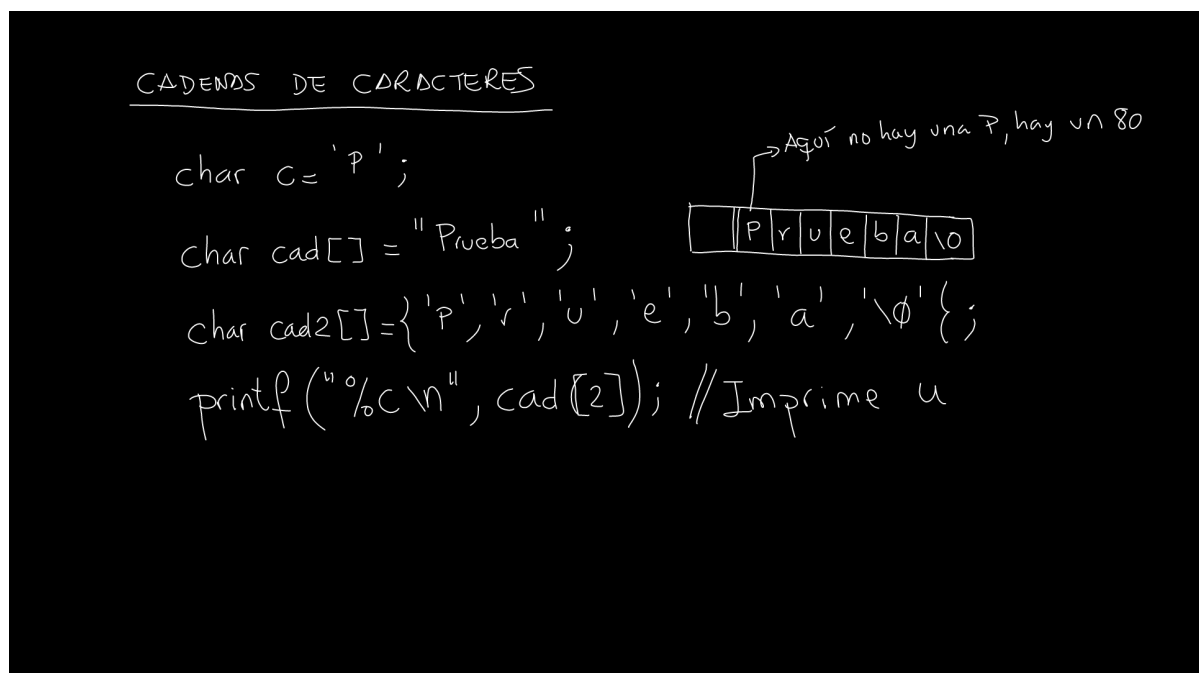


(Versión de fecha 26 de noviembre de 2024)

Nota: En todo lo que sigue en este documento se supone que solo se utilizan caracteres ASCII, esto es, caracteres del alfabeto inglés.

## 1 CARACTERES INDIVIDUALES. EL TIPO CHAR



Para declarar una variable que contiene un carácter individual se utiliza el tipo *char*. Se puede asignar valor mediante un literal entrecomillado. Para ello, se pone el carácter entre **comillas simples**:

```
char c = 'P';
```

También se le puede asignar directamente el código ASCII que le corresponde:

```
char c = 80;
```

En el disco y en la memoria lo que hay no son caracteres, son los códigos numéricos de dichos caracteres en cierto sistema de codificación. En este documento suponemos que el sistema de codificación es el *ASCII*. Por ejemplo, el código de la letra *P* es el número 80. Puedes aprender más sobre el sistema de codificación ASCII en el siguiente enlace:

<https://en.wikipedia.org/wiki/ASCII>

La Imagen 1 muestra los códigos ASCII.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0 ^@ NUL	1 ^A SOH	2 ^B STX	3 ^C ETX	4 ^D EOT	5 ^E ENQ	6 ^F ACK	7 ^G BEL	8 ^H BS	9 ^I HT	10 ^J LF	11 ^K VT	12 ^L FF	13 ^M CR	14 ^N SO	15 ^O SI
	NUL	START OF HEADING	START OF TEXT	END OF TEXT	END OF TRANSM.	ENQUIRY	ACKNOWLEDGE	BELL	BACKSP.	CHARACT. TABULATION	LINE FEED	LINE TABULATION	FORM FEED	CARRIAGE RETURN	SHIFT OUT	SHIFT IN
1	16 ^P DLE	17 ^Q DC1	18 ^R DC2	19 ^S DC3	20 ^T DC4	21 ^U NAK	22 ^V SYN	23 ^W ETB	24 ^X CAN	25 ^Y EM	26 ^Z SUB	27 ^[ ESC	28 ^\ FS	29 ^] GS	30 ^^ RS	31 ^_ US
	DATALINK ESCAPE	DEVICE CONTROL 1	DEVICE CONTROL 2	DEVICE CONTROL 3	DEVICE CONTROL 4	NEG. ACK- NOWLEDGE	SYNCHRO- NIZE	END OF TRANS.	CANCEL	END OF MEDIUM	SUBS- STITUTE	ESCAPE	INFO. SEP. 4	INFO. SEP. 3	INFO. SEP. 2	INFO. SEP. 1
2	32 SPACE	33 ^_excl !	34 ^_quot "	35 ^_num #	36 ^_dollar \$	37 ^_percent %	38 ^_amp &	39 ^_apos '	40 ^_lpar (	41 ^_rpar )	42 ^_ast *	43 ^_plus +	44 ^_comma ,	45 ^_hyphen -	46 ^_period .	47 ^_solidus /
	SPACE	EXCLAM. MARK	QUOT. MARK	NUMBER SIGN	DOLLAR SIGN	PERCENT SIGN	AMPER- SAND	APOS- TROPHE	LEFT PAREN.	RIGHT PAREN.	ASTERISK	PLUS SIGN	COMMA	HYPHEN- MINUS	FULL STOP	SOLIDUS
3	48 DIGIT ZERO	49 DIGIT ONE	50 DIGIT TWO	51 DIGIT THREE	52 DIGIT FOUR	53 DIGIT FIVE	54 DIGIT SIX	55 DIGIT SEVEN	56 DIGIT EIGHT	57 DIGIT NINE	58 colon	59 semi	60 lt	61 equals	62 gt	63 quest
	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	64 commat @	65 A	66 B	67 C	68 D	69 E	70 F	71 G	72 H	73 I	74 J	75 K	76 L	77 M	78 N	79 O
	COMM'IAL AT															
5	80 P	81 Q	82 R	83 S	84 T	85 U	86 V	87 W	88 X	89 Y	90 Z	91 lsqb	92 bsol	93 rsqb	94 hat	95 lowbar
	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	96 grave ,	97 a	98 b	99 c	100 d	101 e	102 f	103 g	104 h	105 i	106 j	107 k	108 l	109 m	110 n	111 o
	GRAVE ACCENT															
7	112 p	113 q	114 r	115 s	116 t	117 u	118 v	119 w	120 x	121 y	122 z	123 {	124 	125 }	126 ~	127 ^? DEL
												L. CURLY BRACKET	VERTICAL LINE	R. CURLY BRACKET	TILDE	DELETE

ASCII code table including entity references, control codes and Unicode names (1.1)

Tom Gibara July 2014

Figura 1: Tabla de códigos ASCII

Cada valor del tipo *char* utiliza un byte de almacenamiento en memoria. En concreto, los códigos ASCII son números entre 0 y 127. Desde el 0 hasta el 31 son *caracteres especiales no imprimibles*. Por ejemplo, el cambio de línea, `\n`, es el código 10 (*line feed*).

