caracteres en lugar de números.
Pese a que todo lo visto hasta ahora para los *arrays* sirve también para los *strings*, las cadenas de caracteres presentan peculiaridades que no son aplicables a las tablas numéricas, y que se tratarán en el siguiente epígrafe. En general, los *strings* pueden tratarse de dos maneras:

- De modo análogo al de las tablas numéricas.
- Recurriendo a las características propias de los *strings*.

Habrá ocasiones en las que no quede más remedio que optar por la primera alternativa, pero en la mayoría de los casos se hará uso de las particularidades de las cadenas de caracteres para resolver de modo más sencillo el problema que se plantee.
La declaración de un *string* es similar a la de un *array*, pues es suficiente con definir un vector con elementos de tipo carácter (*char*), incluyendo un nombre y un número de elementos.

```
char nombre_del_string [número_de_elementos];
```

Ejemplo

Supongamos que se quiere usar un *string*, con un máximo de 30 elementos, de nombre cadena1. Habría que declararlo así:

```
char cadena[30];
```

Como consecuencia, en memoria se habrá reservado espacio para 30 valores de tipo *char*, representados de la siguiente manera:

cadena1

¿?	¿?	¿?	¿?	¿?	¿?	¿?	¿?	¿?	¿?		¿?	¿?	¿?	¿?
0	1	2	3	4	5	6	7	8	9	…		27	28	29

Tal y como ocurre con las tablas numéricas, al declarar una cadena de caracteres los valores que se asignan a cada una de las posiciones son desconocidos (valores *basura*). Una vez declarado un *string*, cada una de sus posiciones se puede utilizar como si se tratara de una variable de tipo *char*.

2 CARACTERÍSTICAS ESPECÍFICAS DE LOS STRINGS

Como se ha adelantado en la introducción, las cadenas de caracteres ofrecen una serie de peculiaridades que permiten utilizarlos de modo diferente a las tablas numéricas. Evidentemente, los *arrays* no proporcionan estas características, por lo que resulta absurdo intentar aplicarlas a un vector que no sea de caracteres.

2.1 El carácter de final de cadena ('\0')

Hasta este momento se ha visto algunos de los valores que puede tomar un elemento de tipo carácter; además de las letras ('a'...'z' y 'A'...'Z) y los dígitos ('0'...'9'), existe una colección de caracteres no alfanuméricos, entre los que cabe mencionar los de puntuación ('.', ',', etcétera), los de separación (' ', '\n' y '\t') y los especiales del teclado ('*', '|', '@', y '€', por citar alguno).

Existe un valor, conocido como carácter fin de cadena y representado como '\0', muy importante a la hora de utilizar *strings*. El carácter de final de cadena simboliza la última posición utilizada de la cadena de caracteres.

cadena1

'H'	'o'	'l'	'a'	'\0'	¿?	¿?	¿?	¿?	¿?		¿?	¿?	¿?	¿?
0	1	2	3	4	5	6	7	8	9	…		27	28	29

De esta manera, si siempre se ubica un '\0' después del último carácter válido del *string*, no será necesario conocer cuántos elementos tiene la cadena (tal y como ocurría con los *arrays*); bastará con recorrer las posiciones de la cadena hasta encontrar la que tiene un carácter fin de cadena, momento en el que se habrá alcanzado el último carácter válido.

Cuando los caracteres de un *string* se almacenan de uno en uno, la responsabilidad de añadir el carácter '\0' en la última posición ocupada es del programador.

Ejemplo

```
char cadena1[30];

cadena1[0] = 'H';
cadena1[1] = 'o';
cadena1[2] = 'l';
cadena1[3] = 'a';
cadena1[4] = ' ';   /*espacio en blanco*/
cadena1[5] = 'm';
cadena1[6] = 'u';
cadena1[7] = 'n';
cadena1[8] = 'd';
cadena1[9] = 'o';
cadena1[10] = '\0';   /* carácter de fin de cadena */
```

Tras ejecutar estas instrucciones, el *string cadena1* quedaría así:

cadena1

'H'	'o'	'l'	'a'	' '	'm'	'u'	'n'	'd'	'o'	'\0'	¿?		¿?	¿?
0	1	2	3	4	5	6	7	8	9	10	11	…	28	29

Existen casos en los que el programador no tiene que añadir explícitamente el carácter '\0' a una cadena de caracteres, pues, como se verá en los siguientes epígrafes, en estas ocasiones el final de cadena se incorpora automáticamente en la posición que le corresponde.

