Fundamentos de Programación – Programación Estructurada en C:
8.- CADENAS DE CARACTERES (STRINGS)

Maider Huarte Arrayago

Fundamentos de Programación – Programación Estructurada en C:

8.- CADENAS DE CARACTERES (STRINGS)

8.- CADENAS DE CARACTERES (STRINGS)

8.1.- INTRODUCCIÓN

Una cadena de caracteres o string, es una **array de caracteres que termina con** un carácter especial, que es el carácter NULL, **'\0'**. La diferencia con un array de caracteres normal es entonces esa, que un array de caracteres normal no tiene porqué terminar con el carácter '\0'.

Los strings se declaran como los arrays de tipo *char*, que es lo que al fin y al cabo son. A la hora de reservar memoria para el string, hay que tener en cuenta que el último carácter será '\0', es decir, si esperamos una cadena de 14 caracteres, declararemos una variable string con 15 elementos, para meter '\0' en la última posición.

La declaración e inicialización de strings se puede hacer de dos formas:

```
char cadena[5]={'H','O','L','A','\0'}; //4 caracteres + '\0'
```

o

```
char cadena[5]="HOLA";  //4 caracteres + '\0'
```

En la primera forma, se ha dado valor a cada uno de los caracteres de forma separada. Para ello, se escribe entre llaves ({...}) una lista de caracteres separados por comas, teniendo en cuenta que los caracteres serán asignados a los elementos del array según aparezcan en la lista.

En la segunda forma, se ha usado una **constante cadena** (punto 1.3.1.2.- Constantes del tema **1.- INTRODUCCIÓN AL LENGUAJE C**). Los caracteres que forman la constante cadena, se introducen en los elementos del string, según el orden en el que aparecen. No hace falta indicar el carácter '\0', porque está implícito en la constante cadena (las comillas dobles indican que el último carácter de la cadena es '\0').

Al inicializar el string en el momento de la declaración, no es necesario indicar el tamaño del array. Incluso, se puede indicar un tamaño e inicializar el string con menos caracteres que los indicados en el tamaño. Es decir, lo siguiente sería válido:

```
char cadena_0[]="Esto es una cadena"; //Esto reserva memoria para 19 elementos char
                                      //seguidos, los 18 de la frase y el último carácter '\0'

char cadena_1[19]="Esto es"; //Esto reserva memoria para 19 elementos char, en los 7
                             //primeros introduce los caracteres 'E', 's', 't', 'o', ' ', 'e', 's',
                             //y el 8º elemento será '\0'. El resto de elementos no se
                             //habrán inicializado.
```

```
char cadena_2[]=""; //Esto reserva memoria para un sólo elemento, el '\0'.
```

En el primero de los casos, el número de bytes reservados (la capacidad) para el string, lo determina el tamaño de la constante cadena que se ha usado para inicializarlo. Así, aunque se cambie el valor del string a lo largo del programa, siempre tendrá el mismo número de bytes reservados (19), y el rango de los índices siempre será el mismo (rango [0,18]).

En el segundo caso, se ha reservado la capacidad del string de forma explícita ([19]), pero la inicialización se ha hecho con menos elementos.

En cualquier caso, el compilador siempre ha de tener claro cuál es el rango válido para los elementos del array. Por eso, si no se indica la capacidad en la declaración, hay que hacerlo inicializando el contenido del string.

Tal y como ocurría con el resto de arrays, una cosa es la capacidad de elementos que puede contener (que depende del número de bytes reservados en la declaración) y otra cosa es cuántos elementos válidos contiene en un momento dado, es decir, el tamaño. Para los arrays de cualquier otro tipo, había que tener en cuenta ese tamaño mediante una segunda variable, por ejemplo, de tipo *int*. Con los strings eso no es necesario, ya que basta con buscar el **primer** valor '\0', para saber que ahí se acaban los elementos válidos (el resto, se ignoran).

8.2.- ARRAYS DE STRINGS

Los arrays de strings, son matrices (arrays bidimensionales), en los que cada fila está constituida por un string. Eso quiere decir que en cada fila habrá una cadena de caracteres que termina en '\0'. Así, son diferentes a las matrices de caracteres normales, donde ningún elemento de una fila ha de ser '\0'.

Sintaxis de declaración:

char <nombre_matriz>**[**<nº cadenas>**][**< capacidad_cadena>**];**

La inicialización, se puede hacer dando valores a cada uno de los strings de cada fila, de las dos formas diferentes que se han visto en **8.1.- INTRODUCCIÓN**, es decir, dando valor a cada elemento o al string completo. De cualquiera de las formas, se han de inicializar todas las cadenas de la matriz, separándolas por comas, encerrando el bloque entre llaves y terminando la declaración con ;.