Al imprimir una variable del tipo *char* con *printf()*, se puede imprimir la letra que representa o su código numérico. Para imprimir la letra, se emplea la especificación de formato `%c` (carácter); para escribir su código numérico se utiliza la especificación de formato `%d` (número entero).

Observa el código del Ejemplo 1.

## Ejemplo 1 Declaración de caracteres

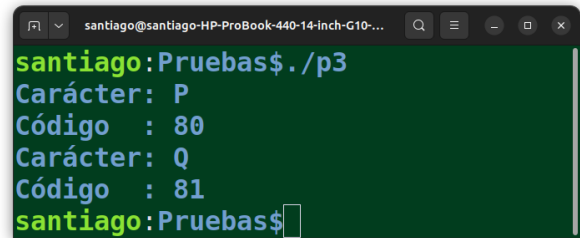
```
#include <stdio.h>

int main() {

    // Asignación de literal entrecomillado
    char c_1 = 'P';
    printf("Carácter: %c\n", c_1);
    printf("Código : %d\n", c_1);

    // Asignación de código ASCII
    char c_2 = 81;
    printf("Carácter: %c\n", c_2);
    printf("Código : %d\n", c_2);

}
```



```
santiago@Pruebas$ ./p3
Carácter: P
Código : 80
Carácter: Q
Código : 81
santiago@Pruebas$
```

## 2 CADENAS DE CARACTERES

En C, una cadena de caracteres (en inglés *character string*, o simplemente *string*) es un array cuyos elementos son del tipo *char* y en el que su último elemento es el carácter `'\0'` (el carácter `'\0'` corresponde al código ASCII 0).

Podemos declarar una cadena y asignarle valor, carácter a carácter:

```
char cad[] = {'A', 'B', 'C', '\0'};
```

También podríamos asignar los códigos ASCII directamente:

```
char cad[] = {65, 66, 67, 0};
```

Observa que, en ambos casos, hay que añadir el carácter `'\0'` al final de la cadena.

También podemos asignar el valor con un literal entrecomillado con comillas dobles:

```
char cad[] = "ABC";
```

Observa que, utilizando un literal entre comillas dobles, no es necesario poner el `'\0'` final, el compilador lo añade automáticamente.

Al tratarse de un array, es posible acceder a los elementos individuales utilizando un índice entre corchetes. Por ejemplo, en la cadena anterior:

`cad[2]`  $\Rightarrow$  Devuelve el carácter B

Para imprimir una cadena de caracteres con *printf()* se utiliza la especificación de formato `%s`. Observa el siguiente ejemplo:

## Ejemplo 2 Declaración de cadenas de caracteres

```
#include <stdio.h>

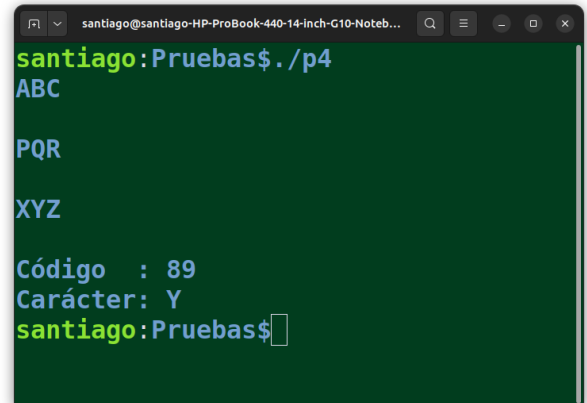
int main() {

    char cad_1[] = {'A', 'B', 'C', '\0'};
    printf("%s\n\n", cad_1);

    char cad_2[] = {80, 81, 82, 0};
    printf("%s\n\n", cad_2);

    char cad_3[] = "XYZ";
    printf("%s\n\n", cad_3);

    char c = cad_3[2];
    printf("Código : %d\n", cad_3[1]);
    printf("Carácter: %c\n", cad_3[1]);
}
```



```
santiago@Pruebas$ ./p4
ABC

PQR

XYZ

Código : 89
Carácter: Y
santiago@Pruebas$
```

