Sistemas Informáticos Industriales

Apuntes de

Tratamiento de cadenas de caracteres en C

Grado en Electrónica y Automática
Departamento de Informática de Sistemas y Computadores
Escuela Técnica Superior de Ingeniería del Diseño

tratamiento_cadenas_en_C_r11.odt

Índice de contenido

5 TRATAMIENTO DE CADENAS DE CARACTERES EN C

5.1 INTRODUCCIÓN

La representación gráfica de los símbolos alfanuméricos que empleamos en la escritura (números, letras, signos de puntuación, etc.) requieren que el computador los haga equivalentes internamente a un valor numérico digital.

Muchos dispositivos industriales emplean este tipo de codificaciones para representar e intercambiar información, por lo que es importante conocer las técnicas que permiten manipular este tipo de información.

En esta unidad se pretende describir los métodos para representar y manipular información textual en C. Estas técnicas serán imprescindibles para, posteriormente, desarrollar aplicaciones que intercambien información entre un computador y un dispositivo industrial.

Esta unidad está pensada para trabajarla de forma lineal sin necesidad de acudir a otras fuentes ni materiales adicionales.

En puntos concretos se intercalan actividades que permitirán practicar los conocimientos adquiridos.

En caso de dificultad en la resolución de actividades sí se recomienda acceder a fuentes externas, por ejemplo, a la bibliografía recomendada.

5.2 OBJETIVOS

Conocer el código ASCII de representación de caracteres.

Conocer la representación interna de cadenas de caracteres en C.

Aprender a manipular directamente cadenas de caracteres.

Aprender a utilizar las funciones de tratamiento de cadenas proporcionadas por las bibliotecas estándar de C.

Aprender a convertir información textual entre cadenas C estándar y otras representaciones.

5.3 EL CÓDIGO ASCII

Una de las representaciones de caracteres más ampliamente aceptadas es el código de representación ASCII (American Standard Code for Information Interchange) que originalmente era un código de 7 bits (2^7=128 códigos diferentes) al que se le solía añadir un bit más para la detección de errores de transmisión. En la actualidad, el octavo bit se emplea para conseguir 128 códigos más (2^8=256 códigos diferentes).

Los códigos ASCII se pueden agrupar en 3 clases:

- Caracteres de control (del 0 al 31): Tienen funciones especiales y se usan para el control de archivos de texto, control de comunicación, indicación de final de línea, de texto, tabuladores, ...
- Caracteres normales (del 32 al 127): Representan los números del 0 al 9, las letras mayúsculas y minúsculas, y otro símbolos.
- Caracteres extendidos (del 128 al 255): Son caracteres gráficos no normalizados que suelen incorporar símbolos específicos del idioma de cada país particular.

Las Tabla 1-1 y Tabla 1-2 resumen el significado ASCII asociado a cada posible código de 8 bits. Recuérdese que del código 128 al 255 no son estándar, y pueden variar de un país a otro en función de la configuración específica de cada computador por lo que no se han representado.

Esta codificación es importante para nosotros porque en lenguaje C los caracteres suelen representarse mediante este código de 8 bits (1 byte u octeto).

Para la parte internacional de los códigos ASCII hay bastantes combinaciones e intentos de normalización. Uno de las extensiones más aceptadas para los países europeos del oeste es ISO 8859-1, también conocido como Latín 1. Esta codificación es adecuada para representar el español, alemán, francés, etc.

Se debe tener siempre presente que este código impone serias limitaciones en la internacionalización de aplicaciones, por lo que se recomienda emplear otras codificaciones para hacer frente a este problema.

