CAPÍTULO 8
Las cadenas de caracteres

INTRODUCCIÓN

Las computadoras normalmente sugieren operaciones aritméticas ejecutadas sobre datos numéricos. Sin embargo, ese concepto no es estadísticamente cierto, sino que, al contrario, hoy día es cada vez más frecuente el uso de las computadoras para procesar problemas de tipo esencialmente alfanuméricos o de tipo texto. En el Capítulo 3 se estudió el concepto de tipo de datos carácter (**char**) y se definió un carácter como un símbolo del juego de caracteres de la computadora. Una **constante carácter** se definió como cualquier carácter encerrado entre separadores (apóstrofos o dobles comillas). Una secuencia finita de caracteres se denomina normalmente una **cadena** (*string*), y una **constante tipo cadena** consiste en una cadena encerrada entre apóstrofos o dobles comillas. El procesamiento de cadenas es el objetivo fundamental de este capítulo.

8.1. INTRODUCCIÓN

Las computadoras nacieron para resolver problemas numéricos en cálculos científicos y matemáticos. Sin embargo, el paso de los años ha cambiado las aplicaciones y hoy día las computadoras no sólo se utilizan en cálculos numéricos, sino también para procesar datos de caracteres. En aplicaciones de gestión, la generación y actualización de listas de dirección, inventarios, etc., la información alfabética es fundamental. La edición de textos, traductores de lenguajes y base de datos son otras aplicaciones donde las cadenas de caracteres tienen gran utilidad.

En este capítulo se tratará el concepto de cadena de caracteres y su procesamiento, utilizando para ello una notación algorítmica similar a la utilizada hasta ahora. Una *cadena de caracteres* es una secuencia de cero o más símbolos, que incluyen letras del alfabeto, dígitos y caracteres especiales.

8.2. EL JUEGO DE CARACTERES

Los lenguajes de programación utilizan *juegos de caracteres* "alfabeto" para comunicarse con las computadoras. Las primeras computadoras sólo utilizaban informaciones numéricas digitales mediante el código o alfabeto digital, y los primeros programas se escribieron en ese tipo de código, denominado *código máquina* —basado en dos dígitos, 0 y 1—, por ser inteligible directamente por la máquina (computadora). La enojosa tarea de programar en código máquina hizo que el alfabeto evolucionase y los lenguajes de programación comenzaran a utilizar códigos o juegos de caracteres similares al utilizado en los lenguajes humanos. Así, hoy día la mayoría de las computadoras trabajan con diferentes tipos de juegos de caracteres de los que se destacan el código ASCII y el EBCDIC.

De este modo, una computadora a través de los diferentes lenguajes de programación utiliza un juego o código de caracteres que serán fácilmente interpretados por la computadora y que pueden ser programados por el usuario. Tres son los códigos más utilizados actualmente en computadoras, **ASCII** (American Standard Code for Information Interchange), **EBCDIC** (Extended Binary Coded Decimal Interchange Code) y **Unicode**.

El *código ASCII básico* utiliza 7 *bits* (dígitos binarios, 0, 1) para cada carácter a representar, lo que supone un total de 2^7 (128) caracteres distintos. El código ASCII ampliado utiliza 8 bits y, en ese caso, consta de 256 caracteres. Este código ASCII ha adquirido una gran popularidad, ya que es el estándar en todas las familias de computadoras personales.

El *código EBCDIC* utiliza 8 *bits* por carácter y, por consiguiente, consta de 256 caracteres distintos. Su notoriedad reside en ser el utilizado por la firma IBM (sin embargo, en las computadoras personales PC, XT, AT y PS/2 IBM ha seguido el código ASCII).

El *código universal Unicode* para aplicación en Internet y en gran número de alfabetos internacionales.

En general, un carácter ocupará un *byte* de almacenamiento de memoria.