### 3 LAS CADENAS DE CARACTERES SON ARRAYS

Los arrays no se pueden copiar con `=` → operador de asignación

```
int x=10;
int y=x; // y vale 10

typedef struct {
    int x,y;
} P2D;
P2D p1;
p1.x=10; p1.y=20;
P2D p2=p1; // p2.x vale 10, p2.y vale 20
```

→ `int p[2]={10,20};`  
~~`int q[2]=p;`~~ → NO FUNCIONA

→ `char cad[]="Prueba";`  
~~`char cad2[]=cad;`~~  
NO FUNCIONA!

En las cadenas de caracteres, igual que sucede con cualquier array, no se puede utilizar el operador de asignación para copiar una cadena en otra. Tampoco se pueden utilizar los operadores de comparación para comparar arrays:

```
char cad_1[] = ``ABC``;
char cad_2 = cad_1; ⇒ ¡ERROR!
cad_2 == cad_1 ⇒ ¡ERROR!
```

Recuerda que con variables del tipo *int*, *double* o caracteres individuales *char*, sí que se pueden crear copias utilizando el signo igual (operador de asignación `=`) y se pueden comparar utilizando operadores de comparación.

Con las variables del tipo *struct*, se pueden crear copias utilizando el operador de asignación, pero no se pueden comparar con operadores de comparación.

## 4 LA LIBRERÍA *STRING.H*

### LA LIBRERÍA *string.h*

- Longitud de una cadena:  
`strlen(cadena)` → nº bytes (sin contar el `\0`)
- Copiar cadenas:  
`strcpy(cad_dest, cad_src);`  
`strncpy(cad_dest, cad_src, n);`
- Comparar cadenas:  
`strcmp(cad1, cad2)` → si son iguales da `0`  
`strncmp(cad1, cad2, n)` → compara n primeros caracteres
- Concatenar:  
`strcat(cad_dest, cad_src);`

La librería *string.h* proporciona funciones utilitarias para trabajar con cadenas de caracteres. En el curso estudiamos las funciones que se muestran en la figura anterior, pero hay algunas más. Puedes consultarlas en la siguiente página Web:

[https://www.tutorialspoint.com/c\\_standard\\_library/string\\_h.htm](https://www.tutorialspoint.com/c_standard_library/string_h.htm)

Se muestran a continuación algunos ejemplos en los que se muestra la utilización de las funciones que estudiamos en el curso:

**`strlen()`: número de caracteres de una cadena**

Esta función devuelve la longitud de una cadena de caracteres medida en *bytes*. Si la codificación utilizada es ASCII, el número de bytes coincide con el número de caracteres.

El Ejemplo 3 muestra una función utilitaria, llamada *disp\_bytes()*, que permite mostrar en pantalla los códigos de los caracteres individuales de una cadena. La función hace uso de la función *strlen()* para saber cuántos caracteres tiene que mostrar en pantalla:

### Ejemplo 3 Utilización de `strlen()`

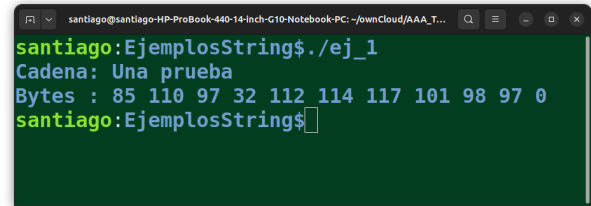
```
#include <stdio.h>
#include <string.h>

void disp_bytes(char cad[]) {
    for(int i=0; i<=strlen(cad); i++) {
        printf("%d ", cad[i]);
    }
    printf("\n");
}

int main() {

    char cad[] = "Una prueba";

    printf("Cadena: %s\n", cad);
    printf("Bytes : ");
    disp_bytes(cad);
}
```



```
santiago:EjemplosString$ ./ej_1
Cadena: Una prueba
Bytes : 85 110 97 32 112 114 117 101 98 97 0
santiago:EjemplosString$
```

Es importante que te fijas en dos detalles:

- La función `strlen()` devuelve el número de caracteres de la cadena, sin contar el carácter `'\0'` final.
- En la función `disp_bytes()` se ha querido que se imprima dicho carácter `'\0'` final y por eso se ha usado el `<=` como límite superior del índice del bucle: `i<=strlen(cad)`.