Dec	Hex	Sym	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	0	NUL	32	20		64	40	@	96	60	`
1	1	SOH	33	21	!	65	41	A	97	61	a
2	2	STX	34	22	"	66	42	B	98	62	b
3	3	ETX	35	23	#	67	43	C	99	63	c
4	4	EOT	36	24	$	68	44	D	100	64	d
5	5	ENQ	37	25	%	69	45	E	101	65	e
6	6	ACK	38	26	&	70	46	F	102	66	f
7	7	BEL	39	27	'	71	47	G	103	67	g
8	8	BS	40	28	(	72	48	H	104	68	h
9	9	TAB	41	29	)	73	49	I	105	69	i
10	A	LF	42	2A	*	74	4A	J	106	6A	j
11	B	VT	43	2B	+	75	4B	K	107	6B	k
12	C	FF	44	2C	,	76	4C	L	108	6C	l
13	D	CR	45	2D	-	77	4D	M	109	6D	m
14	E	SO	46	2E	.	78	4E	N	110	6E	n
15	F	SI	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	48	30	0	80	50	P	112	70	p
17	11	DC1	49	31	1	81	51	Q	113	71	q
18	12	DC2	50	32	2	82	52	R	114	72	r
19	13	DC3	51	33	3	83	53	S	115	73	s
20	14	DC4	52	34	4	84	54	T	116	74	t
21	15	NAK	53	35	5	85	55	U	117	75	u
22	16	SYN	54	36	6	86	56	V	118	76	v
23	17	ETB	55	37	7	87	57	W	119	77	w
24	18	CAN	56	38	8	88	58	X	120	78	x
25	19	EM	57	39	9	89	59	Y	121	79	y
26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	60	3C	<	92	5C	\	124	7C	|
29	1D	GS	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	62	3E	>	94	5E	^	126	7E	~
31	1F	US	63	3F	?	95	5F	_	127	7F	□

Tabla 1-1. Códigos ASCII del 0 al 127.

Name	Char	Dec	Hex	Description
NUL	^@	00	00	null
SOH	^A	01	01	start of heading
STX	^B	02	02	start of text
ETX	^C	03	03	end of text
EOT	^D	04	04	end of transmission
ENQ	^E	05	05	enquiry
ACK	^F	06	06	acknowledge
BEL	^G	07	07	bell
BS	^H	08	08	backspace
HT	^I	09	09	horizontal tab
LF, NL	^J	10	0A	line feed, new line
VT	^K	11	0B	vertical tab
FF, NP	^L	12	0C	form feed, new page
CR	^M	13	0D	carriage return
SO	^N	14	0E	shift out
SI	^O	15	0F	shift in
DLE	^P	16	10	data link escape
DC1	^Q	17	11	device control 1
DC2	^R	18	12	device control 2
DC3	^S	19	13	device control 3
DC4	^T	20	14	device control 4
NAK	^U	21	15	negative acknowledge
SYN	^V	22	16	synchronous idle
ETB	^W	23	17	end of trans. block
CAN	^X	24	18	cancel
EM	^Y	25	19	end of medium
SUB	^Z	26	1A	substitute
ESC	^[	27	1B	escape
FS	^\	28	1C	file separator
GS	^]	29	1D	group separator
RS	^^	30	1E	record separator
US	^_	31	1F	unit separator

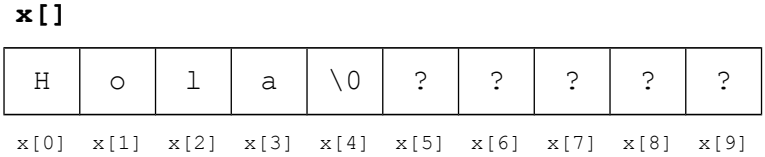
Tabla 1-2. Significado asociado a los caracteres de control ASCII.

5.4 REPRESENTACIÓN DE CADENAS DE CARACTERES EN C

Una cadena de caracteres es una secuencia de símbolos alfanuméricos. Para su representación, el lenguaje C emplea un caso especial de matriz de caracteres en el que el final de la cadena se marca con el código ASCII 0 (en C `\x0` ó `\0`).

Por ejemplo, si se coloca la cadena “Hola” en un vector de 10 caracteres se tendrá:

```
char x[10];
```



Es necesario el carácter de control `\0` para marcar el final de la cadena, por lo que una matriz de N caracteres podrá contener como máximo N-1 caracteres.

