Manejo de Cadenas de Caractes/Texto

Existe un tipo *string* como en otros lenguajes, pero no existe un tipo de datos específico para almacenar texto, sino que se utilizan *arrays* de char. Funcionan igual que los demás *arrays* con la diferencia de que ahora se manejan letras en vez de números. Se les llama cadenas, ***strings*** o tiras de caracteres y a partir de ahora les llamaremos **cadenas**.

Para declarar una cadena se hace como en el caso de un *array*. Por ejemplo, si queremos declarar una cadena de longitud 20 caracteres se haría:

```
char texto[20];
```

Al igual que en los *arrays*, no podemos entonces introducir más de 20 elementos en la cadena. Vamos a ver un ejemplo para mostrar el nombre del usuario en pantalla:

```
#include <stdio.h>

main(){
        char nombre[20];

        printf( "Introduzca su nombre (20 letras máximo): " );
        scanf( "%s", nombre );
        printf( "\nEl nombre que ha escrito es: %s\n", nombre );
}
```

Obsérvese que en la sentencia scanf no se usa el símbolo & delante de nombre. No hace falta porque se trata de un *array*, de modo que escribir el nombre del *array* es equivalente a poner &nombre[0].

También puede llamar la atención sobre la forma de imprimir el *array*. Con sólo usar %s ya se imprime su totalidad. Ya veremos esto más adelante.

Por si alguien está acostumbrado a programar en otro lenguaje es preciso hacer notar que en C no se puede hacer esto:

```
#include <stdio.h>

main(){
        char texto[20];
        texto = "Hola";
}
```

Es interesante saber cómo funciona una cadena por dentro, por eso vamos a ver primero cómo se inicializa:

```
#include <stdio.h>

main(){
        char nombre[] = "Gandalf";
```

```
        printf( "Texto: %s\n", nombre );
        printf( "Tamaño de la cadena: %i bytes\n", sizeof nombre );
}
```

Resultado al ejecutar:

```
Texto: Gandalf
Tamaño de la cadena: 8 bytes
```

Curiosamente la respuesta nos dice que "Gandalf" ocupa 8 bytes. Como cada elemento char ocupa un byte eso quiere decir que la cadena tiene 8 elementos, a pesar de que "Gandalf" sólo cuenta con 7 letras. La razón de esta aparente paradoja estriba en que la cadena tiene como carácter final el símbolo **'\0'**, cuyo significado es "fin de cadena". De esta forma, cuando queremos escribir la cadena basta con usar %s y el compilador sabe cuántos elementos debe escribir: hasta que encuentre '\0'.

El programa anterior sería equivalente a:

```
#include <stdio.h>

main(){
        char nombre[] = { 'G', 'a', 'n', 'd', 'a', 'l', 'f', '\0' };
        printf( "Texto: %s\n", nombre );
        printf( "Tamaño de la cadena: %i bytes\n", sizeof nombre );
}
```

Aquí ya se observa claramente que nombre tiene 8 elementos. Pero, ¿qué pasaría si no pusiéramos '\0' al final?

```
#include <stdio.h>

main(){
        char nombre[] = { 'G', 'a', 'n', 'd', 'a', 'l', 'f' };
        printf( "Texto: %s\n", nombre );
        printf( "Tamaño de la cadena: %i bytes\n", sizeof nombre );
}
```

En mi ordenador se obtiene:

```
Texto: Gandalf-
Tamaño de la cadena: 7 bytes
```

Pero en otro después de "Gandalf" puede aparecer cualquier cosa. Lo que aquí sucede es que printf no encuentra el símbolo '\0' y no sabe cuándo dejar de imprimir. Afortunadamente, cuando introducimos una cadena se hace de la primera forma y el C se encarga de poner el símbolo al final.

Es importante no olvidar que la longitud de una cadena es la longitud del texto más el símbolo de fin de cadena. Por eso cuando definamos una cadena tenemos que reservarle un espacio adicional. Por ejemplo:

```
char nombre[8] = "Gandalf";
```

Funciones de manejo de cadenas

Existen unas cuantas funciones el la biblioteca estándar de C para el manejo de cadenas:

- strlen
- strcpy
- strcat
- sprintf
- strcmp

Para usar estas funciones en nuestro programa hay que añadir la directiva:

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

strlen

*size_t *strlen(const char *cadena);*

Esta función devuelve el número de caracteres que tiene la cadena (sin contar el '\0').