2.2 Inicialización de una cadena de caracteres

Al gual que ocurre con los *arrays*, las primeras posiciones de una cadena de caracteres pueden tomar valores iniciales desde el momento de la declaración del *string*. Por ejemplo:

```
char cad1[20] = "Urano";
```

cad1

'U'	'r'	'a'	'n'	'o'	'\0'	¿?		¿?	¿?
0	1	2	3	4	5	6	…	18	19

Recuérdese que este tipo de asignación sólo puede realizarse en el momento de la declaración de la cadena de caracteres, pues efectuarla en cualquier otra parte del programa dará como resultado un error de compilación.
Puede observarse que, como resultado de la instrucción anterior, se ha reservado espacio en memoria para el *string cad1* y sus primeras posiciones han sido inicializadas, las cinco primeras con los caracteres que componen la palabra "Urano", y la sexta con el carácter fin de cadena, que se ha añadido automáticamente.

2.3 Lectura de *strings* desde teclado: las funciones *scanf* y *gets*

Como se ha visto en el capítulo precedente, para rellenar una tabla numérica con los valores que teclea el usuario, es necesario capturar los datos de uno en uno, con una instrucción *scanf* incluida dentro de un bucle. Salvo que la cantidad de elementos que se pretende leer sea conocida de antemano, lo más habitual suele ser mantener un contador que almacene el número de valores que se ha introducido en el *array* y evitar gracias a él pedir más datos que el máximo definido. Este contador sirve además para controlar los posteriores accesos al *array*.
En el caso de los *strings*, puede plantearse una lectura similar a la de los *arrays*, esto es, carácter a carácter. Por ejemplo, de la siguiente manera:

```
int i=0;
char car, cadena[30];
do
{
   printf("Introduce un carácter (punto ('.') para terminar");
   fflush(stdin);          /*limpia el buffer de teclado*/
   scanf("%d", &car);
   if (car != '.')
   {
     cadena[i] = car;
     i= i+1;
   }
}while (i<30 && car != '.');
cadena[i]= '\0';      /*añade explícitamente el carácter fin de
cadena*/
```

Aunque la estructura sobre la que se almacenan los *strings* puede funcionar del mismo modo que los *arrays*, lo más habitual es hacer uso de las características propias de las cadenas de caracteres. De hecho, los *strings* pueden tratarse como una unidad, y existen funciones específicas de entrada de datos que permiten leerlos de una sola vez, evitando además los principales problemas que plantea la solución anterior, a saber: se puede leer toda la cadena de una sola vez y no carácter a carácter, se evita tener que limpiar el *buffer* de teclado antes de leer cada nuevo carácter, y no es necesario añadir explícitamente el carácter de final de cadena.

La función *scanf* que ya ha sido presentada y utilizada con anterioridad puede emplearse para leer de una sola vez los caracteres que se almacenarán en una cadena. Para ello se recurre al código "%s", que denota que lo que se desea capturar es un *string*.

```
char cadena[20];
...
scanf("%s", cadena);
```

Obsérvese que éste es el único caso en el que el parámetro de la función *scanf* sobre el que se va a efectuar la lectura (en el ejemplo, *cadena*) no va precedido por el símbolo *ampersand* (&).
La instrucción anterior leerá y añadirá a las posiciones del *string* cadena todos los caracteres que el usuario teclee, hasta un máximo de 19 (en cuyo caso los sobrantes permanecerán en el *buffer* de teclado esperando a ser leídos) o hasta que se capture un espacio en blanco (' '), una tabulación ('\t') o un salto de línea ('\n', *intro* o *return*).
La función *gets* realiza exactamente la misma operación, con la diferencia de que no se detiene al leer un espacio en blanco o una tabulación, sino que lo añade a la cadena como si de otro carácter más se tratara. Precisamente por este motivo, es más habitual realizar las lecturas de cadenas de caracteres con la instrucción *gets* en lugar de utilizar *scanf*.

Al leer una cadena de caracteres, ambas funciones, *scanf* y *gets*, añaden automáticamente el carácter de final de cadena en la posición que corresponda.

Ejemplo:

A continuación se muestra un ejemplo de código en C que solicita al usuario una secuencia de caracteres y la almacena en un string.

```
char cadcar[30];

printf("Introduce una serie de como mucho 29 caracteres:\n");
fflush(stdin);
gets(cadcar);
```

Ejemplo de ejecución:

La siguiente figura muestra el mensaje que se escribiría en pantalla al ejecutar este pequeño trozo de código. Al pedírsele una cadena de caracteres, el usuario deberá teclear lo que desee, finalizando con la tecla *return* o *intro*, que, como ya se ha adelantado, es la única con la que la instrucción *gets* deja de capturar caracteres. Supongamos que el texto que teclea el usuario, y que aparece identificado en tipografía cursiva y subrayada, es la palabra "Mercurio".