ATENCIÓN

Para representar escrita una cadena en un programa C, ésta se escribe entre comillas dobles ("). Para representar un carácter individual se utilizan las comillas simples ('), así que:

`'A'` representa el carácter ASCII 41h

`"A"` representa la cadena formada por `'A'` y `\0`

La cadena almacenada en el vector se puede manipular accediendo a los componentes del vector. Por ejemplo, si se hace la siguiente asignación,

```
x[3]='o';
```

el vector `x[]` contendrá la cadena `"Holo"`.

Si ahora se hace la siguiente asignación,

```
x[3]='\0';
```

el vector `x[]` contendrá la cadena `"Hol"`

Actividad

Realizar un programa que contabilice en una variable entera el número de `'a'`s que hay en un vector de caracteres cuya definición es `char texto[1000]`. Téngase en cuenta que el vector no tiene porque estar lleno del todo.

Actividad

Modificar el programa anterior para que sustituya las `'a'` por `'o'` en la variable que contiene la cadena.

5.5SECUENCIAS DE ESCAPE EN C

Algunos de los caracteres de la tabla de códigos ASCII no representan símbolos alfanuméricos o corresponden a caracteres reservados para la sintaxis del lenguaje C, por lo que no pueden escribirse directamente en un programa en C. Para representar algunos de estos caracteres en C se emplea la barra invertida (\) seguida de un carácter según la siguiente tabla:

Cadena	Significado
\n	nueva línea (newline)
\t	tabulador (tab)
\b	espacio atrás (backspace)
\r	retorno de carro (carriage return)
\f	avance de página (form feed)
\\	barra invertida (backslash)
\’	comilla simple (single quote)
\"	comillas dobles (double quote)
\0	fin de cadena C, ascii 0
\xHH	caracter con ASCII HH en hexadecimal

Por ejemplo, para lograr que la función `printf()` muestre comillas dobles por pantalla se podría hacer:

```
printf("Juan es \"melón\"");
```

La notación `\xHH`, donde `HH` representa un número hexadecimal entre 00h y Ffh permite conseguir cualquiera de los caracteres ASCII en C. Por ejemplo, si se escribe `'\x41'` se está representando el código ASCII 41h. Escrito en C sería,

```
char letra;
letra='\x41';
```

que equivale a asignar la letra 'A'.

Otras opciones para hacer lo mismo serían,

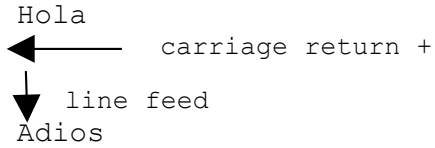
```
letra='A';
letra=0x41;
letra=65;
```

5.6 INCOMPATIBILIDADES ENTRE PLATAFORMAS

La representación de información ASCII en cada computador/sistema operativo puede dar un significado diferente a los códigos de control, por lo que pueden presentarse incompatibilidades al transferir información textual entre distintas máquinas, máquinas que no tienen porque ser computadores.

Por ejemplo, existe un problema de compatibilidad importante entre MS-DOS/Windows y Unix en la representación de los retornos de carro.

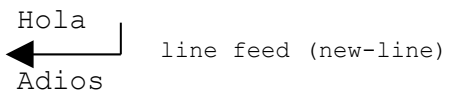
Como ejemplo, supóngase que un programa en C muestra por pantalla el texto "Hola" en una línea y el texto "Adios" en la siguiente línea. En MS-DOS/Windows para conseguir esto se utilizan los siguientes caracteres de control para el movimiento de la posición de escritura al lugar adecuado:



por tanto, la cadena C que permite conseguir esto es:

H	o	l	a	\x0D	\x0A	A	d	i	o	s	\0

En Unix esto se consigue usando únicamente el código de control \x0D, y que coincide con uno de los usados en Microsoft Windows, pero que tiene ahora un significado diferente:



por tanto, la cadena C que permite conseguir esto es:

H	o	l	a	\x0D	A	d	i	o	s	\0

Así, por ejemplo, si transferimos el texto anterior del sistema operativo Linux (Unix) al sistema operativo Windows y lo mostramos por pantalla saldrá:

```
Hola
     ↓
     Adios
```

Tanto en Unix como en DOS/Windows se puede haber utilizado la siguiente construcción para generar la cadena,

```
printf("Hola\nAdios");
```

pero cada compilador genera realmente la cadena con una de las dos formas anteriores en función de la plataforma destino.