8.2.1. Código ASCII

El código ASCII se compone de los siguientes tipos de caracteres:

- *Alfabéticos* (`a, b, ..., z/A, B, ..., Z`).
- *Numéricos* (`0, 1, 2, 3, ..., 8, 9`).
- *Especiales* (`+, -, *, /, {, }, <, >, etc.`).
- *De control* son caracteres no imprimibles y que realizan una serie de funciones relacionadas con la escritura, transmisión de datos, separador de archivos, etc., en realidad con los dispositivos de entrada/salida. Destacamos entre ellos:

DEL	*eliminar o borrar*
STX	*inicio de texto*
LF	*avance de línea*
FF	*avance de página*
CR	*retorno de carro*

Los caracteres del 128 al 255, pertenecientes en exclusiva al código ASCII ampliado, no suelen ser estándar y normalmente cada fabricante los utiliza para situar en ellos caracteres específicos de su máquina o de otros alfabetos, caracteres gráficos, etc. En la Figura 8.2 se muestra el código ASCII de la familia de computadoras IBM PC y compatibles, donde se puede apreciar tanto el ASCII básico estándar como el ampliado.

Valor ASCII	Carácter	Valor ASCII	Carácter	Valor ASCII	Carácter	Valor ASCII	Carácter
000	NUL	032	espacio	064	@	096	'
001	SOH	033	!	065	A	097	a
002	STX	034	"	066	B	098	b
003	ETX	035	#	067	C	099	c
004	EOT	036	$	068	D	100	d
005	ENQ	037	%	069	E	101	e
006	ACK	038	&	070	F	102	f
007	BEL	039	'	071	G	103	g
008	BS	040	(	072	H	104	h
009	HT	041	)	073	I	105	i
010	LF	042	*	074	J	106	j
011	VT	043	+	075	K	107	k
012	FF	044	,	076	L	108	l
013	CR	045	-	077	M	109	m
014	SO	046	.	078	N	110	n
015	SI	047	/	079	O	111	o
016	DLE	048	0	080	P	112	p
017	DC1	049	1	081	Q	113	q
018	DC2	050	2	082	R	114	r
019	DC3	051	3	083	S	115	s
020	DC4	052	4	084	T	116	t
021	NAK	053	5	085	U	117	u
022	SYN	054	6	086	V	118	v
023	ETB	055	7	087	W	119	w
024	CAN	056	8	088	X	120	x
025	EM	057	9	089	Y	121	y
026	SUB	058	:	090	Z	122	z
027	ESC	059	;	091	[	123	{
028	FS	060	<	092	\	124	|
029	GS	061	=	093	]	125	}
030	RS	062	>	094	↑	126	~
031	US	063	?	095	_	127	DEL

NOTA: Los 32 primeros caracteres y el último son caracteres de control; no son imprimibles.

Figura 8.1. Código ASCII básico.

8.2.2. Código EBCDIC

Este código es muy similar al ASCII, incluyendo también, además de los caracteres alfanuméricos y especiales, caracteres de control. Es propio de computadoras de IBM, con la excepción de los modelos PC, XT, AT y PS/2.

8.2.3. Código universal Unicode para Internet

Aunque ASCII es un código ampliamente utilizado para textos en inglés, es muy limitado, ya que un código de un byte sólo puede representar 256 caracteres diferentes ($2^8 = 256$). El lenguaje Java comenzó a utilizar la representación internacional *Unicode* más moderna y más amplia en juego de caracteres, ya que es un código de dos bytes (16 bits), que permiten hasta 65.536 caracteres diferentes ($2^{16} = 65.536$).

El código estándar **Unicode** es un estándar internacional que define la representación de caracteres de una amplia gama de alfabetos. Tradicionalmente, como ya se ha comentado, los lenguajes de programación utilizaban el código ASCII cuyo juego de caracteres era 127 (o 256 para el código ASCII ampliado) que se almacenaban en 7 (o en 8) bits y que básicamente incluían aquellos caracteres que aparecían en el teclado estándar (QWERTY). Para los programadores que escriben en inglés estos caracteres son más o menos suficientes. Sin embargo, la aparición de **Java** y posteriormente **C#** como lenguajes universales requieren que éstos puedan ser utilizados en lenguajes internacionales, como español, alemán, francés, chino, etc. Esta característica requiere de más de 256 caracteres diferentes. La representación *Unicode* que admite hasta 65.536 resuelve estos problemas.