Pantalla

```
Introduce una serie de como mucho 29 caracteres:
Mercurio
```

La cadena *cadcar* quedaría de la siguiente manera

cadcar

'M'	'e'	'r'	'c'	'u'	'r'	'i'	'o'	'\0'	'¿?'		¿?	¿?
0	1	2	3	4	5	6	7	8	9	…	28	29

Como se aprecia en el ejemplo, el programador no debe preocuparse por el carácter '\0', dado que al ejecutarse la instrucción *gets* el ordenador lo introduce automáticamente en la posición correspondiente.

2.4 Escritura de *strings* en pantalla: la función *printf*

Para escribir en pantalla los elementos de una tabla numérica se hace necesario recorrer todas las posiciones útiles del *array*, con el fin de ir mostrando sus elementos uno a uno, tal y como se ha visto en ejemplos anteriores. En definitiva, hace falta implementar un bucle que en cada iteración imprima por pantalla un único valor.

Aunque en el caso de las cadenas de caracteres también puede escribirse los elementos del *string* de uno en uno de esta manera, gracias a la existencia del carácter de final de cadena, es posible efectuar la escritura de todos los elementos útiles de la cadena con una única instrucción *printf*, esto es, sin necesidad de implementar un bucle.
A raíz de lo comentado en el epígrafe anterior acerca de la instrucción *scanf*, no es de extrañar que la instrucción siguiente muestre por pantalla, automáticamente y de una sola vez, los valores de la cadena *cadcar* que hay antes del carácter '\0', que no se escribe.

```
printf("%s", cadcar);
```

Ejemplo de ejecución:

A continuación se muestra el código que solicita el nombre al usuario para posteriormente saludarle de forma personalizada. Como antes, el texto que teclea el usuario figura en letra cursiva y subrayada para distinguirlo de los mensajes que muestra el programa en pantalla.

```
char nombre[30];

printf("Introduce tu nombre, por favor: ");
fflush(stdin);
gets(nombre);
printf("Hola, %s.", nombre);
```

Pantalla

```
Introduce tu nombre, por favor: Ana Rosa
Hola, Ana Rosa.
```

La cadena *nombre* quedaría de la siguiente manera

nombre

'A'	'n'	'a'	' '	'R'	'o'	's'	'a'	'\0'	'¿?'		¿?	¿?
0	1	2	3	4	5	6	7	8	9	…	28	29

2.5 Número de elementos útiles de un *string*: la función *strlen*

Dado que la última posición válida de una cadena de caracteres la ocupa el carácter '\0', el programador deberá declarar los *strings* con una posición más que el máximo que se espera para ellos. Así, si se desea almacenar un NIF que ocupa 9 caracteres como máximo, el *string* que lo almacenará deberá dar cabida a 10 caracteres (hasta 8 dígitos, una letra y el carácter '\0').

Ejemplo:

```
char nif[10];

printf("Introduce tu NIF, por favor: ");
fflush(stdin);
gets(nif);
```

Pantalla

```
Introduce tu NIF, por favor: 14278210P
```

La cadena *nif* quedaría, en este caso, completa hasta su última posición física:

nif

'1	'4'	'2'	'7'	'8'	'2'	'1'	'0'	'P'	'\0'
0	1	2	3	4	5	6	7	8	9

En este ejemplo se ha visto que el usuario ha tecleado exactamente 9 caracteres, que es el máximo que puede almacenar el *string* sin contar el carácter '\0'. Si se hubiesen tecleado menos de 9 caracteres, el carácter de final de cadena aparecería ubicado en alguna de las posiciones más a la izquierda, tal y como ocurría en los ejemplos anteriores de *Mercurio* y *Ana Rosa*. Los problemas pueden venir cuando el usuario teclea 10 o más caracteres, ya que sólo se capturan los 9 primeros, se almacena el carácter '\0' en la última posición del *string*, y los caracteres sobrantes permanecen en el *buffer* de teclado esperando a ser leídos (o eliminados con una instrucción *fflush*).