Para tener control sobre la generación de estas secuencias se propone utilizar siempre la secuencia de escape \xNN al comunicarse con dispositivos externos. Por ejemplo,

```
printf("Hola\x0D\x0AAdios");
```

Actividad

Realiza una función a la que se le pase como parámetro un puntero a una cadena de MS-DOS y elimine de dicha cadena los códigos `\x0A` para que la cadena sea compatible con Unix.

Ejemplo de uso,

```
char texto[]="En\nCastilla"; //suponemos estamos en MS-DOS
...
cadenaDosAUnix(texto);  //funcion a desarrollar
...
```

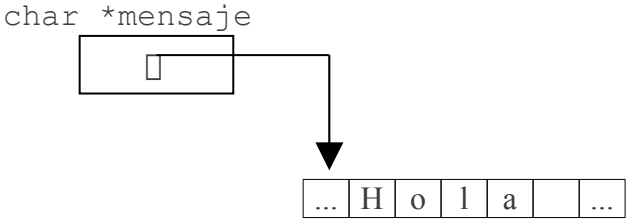
5.7 FUNCIONES Y MACROS PARA TRATAMIENTO DE CADENAS

Antes de entrar en materia:

ATENCIÓN

En general, cuando se escribe una cadena en un programa C, el compilador crea un vector con el formato antes comentado y un puntero a la dirección donde comienza la cadena en memoria. Por ejemplo, la siguiente declaración es correcta:

```
char *mensaje = "Hola, don pepito";
```



Esto es aplicable a cualquier cadena, por ejemplo si escribimos,

```
printf("Hola");
```

la cadena "Hola" se genera en una zona de memoria distinta a donde se ha generado el código para printf(). Y a printf() se le pasa un puntero con la dirección de memoria dónde se ha creado la cadena.

Las bibliotecas estándar de C disponen de una serie de funciones y macros que facilitan la realización de las operaciones más comunes con cadenas de caracteres.

Para utilizar dichas funciones hay que incluir el fichero de cabecera adecuado a cada función en el módulo C que las emplee.

Las funciones básicas para el trabajo a desarrollar aquí son:

Nombre	Cabecera	Propósito
int strlen(char *s);	string.h	devuelve la longitud de una cadena de caracteres
int strcmp(char *cd1, char *cd2);	"	compara dos cadenas
char *strcpy(char *dest, char *or);	"	copia una cadena a un destino
char *strncpy(char * dest, char *or, unsigned n);	"	copia n caracteres de una cadena a un destino
char *strcat(char *dest, char *or);	"	concatena dos cadenas
void *memcpy(void *dest, void *or, unsigned n);	"	copia n octetos en un destino
char *strchr(const char *s, int c);	"	explora la cadena hasta localizar la primera ocurrencia del caracter
char *strlwr(char *s);	"	pasar cadena a minúsculas
char *strpbrk(const char *s1, const char *s2);	"	explora una cadena en busca de la ocurrencia de alguno de los caracteres de la cadena s2

char *strrev(char *s);	"	dar la vuelta a una cadena
char *strset(char *s, int ch);	"	rellenar la cadena con un caracter hasta encontrar el fin
size_t strspn(const char *s1, const char *s2);	"	explora la cadena hasta encontrar un segmento que sea un subconjunto de los caracteres de s2
char *strstr(const char *s1, const char *s2);	"	buscar en la cadena la primera ocurrencia de la cadena s2
char *strupr(char *s);	"	pasar una cadena a mayúsculas

Por tanto, si se quiere:

- Saber la longitud de una cadena: `strlen()`

  Por ejemplo,

  ```
  printf("Longitud = %d\n", strlen ("otorrino");
  ```

  Como inciso, observar que, como parámetro, `strlen()` admite un puntero. Como ya se ha indicado, el compilador ha creado la cadena en una zona de memoria particular y le pasa a la función un puntero a esa cadena.