**`strcpy()` `strncpy()`**: Funciones para copiar cadenas

Como se ha comentado, no se puede utilizar el operador de asignación, el signo `=`, para copiar una cadena en otra. Para copiar cadenas se utilizan las funciones `strcpy()` y `strncpy()`:

- **`strcpy(cad_destino, cad_origen)`**: copia el contenido de la cadena `cad_origen` en la cadena `cad_destino`.
- **`strncpy(cad_destino, cad_origen, n)`**: sustituye los `n` primeros caracteres de la cadena `cad_destino` con los `n` primeros caracteres de la cadena `cad_origen`.

La cadena origen de la copia puede ser una variable que contiene una cadena de caracteres o un literal entrecomillado. La cadena destino tiene que haberse creado previamente y disponer de espacio suficiente para realizar la copia. En el siguiente ejemplo, se utilizan las dos funciones utilizando las dos posibilidades para la cadena origen.

### Ejemplo 4 Funciones para copia de cadenas

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

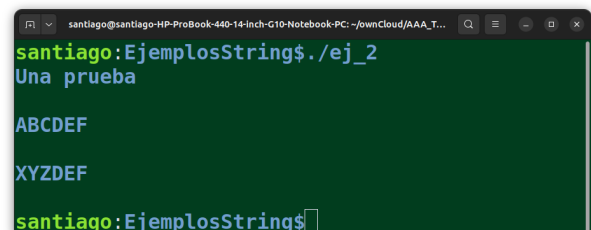
int main() {

    char cad_origen[] = "Una prueba";
    char cad_destino[40]="Q";

    strcpy(cad_destino, cad_origen);
    printf("%s\n\n", cad_destino);

    strcpy(cad_destino, "ABCDEF");
    printf("%s\n\n", cad_destino);

    strncpy(cad_destino, "XYZ", 3);
    printf("%s\n\n", cad_destino);
}
```



```
santiago:EjemplosString$ ./ej_2
Una prueba

ABCDEF

XYZDEF
santiago:EjemplosString$
```

**strcmp() strncmp():** Funciones para comparar cadenas

- **strcmp(cad\_1, cad\_2):** compara las cadenas *cad\_1* y *cad\_2*. Devuelve 0 si son iguales y un valor distinto de 0 si no lo son. Un valor devuelto menor que 0 indica que, si se ordenaran alfabéticamente, la primera cadena iría primero que la segunda. Si el resultado es positivo, sucede al contrario.
- **strncmp(cad\_1, cad\_2, n):** funciona como la función *strcmp()*, pero solo compara los *n* primeros caracteres.

Algunos ejemplos serían:

```
strcmp(``ABCDE'', ``XY'')           ⇒  -23
strcmp(``ABCDE'', ``ABCDE'')        ⇒   0
strcmp(``zapato'', ``Zapato'')       ⇒   1 (Las mayúsculas van antes)
strncmp(``ABC'', ``ABCDEF'', 3)      ⇒   0
```



### `strcat()`: Concatenar cadenas

- **`strcat(cad_1, cad_2)`**: añade al final de la cadena `cad_1` el contenido de la cadena `cad_2`. Es necesario que la cadena `cad_1` tenga suficiente espacio, si no se produce un error.
- **`strncat(cad_1, cad_2, n)`**: solo concatena los primeros `n` caracteres de `cad_2`.

#### Ejemplo 5 Concatenar cadenas

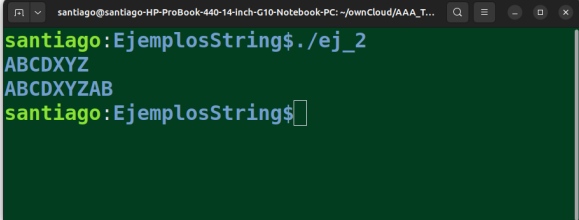
```
#include <stdio.h>
#include <string.h>