En la declaración de una matriz de strings, aunque se haga también una inicialización, hay que especificar el capacidad_cadena, no se puede dejar el 2º corchete vacío como en la declaración de strings.

Ejemplo:

```
char cadenas_0[2][5];

char cadenas_1[2][5]={“Uno”,“Dos”};

char cadenas_2[][5]={‘U’,’n’,’o’,’\0’,’.’,’D’,’o’,’s’,\0’,’.’};

/* En memoria:
```

Variable	Dirección	Memoria	Tamaño
cadenas_0[0][0]	1245008	-	1 byte
cadenas_0[0][1]	1245009	-	1 byte
cadenas_0[0][2]	1245010	-	1 byte
cadenas_0[0][3]	1245011	-	1 byte
cadenas_0[0][4]	1245012	-	1 byte
cadenas_0[1][0]	1245013	-	1 byte
cadenas_0[1][1]	1245014	-	1 byte
cadenas_0[1][2]	1245015	-	1 byte
cadenas_0[1][3]	1245016	-	1 byte
cadenas_0[1][4]	1245017	-	1 byte
cadenas_1[0][0]	1245024	‘U’	1 byte
cadenas_1[0][1]	1245025	‘n’	1 byte
cadenas_1[0][2]	1245026	‘o’	1 byte
cadenas_1[0][3]	1245027	‘\0’	1 byte
cadenas_1[0][4]	1245028	-	1 byte
cadenas_1[1][0]	1245029	‘D’	1 byte
cadenas_1[1][1]	1245030	‘o’	1 byte
cadenas_1[1][2]	1245031	‘s’	1 byte
cadenas_1[1][3]	1245032	‘\0’	1 byte
cadenas_1[1][4]	1245033	-	1 byte
cadenas_2[0][0]	1245040	‘U’	1 byte
cadenas_2[0][1]	1245041	‘n’	1 byte
cadenas_2[0][2]	1245042	‘o’	1 byte
cadenas_2[0][3]	1245043	‘\0’	1 byte
cadenas_2[0][4]	1245044	‘.’	1 byte
cadenas_2[1][0]	1245045	‘D’	1 byte
cadenas_2[1][1]	1245046	‘o’	1 byte
cadenas_2[1][2]	1245047	‘s’	1 byte
cadenas_2[1][3]	1245048	‘\0’	1 byte
cadenas_2[1][4]	1245049	‘.’	1 byte

Se pueden usar también arrays de punteros a strings. La diferencia con las matrices de strings, será que no hay que especificar el tamaño de los strings en la declaración del array, sólo hay que especificar el tamaño del propio array, al fin y al cabo, se trata de un array de punteros. Además, los strings a los que apuntan los punteros pueden tener diferente capacidad, es decir, se tendrá una matriz cuyas filas tienen diferente tamaño. No hay porqué inicializar todos los strings en la declaración:

```
char *cadenas_3[2]={“Tres”,”Cuatro”};

char cadenas_4[2];
```

/* En memoria:

Variable	Dirección	Memoria	Tamaño
*cadenas_4[0]	1244992 ... 1244995	-	4 bytes
*cadenas_4[1]	1244996 ... 1244999	-	4 bytes
*cadenas_3[0]	1245000 ... 1245003	4321320	4 bytes
*cadenas_3[1]	1245004 ... 1245007	4321308	4 bytes
cadenas_3[1][1]	4321308	'C'	1 byte
cadenas_3[1][2]	4321309	'u'	1 byte
cadenas_3[1][3]	4321310	'a'	1 byte
cadenas_3[1][4]	4321311	't'	1 byte
cadenas_3[1][5]	4321312	'r'	1 byte
cadenas_3[1][6]	4321313	'o'	1 byte
cadenas_3[1][7]	4321314	'\0'	1 byte
cadenas_3[0][0]	4321320	'T'	1 byte
cadenas_3[0][0]	4321321	'r'	1 byte
cadenas_3[0][1]	4321322	'e'	1 byte
cadenas_3[0][2]	4321323	's'	1 byte
cadenas_3[0][3]	4321324	'\0'	1 byte

*/

Tal y como se ve en el ejemplo, los strings que conforman cada fila no tienen por qué haberse reservado seguido en memoria, ni siquiera en orden (el string de la fila 1 se ha reservado en posiciones de memoria anteriores que el string de la fila 0). Si se cambiara el valor de los strings de las filas de la matriz, habrá que tener en cuenta que las posiciones de memoria reservadas para cada una de ellas no han de sobrepasar las reservadas en la declaración, como hay que tener en cuenta con todo tipo de arrays. Para la fila 0, se han reservado 5 bytes de memoria, y para la fila 1, 7; de esos bytes reservados, habrá que tener en cuenta también, que un byte será para '\0'.