- Copiar cadenas: `strcpy()`, `strncpy()`

  Por ejemplo, si se quiere hacer una copia de una cadena al vector `buffer`,

  ```
  char buffer[1000];
  strcpy(buffer, "Hola");
  ```

- Concatenar dos cadenas: `strcat()`

Por ejemplo, si se quiere añadir una nueva cadena al final de la cadena contenida en el vector `buffer`,

```
char buffer[1000];
strcpy(buffer, "Hola");
strcat(buffer, " y Adios");
```

en `buffer` habrá `"Hola y Adios"`

- Saber si dos cadenas son iguales: `strcmp()`, pudiéndose saber también si una cadena está, alfabéticamente hablando, por delante o por detrás de otra. El problema es que esta comparación se hace utilizando el alfabeto inglés, por lo que el uso de símbolos como ñ, ç, o acentos da lugar a errores en la clasificación.

  Por ejemplo,

  ```
  strcmp(cad1, cad2)
  ```

  devuelve:

  `< 0` si `cad1` está antes en el alfabeto que `cad2`
  `==0` si `cad1` y `cad2` son iguales
  `> 0` si `cad1` está después en el alfabeto que `cad2`

Hay muchas más funciones en la biblioteca estándar de C y una buena forma de conocerlas es utilizar la ayuda interactiva de los entornos de desarrollo utilizados.

Actividad

Sea

```
char *c1 = "Mortadelo";
char *c2 = "Filemón";
char *c3 = "Bacterio";
char buffer[1000];
```

Consigue meter en el vector `buffer`:

```
"Mortadelo, Filemón y el Bacterio"
```

y mete en una variable entera cuántos caracteres tiene la cadena.

Actividad

Eliminar los 11 primeros caracteres del vector `buffer` anterior para que quede la cadena `"Filemón y el Bacterio"`. Dos posibles estrategias para hacerlo son:

1. Acceder directamente a los elementos del vector para ir copiando los elementos 11 posiciones hacia atrás a partir del elemento 11 y hasta llegar al `\0` inclusive.
2. Usar la función `strcpy()` para que copie un fragmento de `buffer` sobre si mismo, para ello pasarle los punteros adecuados de manera que uno apunte al principio de la cadena y otro a la posición a partir de donde se quiere copiar.

Si se desea analizar los caracteres que componen una cadena se dispone de una serie de macros (similares a funciones) a las que se accede incluyendo la cabecera `ctype.h`.

Estas macros devuelven *verdadero* (un valor distinto de 0) si se cumple que el carácter pasado como argumento cumple determinadas condiciones. Algunas de estas macros son:

Macro	Cabecera	Condición para verdadero
isalpha(c)	ctype.h	A-Z, a-z
isupper(c)	ctype.h	A-Z
islower(c)	ctype.h	a-z
isdigit(c)	ctype.h	0-9
isxdigit(c)	ctype.h	0-9, A-F, a-f
isspace(c)	ctype.h	espacio en blanco
isalnum(c)	ctype.h	0-9, A-Z, a-z

isprint(c)	ctype.h	carácter representable

Por simplicidad, estas macros se pueden considerar como funciones con el prototipo `int isalpha(char c);`.

Por ejemplo,

```
if (isalnum('1')) {
   printf("Es alfanumérico"); //esta es la que saldrá
} else {
   printf("No es alfanumérico");
}
```

Recuérdese que continúan las restricciones debidas a que estas macros están pensadas para el inglés, por tanto, en los rangos alfabéticos, los caracteres como la ñ, ç, vocales acentuadas, ... no son contemplados.

Actividad

Analiza la cadena almacenada en `char buffer[1000]` obtenida de la actividad anterior, y almacena en variables enteras cuántas mayúsculas hay, cuántas minúsculas y cuántos espacios en blanco.

Recuerda que contendrá la cadena “Mortadelo, Filemón y el Bacterio”.

¿Presenta algún problema la cadena anterior?