Cuando se efectúa una lectura carácter a carácter de los elementos de un *string* es muy sencillo saber cuántos caracteres se han leído de teclado. Dado que en esta situación es necesario utilizar una variable que indique en qué posición de la cadena se va a ubicar cada carácter (y que en el ejemplo de la sección 7.2.3 se llama *i*), una vez finalizada la captura de elementos, esta misma variable será la que indique cuántos caracteres se han capturado.
Sin embargo, cuando se recurre a una lectura directa del *string*, esto es, a la captura de todos los caracteres introducidos por teclado mediante una única instrucción (*scanf* o, por lo general, *gets*) no se dispone del número de caracteres almacenados en la cadena. Afortunadamente, la instrucción de lectura habrá ubicado el carácter de final de cadena inmediatamente después de copiar el último valor tecleado por el usuario. Teniendo en cuenta que las posiciones del array comienzan a numerarse del cero en adelante, la posición en la que se encuentre el carácter '\0' indicará el número de elementos que se han leído. Así, si se retoma el ejemplo anterior en el que se capturaban de golpe los caracteres que ocupa la cadena *Ana Rosa*, se puede observar que el final de cadena está en la octava posición, y precisamente 8 son los caracteres útiles que se han leído.

nombre

'A'	'n'	'a'	' '	'R'	'o'	's'	'a'	'\0'	'¿?'		¿?	¿?
0	1	2	3	4	5	6	7	8	9	…	28	29

Para facilitar la tarea de identificar la cantidad de caracteres válidos de un *string*, existe una función llamada *strlen* que recibe una cadena de caracteres como parámetro de entrada y devuelve un valor entero que se corresponde con la posición en la que se encuentra el carácter '\0', o lo que es lo mismo, el número de caracteres útiles de la cadena de entrada. Unas líneas más adelante se verá a modo de ejemplo una posible implementación de la función *strlen*.

Ejemplo:

A continuación se amplía el código que solicita el nombre al usuario para posteriormente saludarle de forma personalizada con un pequeño mensaje que indica cuántas letras tiene su nombre. Como antes, el texto que teclea el usuario figura en letra cursiva y subrayada para distinguirlo de los mensajes que muestra el programa en pantalla.

```
char nombre[30];
int longitud;

printf("Introduce tu nombre, por favor: ");
fflush(stdin);
gets(nombre);
longitud = strlen(nombre);
printf("Hola, %s. Tu nombre ocupa %d caracteres.", nombre,
longitud);
```

Pantalla
```
Introduce tu nombre, por favor: Ana Rosa
Hola, Ana Rosa. Tu nombre ocupa 8 caracteres.
```

3 LAS CADENAS DE CARACTERES Y LAS FUNCIONES

A la hora de desarrollar una función con algún parámetro de tipo *string* hay que tener en cuenta todo lo dicho para las tablas o vectores numéricos.

- En la llamada a la función, para referirse al parámetro real, nunca se deben utilizar los corchetes. Sirvan como ejemplos las llamadas a las funciones *gets* y *strlen* efectuadas en el epígrafe anterior.
- En el prototipo y en la definición de la función hay que indicar que el parámetro formal es una cadena de caracteres, aunque no es necesario especificar su longitud. Para ello, bastará con utilizar [ ], esto es, los corchetes vacíos.

Ejemplo: la función *strlen*

A continuación se muestra el prototipo y una posible definición de la función *strlen*, que, como ya se ha visto, recibe una cadena de caracteres como parámetro de entrada y devuelve un entero. Este número es la posición en la que se encuentra el carácter '\0' dentro del *string*, o lo que es lo mismo, el número de caracteres útiles que tiene la cadena de entrada.

El prototipo de la función *strlen* podría ser el siguiente:

```
int strlen (char cad[]);
```

Como se acaba de recordar, no es necesario especificar el número de elementos del *string*, ni en el prototipo ni en la definición de la función. Aunque podría haberse indicado un tamaño para la cadena de caracteres, en este caso se ha utilizado "cad [ ]", lo que ofrece una gran ventaja: sea cual sea el tamaño con el que se haya definido la cadena que se le pase como parámetro (cadena[30], nif[10], nombre[N], ...), la función *strlen* servirá para calcular su tamaño útil.

Una posible definición o cuerpo de la función *strlen* es la que se presenta a continuación:

```
int strlen (char cad[])
 {
   int i;
   i = 0;
   while (cad[i] != '\0')
    {
      i = i + 1;
    }
   return (i);
 }
```

Puede observarse que, dado que el valor que conviene buscar y devolver en la función coincide con la posición en la que se encuentra el carácter '\0', lo único que hay que hacer es inicializar un índice (*i*) e incrementarlo hasta encontrar el primer valor en el que la tabla, indexada en *i*, contenga el carácter de final de cadena. Evidentemente, si esa posición no existiera, es decir, si ninguna de las posiciones del *string* tuviera el '\0', entonces la implementación aquí presentada daría problemas. No obstante, se ha supuesto desde un primer momento que dicha situación no puede tener lugar, bien porque se ha inicializado la

cadena con una llamada a *gets* o *scanf*, o porque quien haya programado la llamada se ha asegurado de incluir explícitamente el final de cadena.