int main() {

    char cad_1[10] = "ABCD";
    char cad_2[] = "XYZ";

    strcat(cad_1, cad_2);
    printf("%s\n", cad_1);

    strncat(cad_1, cad_1, 2);
    printf("%s\n", cad_1);
}
```



```
santiago:EjemplosString$ ./ej_2
ABCDXYZ
ABCDXYZAB
santiago:EjemplosString$
```

Observa cómo se ha definido la cadena `cad_1`: se ha creado de tamaño 10, aunque luego solo se han guardado 4 caracteres. Esto se ha hecho así para que haya sitio en la cadena para añadir más caracteres.

Es importante entender que las cadenas se crean como un array de determinado tamaño. Si se declara un array de `n` caracteres, se podrán guardar hasta `n-1` caracteres. El último se reserva para el carácter `'\0'`.

En el siguiente ejemplo, se usa una función llamada `disp_bufer()`, similar a la función `disp_bytes()` que se utilizó en el Ejemplo 3, pero que permite mostrar todo el espacio de memoria de la variable, no solo hasta el `'\0'` que marca el final efectivo de los caracteres de la cadena.

#### Ejemplo 6 Memoria ocupada por las cadenas de caracteres

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

#define DIM 10

void disp_bufer(char cad[]) {
    for(int i=0; i<DIM; i++) {
        printf("%d ", cad[i]);
    }
    printf("\n\n");
}

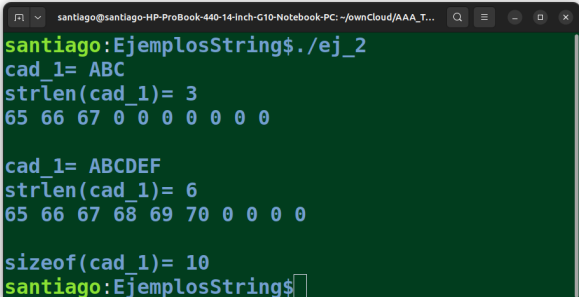
int main() {

    char cad_1[DIM] = {0};

    strcpy(cad_1, "ABC");
    printf("cad_1= %s\n", cad_1);
    printf("strlen(cad_1)= %d\n", strlen(cad_1));
    disp_bufer(cad_1);

    strcat(cad_1, "DEF");
    printf("cad_1= %s\n", cad_1);
    printf("strlen(cad_1)= %d\n", strlen(cad_1));
    disp_bufer(cad_1);

    printf("sizeof(cad_1)= %d\n", sizeof(cad_1));
}
```



```
santiago:EjemplosString$ ./ej_2
cad_1= ABC
strlen(cad_1)= 3
65 66 67 0 0 0 0 0 0

cad_1= ABCDEF
strlen(cad_1)= 6
65 66 67 68 69 70 0 0 0

sizeof(cad_1)= 10
santiago:EjemplosString$
```