Otro grupo importante de funciones para el tratamiento de cadenas son las especializadas en la conversión de cadena de caracteres numéricos a su correspondiente valor numérico. La siguiente tabla muestra algunas:

Nombre	Cabecera	Propósito
double atof(char *p)	stdlib.h	convierte una cadena que represente un número a un double
int atoi(char *p)	“	de cadena a entero
long atol(char *p)	“	de cadena a long
itoa(int i, char *p, int base)	“	de entero a cadena en bases de 2 a 36

Incluyendo el fichero de cabecera adecuado se puede pasar una cadena que contiene un texto correspondiente a un número y usar `atoi()`, `atol()` y `atof()`, para pasar a `int`, `long` y `double` respectivamente.

Por ejemplo,

```
char *cadena = "123.56";
double valor;
valor = atof(cadena);
```

Mediante los dos grupos de funciones que se explican a continuación se puede hacer prácticamente lo mismo con una mayor flexibilidad en la generación y el análisis de cadenas.

5.8 "LOCALIZACIÓN" PARA MI IDIOMA

Antes de entrar en el siguiente punto, se ha visto la problemática que supone el emplear idiomas distintos del inglés.

Este problema se agrava cuando se introducen temas como la separación de decimales, el formato de hora, etc.

Los modernos computadores pueden gestionar esto sin problemas, pero como tendremos que convivir con distintos dispositivos, es bueno saber controlar estos aspectos.

Para controlar la llamada "localización" se puede emplear la función setlocale() disponible a través de la cabecera locale.h.

Por ejemplo, para que los números generados con printf() utilicen la coma española, haríamos:

```
setlocale(LC_NUMERIC, "es_ES");
```

Para que todo se comporte como el lenguaje C de toda la vida:

```
setlocale(LC_ALL, "C");
```

5.9 FORMATEADO DE CADENAS CON SPRINTF

La función `printf()` sirve para mostrar información por pantalla debidamente formateada. Esta función pertenece a una familia de funciones (`printf()`, `sprintf()`, `fprintf()`, ...), cuya diferencia radica en el lugar de destino de la cadena formateada resultante.

En este apartado se comentará el uso de la función `sprintf`, que permite crear cadenas formateadas en memoria.

Su prototipo es:

Nombre	Cabecera	Propósito
int sprintf(char *cad, char *fto, ptr1, ptr2, ...)	stdio.h	crea una cadena en memoria según formato

Un ejemplo de uso puede ser:

```
char buffer[1000];
int i = 345;
sprintf(buffer, "La variable i vale %d \n", i);
```

En buffer se habrá almacenado la cadena `"La variable i vale 345 \n"`.

Para explotar adecuadamente esta función, vale la pena revisar las secuencias de caracteres que permiten controlar el formato de salida de los datos. La siguiente tabla contiene las secuencias más utilizadas para formatear tipos de datos de maneras diferentes:

Secuencia	Tipo de dato	Salida formateada

%d	int	decimal
%o	int	octal
%x	int	hexadecimal
%u	unsigned int	entero sin signo
%c	char	caracter
%s	char *	cadena de caractes
%e	float	real científico m.nnnExx
%f	float	formato mm.nn

Así, por ejemplo, una variable entera tipo `int` se puede formatear de manera que se muestre su contenido en formato decimal, octal o hexadecimal en función de si se usa la secuencia de formateo `%d`, `%o` ó `%x` respectivamente.

Entre el % y el carácter de elección de tipo de formato se puede colocar una secuencia de caracteres que modifica la forma de presentar la información. Algunos de estos caracteres son:

- **La letra `l`:** para indicar que se escribe un `long int` o un `double`. Destacar que cuando se emplean tipos de datos de mayor precisión, es necesario informar de este hecho a la función de formateo, si no se hace, las consecuencias pueden ser desastrosas. Por ejemplo:

```
float lambda = 12.34567;
double delta = 0.5;
printf("Lamba vale %f y delta %lf", lambda, delta);
```

- **Una cifra**: para especificar el número mínimo de caracteres a ocupar. En función del tipo de dato a formatear, se asigna un número de caracteres (espacios) a ocupar por la cadena de salida. Para entenderlo mejor, se propone el siguiente fragmento de código.

```
int i = 37;
printf("@%d@%4d@", i, i);
```

  La salida del formato será, `"@37@  37@"`, es decir, con el `%4d` conseguimos que se reserve un espacio de 4 caracteres para meter el número.