El acceso a cada uno de los elementos del array de strings, se hace de la misma forma que con cualquier matriz de otro tipo de dato, es decir, indicando los índices del elemento entre corchetes.

Ejemplo:

```
char cadenas_2[][5]={'U','n','o','\0','.','D','o','s',\0','.'};

char *cadenas_3[2]={"Tres","Cuatro"};

cadenas_2[0][0];
cadenas_2[0][1];
cadenas_2[1][1];

cadenas_3[0][0];
cadenas_3[0][1];
cadenas_3[1][1];

cadenas_2[0];
cadenas_3[0];
```

8.3.- FUNCIONES DE ENTRADA/SALIDA ESTÁNDAR PARA CADENAS

8.3.1.- Función *printf*

El funcionamiento de esta función se vio en el punto **3.2.- FUNCIÓN *printf*** del tema **3.- FUNCIONES DE ENTRADA Y DE SALIDA**. Para indicar que se saque un string mediante la función *printf*, lo que se hace es poner "%s" en la posición correspondiente de la cadena de control pasada a *printf* como primer argumento.

Para indicar el valor por el que debe sustituir *printf* la expresión de control "%s", se le pasará como argumento el nombre del string, que es la dirección del array que conforma el string. Esto difiere en la forma de indicar los valores de otros tipos de variables, ya que en aquellos casos no se pasaba ninguna dirección, sino el valor de la variable en sí. Hay que entender, que la expresión "%s" espera un argumento de tipo dirección de un string, es decir, una dirección a partir de la cual hay una serie de caracteres seguidos, siendo '\0' el último de la secuencia. Si se quisiera sacar por pantalla uno sólo de los elementos, se usaría la expresión "%c" en la cadena de control, que no espera un puntero, sino el valor de un carácter concreto.

Ejemplos:

```
#include <stdio.h>

int main()
{
  char cadena[]="otra cadena";

  char matriz[5][19]={"Primera cadena","Segunda cadena","Tercera cadena","Cuarta cadena",
                      "Quinta cadena"};

  printf("esto es %s","una cadena"); //En pantalla esto es una cadena
  printf("\nesto es %s",cadena); //En pantalla esto es otra cadena

  printf("\nesto es la fila 0: %s",matriz[0]);
  printf("\nesto es la fila 0: %s",matriz);
  printf("\nesto es la fila 1: %s",matriz[1]);

  printf("\nesto es el elemento (0,0): %c",matriz[0][0]);
  printf("\nesto es el elemento (0,0): %c",*(matriz[0]));
  printf("\nesto es el elemento (0,0): %c",**matriz);

  return 0;
}
```

Si la dirección de memoria que se indica a *printf*, no fuese realmente la de inicio de un string, *printf* sacaría por pantalla todos los bytes que encontrase en memoria, hasta el primer '\0', interpretados como caracteres.

8.3.2.- Función *scanf*

El uso de *scanf* no requiere ningún cambio sobre lo visto anteriormente. Se usa también la expresión "%s" para indicar que se recogerá un string de la entrada estándar. Los caracteres introducidos por el usuario, se van leyendo del buffer de entrada, y metiendo como valores de los elementos del string, hasta encontrar un carácter de espacio en blanco, una tabulación o '\r'. Al leer alguno de esos caracteres, no se introduce en el array, sino que se introduce el carácter '\0', convirtiendo el array de tipo *char* en string. Después, se deja de leer del buffer de entrada.

Hay que tener en cuenta, que debido a que los espacios en blanco, tabulaciones y '\r' se convierten en '\0' finalizando el string, con *scanf* y "%s" sólo se pueden leer cadenas de caracteres de una sola palabra, no frases.

Para asegurarnos de que el usuario no introduce más caracteres que los que caben en el string, hay que indicar la anchura del campo como *capacidad*-1. Si no, podrían sobrescribirse posiciones de memoria que podrían ser de otras variables.

Ejemplo:

```
#include <stdio.h>

int main()
{

   char cadena[20];
   char matriz[5][19];

   scanf("%19s",cadena);

   scanf("%18s",matriz[0]);
   scanf("%18s",matriz);

   return 0;
}
```

8.3.3.- Función *puts*

Prototipo:

```
void puts(<string>);
```

Se trata de una función que se encuentra en la librería *stdio*. Esta función saca por la salida standard (la pantalla), el string que se le pasa como argumento.

Ejemplo:

```
char cadena[5]="Hola";
puts(cadena);
```

Si lo que se le pasara como argumento no fuese realmente un string, sino un array de caracteres, lo que sacaría por pantalla sería los elementos del array, y el contenido de las siguientes posiciones de memoria, interpretadas como caracteres, hasta encontrar un carácter '\0'.
Ejemplo:

```
char cadena[4]={'H','o','l','a'};
puts(cadena);
```

8.3.4.- Función *gets*

Prototipo:

```
void gets(<string>);
```

Se trata de una función que se encuentra en la librería *stdio*. Esta función toma todo lo introducido por la entrada estándar (el teclado), hasta encontrar un '\r', y lo introduce en el array de caracteres que se le pasa como parámetro. Antes de introducir el '\r', lo convierte en '\0', convirtiendo así el array de caracteres en un string.