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

main(){
        char texto[]="Gandalf";
        int longitud;
        longitud = strlen(texto);
        printf( "La cadena \"%s\" tiene %i caracteres.\n", texto, longitud );
}
```

Como ejemplo, vamos a ver cómo se programaría esta función si no dispusiéramos de ella. Hay más información en Recorrido de cadenas con punteros.

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

main(){
        char texto[]="Gandalf";
        char *p;
        int longitud=0;

        p = texto;
        while (*p != '\0') {
                longitud++;
                printf( "%c\n", *p );    /* Mostramos la letra actual */
                p++;                                    /* Vamos a la siguiente letra */
```

```
        }
        printf( "La cadena \"%s\" tiene %i caracteres.\n", texto, longitud );
}
```

Para medir la longitud de la cadena usamos un puntero para recorrerla (el puntero p). Hacemos que p apunte a texto. Luego entramos en un bucle while. La condición del bucle comprueba si se ha llegado al fin de cadena ('\0'). Si no es así, suma 1 a longitud, muestra la letra por pantalla e incrementa el puntero en 1 (con lo que pasamos a la siguiente letra).

strcpy

*char *strcpy(char *cadena1, const char *cadena2);*

Copia el contenido de *cadena2* en *cadena1*. *cadena2* puede ser una variable o una cadena directa (por ejemplo, "hola"). Debemos tener cuidado de que la cadena destino (*cadena1*) tenga espacio suficiente para albergar a la cadena origen (*cadena2*).

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

main(){
        char texto[] = "Éste es un curso de C.";
        char destino[50];
        strcpy( destino, texto );
        printf( "Valor final: %s\n", destino );
}
```

Vamos a ver otro ejemplo en el que la cadena destino es una cadena constante ("Éste es un curso de C.") y no una variable. Además, en este ejemplo vemos que la cadena origen es sustituida por la cadena destino totalmente. Si la cadena origen es más larga que la destino, se eliminan las letras adicionales.

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

main(){
        char destino[50]="Éste no es un curso de HTML, sino de C.";
        printf( "%s\n", destino );
        strcpy( destino, "Éste es un curso de C." );
        printf( "%s\n", destino );
}
```

strcat

*char *strcat(char *cadena1, const char *cadena2);*

Añade la cadena2 al final de la cadena1 (concatena).

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

main(){
        char nombre_completo[50];
        char nombre[]="Gandalf";
        char apellido[]="el Gris";

        strcpy( nombre_completo, nombre );
        strcat( nombre_completo, " " );
        strcat( nombre_completo, apellido );
        printf( "El nombre completo es: %s.\n", nombre_completo );
}
```

Como siempre, tenemos asegurar que la variable en la que añadimos las demás cadenas tenga el tamaño suficiente. Con la primera línea de este programa introducimos el nombre en nombre_completo. Usamos strcpy para asegurarnos de que queda borrado cualquier dato anterior. Luego usamos un strcat para añadir un espacio y, finalmente, introducimos el apellido.

sprintf

*int sprintf(char *destino, const char *format, ...);*

Funciona de manera similar a printf pero, en vez de mostrar el texto en la pantalla, lo guarda en una variable (destino). El valor que devuelve (int) es el número de caracteres guardados en la variable destino.

Con sprintf podemos repetir el ejemplo de strcat de manera más sencilla:

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

main(){
        char nombre_completo[50];
        char nombre[]="Gandalf";
        char apellido[]="el Gris";

        sprintf( nombre_completo, "%s %s", nombre, apellido );
        printf( "El nombre completo es: %s.\n", nombre_completo );
}
```

Se puede aplicar a sprintf todo lo indicado para printf.

strcmp

*int strcmp(const char *cadena1, const char *cadena2);*

Compara cadena1 y cadena2. Si son iguales, devuelve 0. Un número negativo si cadena1 "va" antes que cadena2, y un número positivo si es al contrario:

- < 0 si cadena1 < cadena2
- ==0 si cadena1 == cadena2
- > 0 si cadena1 > cadena2

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

main(){
        char nombre1[]="Gandalf";
        char nombre2[]="Frodo";

        printf( "Comparación con strcmp: %i\n", strcmp(nombre1,nombre2));
}
```

El resultado es:

```
Comparación con strcmp : 1
```

Arrays de cadenas

Un *array* de cadenas puede servirnos para agrupar una serie de mensajes. Por ejemplo, todos los mensajes de error de un programa. Luego, para acceder a cada mensaje, basta con usar su número.

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

int error( int num_err ){
        char *errores[] = {
                "No se ha producido ningún error",
                "No hay suficiente memoria",
                "No hay espacio en disco",
                "Me he cansado de trabajar"
                };
        printf( "Error número %i: %s.\n", num_err, errores[num_err] );
        exit( -1 );
}
main(){   error( 2 );   }
```

El resultado será:

```
Error número 2: No hay espacio en disco.
```

Un *arra*y de cadenas es en realidad un *array* de punteros a cadenas. El primer elemento de la cadena ("No se ha producido ningún error") tiene un espacio reservado en memoria y errores[0] apunta a ese espacio.