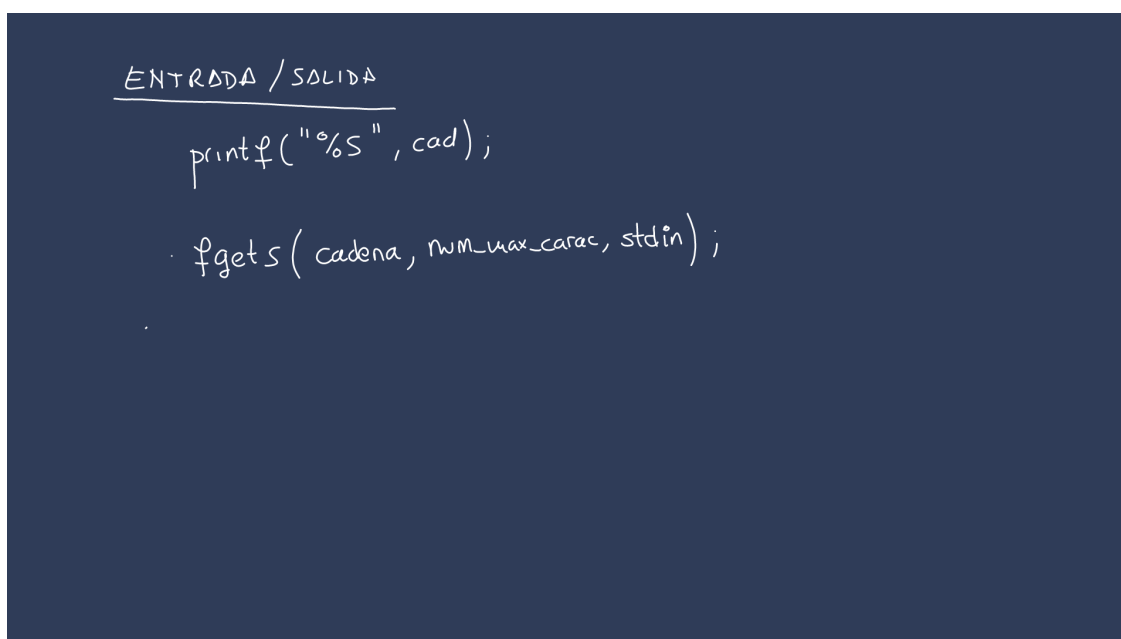
Observa que la variable en memoria siempre tiene los 10 bytes. La función *strlen()* siempre devuelve los caracteres que tenga guardados, no el tamaño de la variable en memoria. Esta función cuenta hasta el primer carácter `'\0'` que haya en la cadena. Si se quiere saber cuánto espacio hay guardado en la memoria hay que usar el operador *sizeof()*.

Otro detalle del ejemplo anterior es que la cadena de caracteres se ha inicializado con la siguiente expresión:

```
char cad_1[10] = {0};
```

De esta forma, el array se inicializa con todos sus elementos valiendo 0. No se puede usar otro valor, solo el 0 o las llaves vacías, que es equivalente.

## 5 ENTRADA DE CADENAS UTILIZANDO *FGETS()*



Para leer cadenas de caracteres tecleadas por el usuario en el terminal se puede utilizar la función *fgets()*:

```
fgets(cadena, dim, stdin);
```

En la expresión anterior:

- *cadena*: es la cadena de texto donde se guardará la cadena tecleada por el usuario. Se debe haber creado antes de llamar a la instrucción y tener un tamaño al menos como el indicado en el parámetro *dim*.
- *dim*: es el número máximo de caracteres que se guardarán en la cadena, incluyendo el carácter `'\0'` final.
- *stdin*: es la cláusula que hay que indicar para que la lectura se haga desde el terminal. Más adelante en el curso veremos cómo leer de un fichero, en cuyo caso este parámetro tendrá un puntero a un fichero.

Hay que tener en cuenta que, cuando el usuario teclea una cadena en el terminal, el último carácter que teclea es la tecla *INTRO*, que es un carácter `'\n'`. Este carácter, que se suele llamar `'\n'` residual, será el último carácter de la cadena leída, antes del carácter `'\0'`.

El Ejemplo 7 utiliza la función *disp\_bufet()* que se utilizó en el Ejemplo 6 para mostrar la memoria de la cadena tecleada por el usuario. Observa el código 10 al final de la cadena: ese es el carácter `'\n'`.

En el Ejemplo 7 también se ha utilizado una función llamada *limpia()* que sustituye el carácter '\n' final por un carácter '\0'. Observa cómo se ha calculado el índice del último elemento de la cadena leída.