D	P	D	P	D	P	D	P	D	P	D	P	D	P	D	P
0		32		64	@	96		128	Ç	160	á	192	└	224	α
1	☺	33	!	65	A	97	a	129	ü	161	í	193	┴	225	β
2		34	"	66	B	98	b	130	é	162	ó	194	┬	226	Γ
3	♥	35	#	67	C	99	c	131	â	163	ú	195	├	227	π
4	♦	36	$	68	D	100	d	132	ä	164	ñ	196	-	228	Σ
5	♣	37	%	69	E	101	e	133	à	165	Ñ	197	┼	229	σ
6	♠	38	&	70	F	102	f	134	å	166	ª	198	╞	230	µ
7	•	39	´	71	G	103	g	135	ç	167	º	199		231	γ
8	◘	40	(	72	H	104	h	136	ê	168	¿	200	╚	232	φ
9	○	41	)	73	I	105	i	137	ë	169	⌐	201	╔	233	θ
10	■	42	*	74	J	106	j	138	è	170	¬	202	╩	234	Ω
11	♂	43	+	75	K	107	k	139	ï	171	½	203	╦	235	δ
12	♀	44	,	76	L	108	l	140	î	172	¼	204	╠	236	∞
13	♪	45	-	77	M	109	m	141	ì	173	¡	205	═	237	∅
14	♫	46	.	78	N	110	n	142	Ä	174	«	206	╬	238	∈
15	☼	47	/	79	O	111	o	143	Å	175	»	207	╧	239	∩
16	►	48	0	80	P	112	p	144	É	176		208	╨	240	≡
17	◄	49	1	81	Q	113	q	145	æ	177	▮	209	╤	241	±
18	↕	50	2	82	R	114	r	146	Æ	178	▮	210	╥	242	≥
19	!!	51	3	83	S	115	s	147	ô	179	▮	211	╙	243	≤
20	¶	52	4	84	T	116	t	148	ö	180	┤	212	╘	244	∫
21	§	53	5	85	U	117	u	149	ò	181	╡	213	╒	245	∫
22	■	54	6	86	V	118	v	150	û	182	╢	214	╓	246	÷
23	↨	55	7	87	W	119	w	151	ù	183	╖	215	╫	247	≈
24	↑	56	8	88	X	120	x	152	Ÿ	184	╕	216	╪	248	°
25	↓	57	9	89	Y	121	y	153	Ô	185	╣	217	┘	249	•
26	→	58	:	90	Z	122	z	154	Ü	186	║	218	┌	250	·
27	←	59	;	91	[	123	{	155	¢	187	╗	219	▮	251	√
28	⌙	60	<	92	\	124	|	156	£	188	╝	220	▪	252	ⁿ
29	↔	61	=	93	]	125	}	157	¥	189	╜	221	▌	253	²
30	▲	62	>	94	^	126	~	158	Pt	190	╛	222	▐	254	▮
31	▼	63	?	95	_	127	⌂	159	*f*	191	┐	223	▪	255	

D: Código decimal.
P: Escritura del carácter correspondiente al código en la pantalla.

Figura 8.2. Código ASCII de la computadora IBM PC.

D	C	D	C	D	C	D	C	D	C	D	C	D	C	D	C
0	NUL	21	NL	43	CU2	79	,	124	@	150	o	195	C	227	T
1	SOH	22	BS	45	ENQ	80	&	125	´	151	p	196	D	228	U
2	STX	23	IL	46	ACK	90	!	126	=	152	q	197	E	229	V
3	ETX	24	CAN	47	BEL	91	$	127	"	153	r	198	F	230	W
4	PF	25	EM	50	SYN	92	*	129	a	155	}	199	G	231	X
5	HT	26	CC	52	PN	93	)	130	b	161	~	200	H	232	Y
6	LC	27	CU1	53	RS	94	;	131	c	162	s	201	I	233	Z
7	DEL	28	IFS	54	UC	95	¬	132	d	163	t	208	}	240	0
10	SMM	29	IGS	55	EOT	96	–	133	e	164	u	209	J	241	1
11	VT	30	IRS	59	CU3	97	/	134	f	165	v	210	K	242	2
12	FF	31	IUS	60	DC4	106	¦	135	g	166	w	211	L	243	3
13	CR	32	DS	61	NAK	107	,	136	h	167	x	212	M	244	4
14	SO	33	SOS	63	SUB	108	%	137	i	168	y	213	N	245	5
15	SI	34	FS	64	SP	109	_	139	{	169	z	214	O	246	6
16	DLE	36	BYP	74	c	110	>	145	j	173	[	215	P	247	7
17	DC1	37	LF	75	.	111	?	146	k	189	]	216	Q	248	8
18	DC2	38	ETB	76	<	121	`	147	l	192	{	217	R	249	9
19	DC3	39	ESC	77	(	122	:	148	m	193	A	224	\	250	I
20	RES	40	SM	78	+	123	#	149	n	194	B	226	S		