- Ordenación de un *array* de cadenas

Vamos a ver un sencillo ejemplo de ordenación de cadenas. En el ejemplo siguiente tenemos que ordenar una serie de dichos populares:

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

#define ELEMENTOS 5

main(){
        char *dichos[ELEMENTOS] = {
                "La avaricia rompe el saco",
                "Más vale pájaro en mano que ciento volando",
                "No por mucho madrugar amanece más temprano",
                "Año de nieves, año de bienes",
                "A caballo regalado no le mires el diente"
                };
        char *temp;
        int i, j;

        printf( "Lista desordenada:\n" );
        for( i=0; i<ELEMENTOS; i++ )
                printf( "  %s.\n", dichos[i] );

        for( i=0; i<ELEMENTOS-1; i++ )
                for( j=i+1; j<ELEMENTOS; j++ )
                        if ( strcmp(dichos[i], dichos[j]) > 0 ) {
                                temp = dichos[i];
                                dichos[i] = dichos[j];
                                dichos[j] = temp;
                        }

        printf( "Lista ordenada:\n" );
        for( i=0; i<ELEMENTOS; i++ )
                printf( "  %s.\n", dichos[i] );
}
```

Cómo funciona el programa:

- tomamos el primer elemento de la matriz. Lo comparamos con todos los siguientes. Si alguno es anterior, los intercambiamos. Cuando acabe esta primera vuelta tendremos "A caballo regalado no le mires el diente" en primera posición.
- Tomamos el segundo elemento. Lo comparamos con el tercero y siguientes. Si alguno es anterior, los intercambiamos. Al final de esta vuelta quedará "A caballo regalado no le mires el diente" en segunda posición.

Para ver con mayor claridad el resultado a partir del desarrollo del proceso, se va a sustituir cada cadena por su primera letra (menos la de "Año de nieves..." que se sustituirá por Añ). Así, el proceso queda:

	0	1	2	3	3'	4	4'	5	6	6'	7	7'	8	8'	9	9'	10	10'
1	L	**L**	**L**	**L**	**Añ**	**Añ**	**A**											
2	M	**M**	M	M	M	M	M	**M**	**M**	**L**	**L**	**Añ**						
3	N	N	**N**	N	N	N	N	**N**	N	N	N	N	**N**	**M**	**M**	**L**		
4	**Añ**	**Añ**	**Añ**	**Añ**	**L**	L	L	L	**L**	**M**	M	M	**M**	**N**	N	N	**N**	**M**
5	**A**	**A**	**A**	**A**	**A**	**A**	**Añ**	Añ	Añ	Añ	**Añ**	**L**	L	L	**L**	**M**	**M**	**N**

Recorrido de cadenas con punteros

Las cadenas se pueden recorrer de igual forma que se hace con los arrays, usando punteros. Vamos a ver un ejemplo: el siguiente sencillo programa cuenta los espacios y las letras e (minúsculas) que hay en una cadena.

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

main()
{
        char cadena[]="El puerto paralelo del PC";
        char *p;
        int espacios = 0, letras_e = 0;

        p = cadena;
        while (*p != '\0') {
                if (*p == ' ') espacios++;
                if (*p == 'e') letras_e++;
                p++;
        }

        printf( "En la cadena \"%s\" hay:\n", cadena );
        printf( "  %i espacios\n", espacios );
        printf( "  %i letras e\n", letras_e );
}
```

El resultado es:

```
En la cadena "El puerto paralelo del PC" hay:
 4 espacios
 3 letras e
```

Para recorrer la cadena necesitamos un puntero p que sea de tipo char. Debemos hacer que p apunte a la cadena (p=cadena). Así, p apunta a la dirección del primer elemento de la misma. El valor de *p sería, por tanto, 'E'. Comenzamos el bucle. La condición comprueba que no se ha llegado al final de la cadena (*p != '\0'). Entonces se comprueba si en la dirección a la que apunta p hay un espacio o una letra e. Si es así, se incrementan las variables correspondientes. Una vez comprobado esto se pasa a la siguiente letra (p++).

En este otro ejemplo substituimos los espacios por guiones:

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

main(){
        char cadena[]="El puerto paralelo del PC";
        char *p;

        p = cadena;
        while (*p != '\0') {
                if (*p == ' ') *p = '-';
                p++;
        }

        printf( "La cadena queda: \"%s\" \n", cadena );
}
```

y se obtiene:

```
La cadena queda: "El-puerto-paralelo-del-PC"
```

Material tomado de:
http://platea.pntic.mec.es/vgonzale/cyr_0204/cyr_01/control/lengua_C/cadenas.htm#arriba