Ejemplo:

```
char cadena[20];
gets(cadena);
```

Si el usuario introdujera más elementos carácter de hasta el '\r', de los que tiene reservados la variable string, se perdería la información que hubiera en las posiciones de memoria siguientes a las reservadas para el string, igual que ocurría con los arrays de otros tipos de datos. Por eso el uso de esta función no es muy adecuado, ya que no podemos asegurarnos de que el usuario ha introducido una cantidad correcta de caracteres.

La única forma de leer en un string una línea introducida por teclado que aprenderemos en la asignatura, sin tener problemas con los espacios en blanco ni tabulaciones, y controlando que no se sobrepase la memoria reservada para el string, es con la función *fgets*. Se verá en el tema de **11.- Ficheros**

8.4.- FUNCIONES ESPECÍFICAS PARA CADENAS DE CARACTERES

Existen una serie de funciones específicas para cadenas de caracteres, que explicaremos a continuación. Se encuentran en la librería *string*.

8.4.1.- strlen(string)

Prototipo:

```
int strlen(<string>);
```

Devuelve el nº de caracteres del string pasado como argumento, sin tener en cuenta el carácter '\0'. Lo que hace es contar los caracteres que hay desde la dirección indicada, hasta el primer carácter '\0' encontrado, sin tener éste último en cuenta.

Ejemplo:

```
char cadena[]="Hola";
int longitud;

longitud=strlen(cadena);
```

La existencia de esta función supone una ventaja sobre los otros tipos de arrays, en los que para controlar la longitud, se necesitaba trabajar, aparte de con el array, con una variable en el que se guardara el tamaño. Al pasar un array completo por referencia a una función, por ejemplo, es necesario también pasar el tamaño real del array. Los strings no necesitan de esa variable adicional, porque su final está marcado por el carácter '\0', y se puede calcular su tamaño mediante esta función.

8.4.2.- strcmp(string1,string2)

Prototipo:

```
int strcmp(<string1>,<string2>);
```

Compara los strings pasados como argumentos, y devuelve un valor entero indicando el resultado de la comparación. La comparación se hace con los valores ASCII de los caracteres, con lo que se tienen en cuenta espacios en blanco y se consideran diferentes las mayúsculas y minúsculas.

Si lo devuelto es:

- Valor 0: las dos cadenas son exactamente iguales.
- Valor <0: El 1er string es alfabéticamente menor (en orden alfabético, va antes) que el 2º (teniendo en cuenta los valores ASCII, no el tamaño de los strings).
- Valor >0: El 1er string es alfabéticamente mayor (en orden alfabético, va después) que el 2º (teniendo en cuenta los valores ASCII, no el tamaño de los strings).

Ejemplo:

```
char cadena1[5]="Hola";
char cadena2[6]="12345";
int comp;

comp=strcmp(cadena1,cadena2);
```

8.4.3.- strcpy(string1,string2)

Prototipo:

```
char *strcpy(<string destino>,<string origen>);
```

Copia físicamente, el contenido del string origen en el string destino. La capacidad de string destino debe ser igual o mayor que string origen, para que no se sobrescriban posiciones de memoria no reservadas para string destino.

Devuelve la dirección inicial del string destino, si no hubo ningún problema.

Ejemplo:

```
char cadena1[10],cadena2[8]=”Hola”;

strcpy(cadena1,cadena2);
```

Como con cualquier tipo de array, no se puede usar el operador asignación para dar el valor de un string a otro, como se haría con cualquier otro tipo de variable. En el caso de arrays normales, la copia de elementos ha de hacerse de uno en uno, usando alguna estructura repetitiva, pero en el caso de los strings, el uso de esta función hace más cómoda la asignación de valores.

Ejemplo:

```
char cadena[8];
char *compr;

cadena=”string”; //ERROR!

strcpy(cadena,”string”); //OK

compr= strcpy(cadena,”string”);
```

8.4.4.- strchr(string,char)

Prototipo:

```
char *strchr(<string>,<carácter>);
```

Esta función devuelve la dirección de la primera posición del string en la que se encuentre un carácter como el pasado como segundo argumento.

Ejemplo:

```
char *p;

p=strchr("Esto es una prueba",'u');

printf("%s",p); //saca por pantalla "una prueba"
```

Los cambios que se hagan sobre la subcadena apuntada por el puntero obtenido, se realizarán físicamente sobre la cadena original.