D: Código decimal.
P: Escritura del carácter correspondiente al código en la pantalla.

Figura 8.3. Código EBCDIC.

En consecuencia, los identificadores en Java y C# deben comenzar con una letra Java o C#, que es cualquier carácter Unicode que no represente un dígito o un carácter de puntuación.

Las letras en inglés, así como los dígitos decimales y los signos de puntuación en inglés, se asignan a los códigos que son los mismos que en el código ASCII. Puede consultar los caracteres *Unicode* en el sitio Web oficial del consorcio **Unicode**:

```
http://www.unicode.org
```

8.2.4. Secuencias de escape

Una **secuencia de escape** es un medio de representar caracteres que no se pueden escribir desde el teclado y, por consiguiente, utilizarlos directamente en un editor. Una secuencia de escape consta de dos partes: el *carácter escape* y un *valor de traducción*. El carácter escape es un símbolo que indica al compilador Java o C (por ejemplo) que ha de traducir el siguiente carácter de un modo especial. En Java, como en lenguaje C, este carácter de escape especial es la barra inclinada inversa (\).

Si la barra inclinada marca el principio de una secuencia de escape, ¿qué se puede utilizar para el valor de la traducción? La parte de la secuencia de escape que sigue al carácter escape y, tal vez, el valor de traducción más fácil para utilizar es un código de carácter Unicode. Los valores Unicode deben estar especificados como un número hexadecimal de cuatro dígitos precedido por una letra `u`. Los literales de caracteres Java o C# se deben encerrar entre comillas simples

Sintaxis `'\uxxxx'`
Ejemplos `'\u0344'` `'\u2122'`

En programas escritos en cualquier lenguaje (en particular en Java o en C#) se pueden utilizar las secuencias de escape Unicode en cualquier parte donde algún tipo de carácter pueda aparecer: en literales "carácter", en literales "cadenas" o incluso en identificadores.

Todos los lenguajes de programación (C, C++, Java, etc.) permiten especificar el carácter de escape para especificar otros tipos de caracteres especiales. Estos caracteres incluyen algunos de los "caracteres invisibles" que se han utilizado tradicionalmente para controlar operaciones de computadora (a veces se les conoce también como "caracteres de control") así como simples comillas, dobles comillas y el propio carácter de escape. Así, para escribir una comilla simple como un literal carácter, se escribe '\''. La Tabla 8.1 proporciona las secuencias de escape que el lenguaje Java reconoce.

Tabla 8.1. Secuencias de escape en Java

Secuencia	Significado
`\b`	Retroceso `(\u0008)`
`\t`	Tabulación `(\U0009)`
`\n`	Nueva línea `(\u000A)`
`\f`	Avance de página `(\u000C)`
`\r`	Retorno de carro `(\u000D)`
`\"`	Dobles comillas `(\u0022)`
`\'`	Comillas simples `(\u0027)`
`\\`	Barra inclinada inversa `(\u005C)`
`\\ddd`	Cualquier carácter especificado por dígitos octales *ddd*

8.3. CADENA DE CARACTERES

Una *cadena* (*string*) de caracteres es un conjunto de caracteres —incluido el blanco— que se almacenan en un área contigua de la memoria. Pueden ser entradas o salidas a/desde un terminal.

La *longitud* de una cadena es el número de caracteres que contiene. La cadena que no contiene ningún carácter se le denomina *cadena vacía o nula*, y su longitud es cero; no se debe confundir con una cadena compuesta sólo de blancos —espacios en blanco—, ya que ésta tendrá como longitud el número de blancos de la misma.