- **Un -**: para imponer alineación a la izquierda. Por defecto, los números se alinean a la derecha, y con este modificador de formato, podemos conseguir que la alineación sea a la izquierda.

  Si se quisiese justificar a la izquierda el número anterior, usaríamos `%-4d` y conseguiríamos `"@37  @"`.

- **Añadir un 0**: antes del modificador de espaciado conseguimos que los espacios en blanco se rellenen con ceros. Por ejemplo:

```
int i = 37;
printf("@%04d@", i);
```

La salida del formato será, `"@0037@"`.

- **Con un punto (`.`)**: se consigue separar dos especificaciones. Un caso habitual es la representación de números en coma flotante, donde se desea una cantidad específica de decimales. Como explicación valga el siguiente ejemplo:

```
float lambda =12.34567;
printf("Lamba vale %7.2f", lambda);
```

La salida del formato será, `"Lambda vale:  12.35"`, es decir, se usan 7 espacios para meter el número y se mostrarán 2 decimales.

ATENCIÓN

Las cadenas de formateo utilizadas en `printf()` y similares no pueden comprobar que el tipo de dato que se les pasa es el correcto. Por ejemplo, si indicamos que queremos mostrar un cadena (`%s`) y como dato le pasamos un entero (`int`), la función de formateo actuará intentando generar la cadena a partir de algo que no lo es. Esto suele provocar la generación de cadenas de salida extrañas, erróneas y fallos intermitentes de las aplicaciones.

IMPORTANTE

Por la razón anterior, las funciones de la familia printf() y scanf() están prohibidas en sistemas empotrados normalizados para automoción, aviónica, etc. Interesados que vean MISRA-C.

Actividad

Conseguir mediante `sprintf()` la cadena :

`"PI vale 3.14, y la palabra tres es el número 0003"`

utilizando obligatoriamente las siguientes variables. Deposita la cadena en `char formateado[100]`.

```
double pi = 3.141592;
int i = 3;
char *t = "tres";
```

Actividad

Escribe una función que cree un volcado en formato "raro" (Intel-HEX) de un vector de N datos enteros con valores entre 0 y 255. La función tendrá el siguiente prototipo:

```
void dumphex(char *destino, int *datos, int num_datos);
```

El siguiente fragmento de programa muestra un ejemplo de uso:

```
int datos[100];
char buffer[1000];

dumphex(buffer, datos, 100);
```

La cadena a generar tendrá el siguiente formato:

```
:XXAABBCCDDEE...NN$
```

- XX el número de bytes en hexadecimal en 2 cifras rellenados con 0s a la izquierda
- AA, BB, ..., NN dato entero en formato hexadecimal de 2 cifras

5.10 EXPLORACIÓN DE CADENAS CON SSCANF

La función `scanf` permite extraer datos de cadenas introducidas por teclado. Como en el caso anterior, dicha función pertenece a una familia de funciones (`scanf`, `sscanf`, `fscanf`, ...) cuya diferencia radica en el origen de la cadena a explorar.

En nuestro caso nos será de utilidad la función `sscanf()`, que permite explorar cadenas de caracteres en memoria para extraer información.

Su prototipo es:

Nombre	Cabecera	Propósito
int sscanf(char *cad, char *fto, ptr1, ...)	stdio.h	analiza una cadena en memoria según formato

Para `sscanf()` y familia se utilizan las mismas secuencias de formato que `printf()` para indicar qué tipo de dato se pretende extraer y en qué tipo de variable se va a guardar.