Es importante recordar aquí que los caracteres que teclees en el terminal deben ser caracteres ASCII, no puedes teclear letras acentuadas o la letra ñ u otros caracteres especiales.

### Ejemplo 7 Uso de *fgets()* para lectura de cadenas desde el terminal

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

#define DIM 10

void disp_bufer(char cad[]) {
    for(int i=0; i<DIM; i++) {
        printf("%d ", cad[i]);
    }
    printf("\n\n");
}

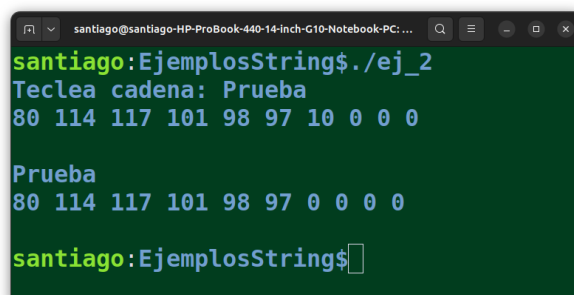
void limpia(char cad[]) {
    int ultimo = strlen(cad)-1;
    if(cad[ultimo] == '\n') {
        cad[ultimo]='\0';
    }
}

int main() {
    char cad[DIM] = {};

    printf("Teclea cadena: ");
    fgets(cad, DIM, stdin);

    disp_bufer(cad);

    limpia(cad);
    printf("%s\n", cad);
    disp_bufer(cad);
}
```



Limpiar \n residual

```
void limpia_cadena(char * cad) {
    int indice_ultimo_car = strlen(cad) - 1;
    if (cad[indice_ultimo_car] == '\n') {
        cad[indice_ultimo_car] = '\0';
    }
}
```

## CADENAS DE CARACTERES (Strings)

cadena  $\equiv$  array de char + '\0'

char cad[] = { 'p', 'r', 'u', 'e', 'b', 'a', '\0' };

char cad[] = "Prueba";  $\rightarrow$  se añade el '\0' automáticamente

Tengo que poner el '\0'

- No puedo usar op. asignación o de comparación

char cad[] = "Presa";

~~char cad2 = cad;~~ // ~~cad2 == cad1~~

- Puedo acceder a elem. indiv. con índice (alfabeto inglés)

cad[2]  $\rightarrow$  e

## LIBRERÍA string.h (alfabeto inglés)

- strlen(cad)  $\rightarrow$  n° bytes  $\equiv$  n° carac.

- strcpy(dest, src);  $\rightarrow$  copia src en dest

- strncpy(dest, src)  $\rightarrow$  copia n primeros carac. de src en dest

- strcmp(cad1, cad2)  $\rightarrow$  devuelve 0 si son iguales

- strncmp(cad1, cad2, n)  $\rightarrow$  devuelve 0 si n primeros iguales

- strcat(cad1, cad2)  $\rightarrow$  añade cad2 a cad1

Mostrar cadena como bytes → equiv: (char cad[])

```
void disp_bytes(char* cad) {  
    for(int i=0; i<strlen(cad); i++) {  
        printf("%d_", cad[i]);  
    }  
    printf("\n");  
}
```

NOTA

char cad[]

cad ≡ &cad[0]

char cad[10]; → 9 caracteres + '\0'

fgets(cad, 10, stdin); → Lee 9 como máximo

<del>x</del> <sub>p</sub>	<del>x</del> <sub>2</sub>	<del>x</del> <sub>p</sub>	<del>x</del> <sub>0</sub>	b	a				
---------------------------	---------------------------	---------------------------	---------------------------	---	---	--	--	--	--

```
#define MAX_TXT 200  
  
typedef char tTexto [MAX_TXT];  
  
tTexto nombre = "Pepito";  
  
typedef struct {  
    tTexto nombre;  
    tTexto ape;  
} tAutor;  
  
tAutor auth1;  
strcpy(auth1.nombre, "Pepe");
```