La representación de las cadenas suele ser con comillas simples o dobles. En nuestro libro utilizaremos las comillas simples por ser esa notación la más antigua utilizada en diversos lenguajes como **Pascal**, **FORTRAN**, etc., aunque hoy día los lenguajes modernos, tales como **C**, **C++**, **Java** y **C#**, utilizan las dobles comillas.

Notaciones de cadenas

Pascal, FORTRAN, UPSAM	`'Cartagena de Indias'`
C, C++, Java, C#	`"Cartagena de Indias"`

EJEMPLO 8.1

```
'12 de octubre de 1492'
'Por fin llegaste'
'           '
'AMERICA ES GRANDE'
```

Las cadenas pueden contener cualquier carácter válido del código aceptado por el lenguaje y la computadora; el blanco es uno de los caracteres más utilizado; si se le quiere representar de modo especial en la escritura en papel, se emplea alguno de los siguientes símbolos:

_ ␣ □ ∪

Por nuestra parte utilizaremos _, dejando libertad al lector para usar el que mejor convenga a su estilo de programación. Las cadenas anteriores tienen longitudes respectivas de 21, 16, 3 y 17.

Una *subcadena* es una cadena de caracteres que ha sido extraída de otra de mayor longitud.

`'12 de'`	*es una subcadena de*	`'12 de octubre'`
`'Java'`	*es una subcadena de*	`'lenguaje Java'`
`'CHE'`	*es una subcadena de*	`'CARCHELEJO'`

Reglas de sintaxis en lenguajes de programación

C++ .... Una cadena es un array de caracteres terminado con el carácter nulo, cuya representación es la secuencia de escape `'0'` y su nombre es NULL (nulo).

C# .... Las cadenas son objetos del tipo incorporado *`String`*. En realidad, String es una clase que proporciona funcionalidades de manipulación de cadenas y en particular construcción de cadenas.

Java... Las cadenas son objetos del tipo *`String`*. *`String`* es una clase en Java y una vez que los objetos cadena se crean, el contenido no se puede modificar, aunque pueden ser construidas todas las cadenas que se deseen.

EJEMPLO 8.2

Cadena 'Carchelejo' representada en lenguaje C++

C	A	R	C	H	E	L	E	J	O	*\0*

8.4. DATOS TIPO CARÁCTER

En el Capítulo 3 se analizaron los diferentes tipos de datos y entre ellos existía el dato tipo *carácter (char)* que se incorpora en diferentes lenguajes de programación, bien con este nombre o bien como datos tipo cadena. Así pues, en esta sección trataremos las constantes y las variables tipo carácter o cadena.

8.4.1. Constantes

Una constante tipo carácter es un carácter encerrado entre comillas y una constante de tipo cadena es un conjunto de caracteres válidos encerrados entre comillas —apóstrofos— para evitar confundirlos con nombres de variables, operadores, enteros, etc. Si se desea escribir un carácter comilla, se debe escribir duplicado. Como se ha comentado anteriormente, existen lenguajes —BASIC, C, C++, Java, etc., por ejemplo— que encierran las cadenas entre dobles comillas. Nuestros algoritmos sólo tendrán una, por seguir razones históricas y por compatibilidad con versiones anteriores del lenguaje UP SAM.

```
'Carchelejo es un pueblo de Jaen'
```

es una constante de tipo cadena, de una longitud fija igual a 31.

```
'¿'
```

es una constante de tipo carácter.

8.4.2. Variables

Una *variable de cadena* o *tipo carácter* es una variable cuyo valor es una cadena de caracteres.

Las variables de tipo carácter o cadena se deben declarar en el algoritmo y según el lenguaje tendrán una notación u otra. Nosotros, al igual que muchos lenguajes, las declararemos en la tabla o bloque de declaración de variables.

```
var
  caracter : A, B
  cadena : NOMBRE, DIRECCION
```

Atendiendo a la declaración de la longitud, las variables se dividen en *estáticas, semiestáticas* y *dinámicas*.

Variables estáticas son aquellas en las que su longitud se define antes de ejecutar el programa y ésta no puede cambiarse a lo largo de éste.