Por ejemplo, el siguiente fragmento de programa hace que se explore una cadena para extraer el valor de una secuencia de caracteres que representan un número hexadecimal.

```
char *cadena = "a3fc";
int valor;

sscanf(cadena,"%x", &valor);
printf("La cadena interpretada como número hexadecimal vale en decimal %d\n",
valor);
```

Para la función `sscanf` y familia deben considerarse los siguientes aspectos:

- Se ignoran los espacios en blanco al comienzo del dato, excepto para `%c`.
- Se pueden especificar varios datos en una misma entrada si se separan, al menos, por un espacio en blanco.
- La conversión se detiene en el primer carácter que no corresponda al tipo de dato deseado.
- Un `*` entre `%` y el formato, hace la conversión, pero no la asigna a ninguna variable, por lo que nos saltamos datos.

Ejemplo:

```
char *cad = "19 32";
int cajas, botellas;


sscanf(cad, "%d%d", &cajas, &botellas);
printf("Hay %d cajas y %d botellas\n", cajas, botellas);
```

Otro ejemplo:

```
char *cad = "El rape está a 19.67 euros";
double rape;

sscanf(cad, "%*15c%lf", &rape);
printf("Señooora, rape a %lf euros, oiga!\n", rape);
```

Sabiendo que usamos punteros, también se podría hacer lo anterior de la siguiente manera:

```
sscanf(cad+15, "%lf", &rape);
```

Actividad

Explora la cadena `"a03451"` de manera que:

- El primer carácter vaya a una variable de tipo `char`.
- Del 2º carácter al 5º meter su valor equivalente en decimal en una variable `int`.
- El 6º carácter vaya a una variable tipo `int`.

5.11 CONVERSIÓN DE CADENAS ESTÁNDAR C A OTROS FORMATOS DE CADENA

5.11.1 FUNDAMENTOS

La forma de representar cadenas de C estándar está bastante limitado en aspectos como la internacionalización, etc.

Por otra parte, es muy posible que utilicemos entornos de desarrollo con características más avanzadas para el tratamiento de cadenas.

A pesar de la potencia que puedan aportar esos entornos, se pretende no salirse del C estándar para lograr la mayor independencia posible del entorno concreto.

Para conseguir una mayor flexibilidad y/o adaptarse a características específicas de un determinado entorno, se pretende aportar aquí una solución que permita combinar ambas posibilidades, explicando la manera de convertir entre tipos.

5.11.2 CONVERSIÓN QSTRING PARA QT

IMPORTANTE: En Qt5, la localización (locale) de las cadenas C es siempre UTF-8. ASCII ha volado. Como recomendación de futuro para lo que sea, se recomienda también codificar el código de cualquier programa usando UTF-8.

En Qt, el tipo de dato principal para representar cadenas de texto es `QString`.

La clase `QString` de Qt permite contener y manipular cadenas de texto en formato Unicode. Esto aporta grandes ventajas a la hora de internacionalizar (lengua) y localizar (formatos decimales, monedas, etc.) una aplicación.

Es, por tanto, aconsejable utilizar esta clase para todo aquello que sea información textual que se va a presentar/solicitar al usuario.

Y, siguiendo la filosofía de la asignatura, no es adecuado si se quiere garantizar independencia de Qt. No se recomienda utilizarlo en aquellas partes de la aplicación que sean susceptibles de migrar a otro entorno.

Para que una cadena C se convierta a `QString` no hace falta hacer nada, basta con asignarla. Por ejemplo:

```
char *cadena_c = "Hola Mundo";
QString cadena_qt;

cadena_qt = cadena_c;          // un ejemplo
cadena_qt = "Y también así";  // otro ejemplo
```

Convertirla a cadena C es un poco más complicado. Para no meterse en explicaciones que necesitan de otras explicaciones se hará de la siguiente manera.

```
char cadena_c[1000];    // hacer sitio para la cadena
QString cadena_qt;

cadena_qt = "Hola Qt";

// ejemplo para copiar el contenido de QString a cadena C
strcpy(cadena_c,cadena_qt.toAscii().data());
```

Básicamente, se llama al método `toLatin1()` para convertir la cadena a ASCII y a continuación se llama al método `data()` para que se genere una cadena C en memoria dinámica y se devuelva un puntero a esa cadena.

5.12 BIBLIOGRAFÍA

Este libro tiene un C muy clarito

- Programación estructurada en C.Antonakos, Mansfield. Prentice-Hall