3.1 La librería *string.h*

Cuando se vaya a trabajar con cadenas de caracteres será necesario añadir la línea siguiente:

```
#include <string.h>
```

Incluir esta línea en un programa permite utilizar las funciones definidas para el tratamiento de *strings*, entre las que cabe mencionar:

int strlen(char cad[]);

- Como ya se ha visto, esta función devuelve la longitud útil de una cadena de caracteres, que coincide con la posición en la que tiene el carácter '\0'.

void strcmp (char cad1[ ], char cad2[ ]);

- Esta función compara las cadenas cad1 y cad2 y devuelve el valor 0 si son iguales. Si son distintas devuelve otro valor, que dependiendo de lo que contengan ambos *strings* de entrada puede ser negativo o positivo.

void strncpy (char cad1[ ], char cad2[ ], int n);

- Esta función copia los primeros *n* caracteres de *cad2* sobre *cad1*.

4 EJERCICIO RESUELTO

4.1 Programa que determina si dos cadenas de caracteres son iguales

Este ejercicio consiste en pedir al usuario dos cadenas de caracteres, que se almacenarán en sendos *strings*, y después determinar si ambas son iguales. Supondremos que las cadenas no tendrán más de 50 caracteres.
La idea es que si, una vez leidas, las cadenas no tienen el mismo número de caracteres útiles significará que son diferentes, mientras que en caso contrario (es decir, si ambas tienen la misma longitud) sólo se considerarán iguales si contienen exactamente los mismos caracteres en el mismo orden.
Para hacer más interesante este ejercicio, no se utilizará la función *strcmp* presentada anteriormente. En su lugar, se implementará una función con la siguiente cabecera:

int son_iguales (char cad1[ ], char cad2[ ]);

- Esta función compara las cadenas cad1 y cad2, y devuelve el valor 1 si son iguales o el valor 0 si no lo son. Se supone que ambas cadenas de entrada (cad1 y cad2) incluyen el carácter fin de cadena en alguna de sus posiciones.

Una posible solución al ejercicio podría ser la que viene a continuación:

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

int son_iguales (char cad1[], char cad2[])
 {
   int long1, long2, i, iguales;

   long1 = strlen (kat1);
```

```
   long2 = strlen (kat2);

   if (long1 != long2)
    {
      return(0);
    }
   else
    {
      i = 0;
      iguales = 1;
      while ((iguales == 1) && (i<long1))
       {
         if (cad1[i] != cad2[i])
          {
            iguales = 0;
          }
         i = i + 1;
       }
      return (iguales);
    }
 }
```

```
void main()
{
  /* Declaración de las variables */
  char cadcar1[51];
  char cadcar2[51];
  int decision;
  /* Petición de las cadenas de entrada */
  printf ("\nEscribe la primera cadena (máximo 50 caracteres): ");
  gets (cadcar1);
  printf ("Escribe la segunda cadena (máximo 50 caracteres): ");
  gets (cadcar2);
                                                                    →
  /* Decisión de si son iguales */
  decision = son_iguales (cadcar1, cadcar2);

  /* Presentación del resultado */
  if (decision == 1)
   {
     printf("\nLas cadenas %s y %s son iguales.",cadcar1,cadcar2);
   }
  else
   {
     printf("\nLas cadenas %s y %s son distintas.",cadcar1,cadcar2);
   }
  system("pause");
}
```

En este ejemplo, el programa principal efectúa los pasos típicos que se han visto hasta ahora: pide valores al usuario, utiliza la función *son_iguales* para obtener una respuesta, y, dependiendo del valor devuelto por la llamada, escribe un mensaje u otro por pantalla. La función *son_iguales*, por su parte, evalúa primero las longitudes de las cadenas que recibe como parámetro, de modo que si son diferentes devuelve un cero, lo que indica que las cadenas no son iguales. En caso contrario, inicializa la variable *iguales* con el valor 1, y compara posición a posición los caracteres de ambas cadenas, de manera que sólo si encuentra una posición para la que los caracteres son diferentes en cada *string* sobreescribe dicha variable con un 0. Así, una vez recorridas todas las posiciones útiles de las cadenas, si *iguales* sigue valiendo 1, es decir, en ningún momento se ha puesto a 0, significa que no se ha encontrado diferencias entre las cadenas, mientras que en caso contrario, al menos se ha identificado una posición con caracteres distintos. Es por este motivo que en el primer caso la función debería devolver un 1, y en el segundo un 0, de ahí que lo que se haga, por simplificar, es devolver directamente el valor de la variable *iguales*.