FORTRAN:
```
CHARACTER A1 * 10, A2 * 15
```

las variables `A1` y `A2` se declaran con longitudes `10` y `15`, respectivamente.

Pascal:
```
var NOMBRE: PACKED ARRAY [1..30] OF CHAR
```
Turbo Pascal:
```
var NOMBRE: array[1..30] of char      o bien
var NOMBRE:STRING[30]
```

En Pascal, una variable de tipo carácter —*char*— sólo puede almacenar un carácter y, por consiguiente, una cadena de caracteres debe representarse mediante un *array* de caracteres. En el ejemplo, `NOMBRE` se declara como una cadena de 30 caracteres (en este caso, `NOMBRE[1]` será el primer carácter de la cadena, `NOMBRE[2]` será el segundo carácter de la cadena, etc.).

Turbo Pascal admite también tratamiento de cadenas semiestáticas `(STRING)` como dato.

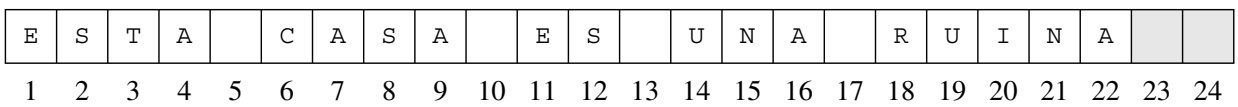
Variables semiestáticas son aquellas cuya longitud puede variar durante la ejecución del programa, pero sin sobrepasar un límite máximo declarado al principio.

Variables dinámicas son aquellas cuya longitud puede variar sin limitación dentro del programa. El lenguaje SNOBOL es típico de variables dinámicas.

La representación de las diferentes variables de cadena en memoria utiliza un método de almacenamiento diferente.

Cadenas de longitud fija

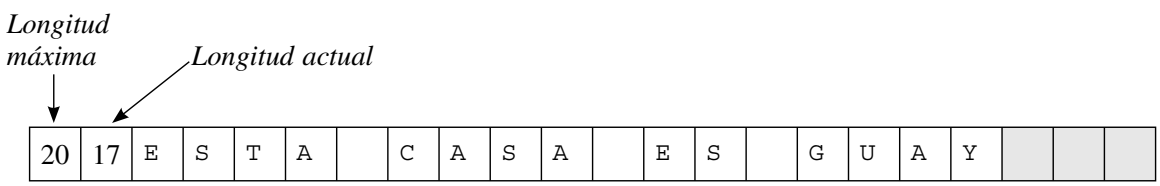
Se consideran vectores de la longitud declarada, con blancos a izquierda o derecha si la cadena no tiene la longitud declarada. Así, la cadena siguiente



se declaró con una dimensión de 24 caracteres y los dos últimos se rellenan con blancos.

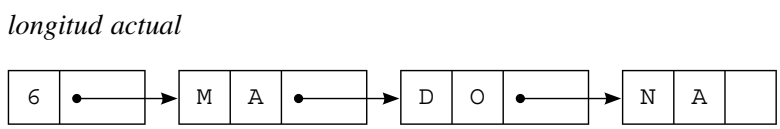
Cadenas de longitud variable con un máximo

Se considera un puntero (en el Capítulo 12 ampliaremos este concepto) con dos campos que contienen la longitud máxima y la longitud actual.



Cadenas de longitud indefinida

Se representan mediante listas enlazadas, que son listas que se unen mediante puntero



Estas listas contienen elementos con caracteres empaquetados —2/elemento— y enlazados cada uno con el siguiente por un puntero (la cadena de caracteres es `'MADONA'`).

8.4.3. Instrucciones básicas con cadenas

Las instrucciones básicas: *asignar* y *entrada/salida (leer/escribir)* se realizan de un modo similar al tratamiento de dichas instrucciones con datos numéricos.

Asignación

Si la variable `NOMBRE` se ha declarado como tipo cadena

```
var cadena : NOMBRE
```

la instrucción de asignación debe contener en el lado derecho de la asignación una constante tipo cadena o bien otra variable del mismo tipo. Así,

```
NOMBRE ←'Luis Hermenegildo'
```

significa que la variable `NOMBRE` toma por valor la cadena `'Luis Hermenegildo'`.

Entrada/Salida

La entrada/salida desde un terminal se puede realizar en modo carácter; para ello bastará asignar —a través del correspondiente dispositivo— una cadena de caracteres a una variable tipo cadena. Así, por ejemplo, si `A`, `B`, `C` y `D` se han declarado como variables tipo cadena

```
var cadena : A, B, C, D
```

las instrucciones

```
leer(A, B)
escribir(C, D)
```

asignarán a `A` y `B` las cadenas introducidas por el teclado y visualizarán o imprimirán en el dispositivo de salida las cadenas que representan las variables `C` y `D`.

8.5. OPERACIONES CON CADENAS

El tratamiento de cadenas es un tema importante, debido esencialmente a la gran cantidad de información que se almacena en ellas. Según el tipo de lenguaje de programación elegido se tendrá mayor o menor facilidad para la realización de operaciones. Así, por ejemplo, **C** tiene grandes posibilidades, **FORTRAN** sólo operaciones elementales y **Pascal**, dependiendo del compilador, soporta procedimientos y funciones predefinidas o es preciso definirlos por el usuario con la natural complejidad que suponga el diseño del algoritmo correspondiente. Todos los lenguajes orientados a objetos como **C++**, **C#** y **Java**, merced a la clase String soportan una gran gama de funciones de manipulación de cadenas. En cualquier caso, las operaciones con cadena más usuales son:

- Cálculo de la longitud.
- Comparación.
- Concatenación.
- Extracción de *subcadenas*.
- Búsqueda de información.

8.5.1. Cálculo de la longitud de una cadena

La longitud de una cadena, como ya se ha comentado, es el número de caracteres de la cadena. Así,

```
'Don Quijote de la Mancha'
```

tiene veinticuatro caracteres.

La operación de determinación de la longitud de una cadena se representará por la función **longitud**, cuyo formato es:

longitud *(cadena)*

La función **longitud** tiene como argumento una cadena, pero su resultado es un valor numérico entero:

```
longitud('Don Quijote de la Mancha')   proporciona 24
longitud('□□□')                        cadena de tres blancos proporciona 3
longitud('□□□Mortadelo')               cadena 'Mortadelo' rellenada de blancos a la izquierda
                                       para tener longitud 12
```

En consecuencia, la función **longitud** se puede considerar un dato tipo entero y, por consiguiente, puede ser un operando dentro de expresiones aritméticas.

```
4 + 5 + longitud('DEMO') = 4+5+4 = 13
```

8.5.2. Comparación

La *comparación* de cadenas (igualdad y desigualdad) es una operación muy importante, sobre todo en la clasificación de datos tipo carácter que se utiliza con mucha frecuencia en aplicaciones de proceso de datos (clasificaciones de listas, tratamiento de textos, etc.).

Los criterios de comparación se basan en el orden numérico del código o juego de caracteres que admite la computadora o el propio lenguaje de programación. En nuestro lenguaje algorítmico utilizaremos el código ASCII como código numérico de referencia. Así,

- El carácter `'A'` *será <* el carácter `'C'`
  (código 65) *(código 67)*
- El carácter `'8'` *será <* el carácter `'i'`
  (código 56) *(código 105)*

En la comparación de cadenas se pueden considerar dos operaciones más elementales: *igualdad* y *desigualdad*.

Igualdad

Dos cadenas *a* y *b* de longitudes *m* y *n* son iguales si:

- El número de caracteres de *a* y *b* son los mismos *(m = n)*.
- Cada carácter de *a* es igual a su correspondiente de *b* si $a = a_1a_2...a_n$ y $b = b_1b_2...b_n$ se debe verificar que $a_i = b_i$ para todo *i* en el rango *1 <= i <= n.*

Así:
```
'EMILIO' = 'EMILIO'      es una expresión verdadera
'EMILIO' = 'EMILIA'      es una expresión falsa
'EMILIO' = 'EMILIO '     es una expresión falsa; contiene un blanco final y, por consiguiente, las
                         longitudes no son iguales.
```

Desigualdad

Los criterios para comprobar la desigualdad de cadena son utilizados por los operadores de relación <, <=, >=, < > y se ajustan a una comparación sucesiva de caracteres correspondientes en ambas cadenas hasta conseguir dos caracteres diferentes. De este modo, se pueden conseguir clasificaciones alfanuméricas

```
'GARCIA' < 'GOMEZ'
```

ya que las comparaciones sucesivas de caracteres es:

```
G-A-R-C-I-A     G = G, A < O, ...
G-O-M-E-Z
```

una vez que se encuentra una desigualdad, no es preciso continuar; como se observa, las cadenas no tienen por qué tener la misma longitud para ser comparadas.

EJEMPLO 8.3

En las sucesivas comparaciones se puede apreciar una amplia gama de posibles casos.

```
'LUIS'        <      'LUISITO'       verdadera
'ANA'         <      'MARTA'         verdadera
'TOMAS'       <      'LUIS'          falsa
'BARTOLO'     <=     'BARTOLOME'     verdadera
'CARMONA'     >      'MADRID'        falsa
'LUIS '       >      'LUIS'          verdadera
```

Se puede observar de los casos anteriores que la presencia de cualquier carácter —incluso el blanco—, se considera mayor siempre que la ausencia. Por eso, `'LUIS '` es mayor que `'LUIS'`.

8.5.3. Concatenación

La concatenación es la operación de reunir varias cadenas de caracteres en una sola, pero conservando el orden de los caracteres de cada una de ellas.

El símbolo que representa la concatenación varía de unos lenguajes a otros. Los más utilizados son:

```
+   //     &      o
```

En nuestro libro utilizaremos `&` y en ocasiones `+`. El símbolo `&` evita confusiones con el operador suma. Las cadenas para concatenarse pueden ser constantes o variables.

```
'MIGUEL'&'DE'&'CERVANTES' == 'MIGUELDECERVANTES'
```

Puede comprobar que las cadenas, en realidad, se *"pegan"* unas al lado de las otras; por ello, si al concatenar frases desea dejar blancos entre ellas, deberá indicarlos expresamente en alguna de las cadenas. Así, las operaciones

```
'MIGUEL '&'DE '&'CERVANTES
'MIGUEL'&' DE'&' CERVANTES
```

producen el mismo resultado

```
'MIGUEL DE CERVANTES'
```

lo que significa que la *propiedad asociativa* se cumple en la operación de concatenación.

El operador de concatenación (+, &) actúa como un operador aritmético.

EJEMPLO 8.4

Es posible concatenar variables de cadena.

```
var cadena : A, B, C
A&B&C equivale a A&(B&C)
```

La asignación de constantes tipo cadena a variables tipo cadena puede también realizarse con expresiones concatenadas.

EJEMPLO 8.5

Las variables A, B son de tipo cadena.

```
var cadena : A, B
A ← 'FUNDAMENTOS'
B ← 'DE PROGRAMACION'
```

La variable `C` puede recibir como valor

```
C ← A+' '+B
```

que produce un resultado de

```
C = 'FUNDAMENTOS DE PROGRAMACION'
```

Concatenación en Java:

El lenguaje Java soporta la concatenación de cadenas mediante el operador + que actúa sobrecargado. Así, suponiendo que la cadena c1 contiene "Fiestas de moros" y la cadena c2 contiene "y cristianos", la cadena c1 + c2 almacenará "Fiestas de moros y cristianos".

8.5.4. Subcadenas

Otra operación —función— importante de las cadenas es aquella que permite la extracción de una parte específica de una cadena: *subcadena*. La operación *subcadena* se representa en dos formatos por:

```
subcadena (cadena, inicio, longitud)
```

- *Cadena* es la cadena de la que debe extraerse una subcadena.
- *Inicio* es un número o expresión numérica entera que corresponde a la posición inicial de la subcadena.
- *Longitud* es la longitud de la subcadena.

```
subcadena (cadena, inicio)
```

En este caso, la subcadena comienza en *inicio* y termina en el final de la cadena.

EJEMPLOS

```
subcadena ('abcdef', 2, 4)     equivale a      'bcde'
subcadena ('abcdef', 6, 1)     equivale a      'f'
subcadena ('abcdef', 3)        equivale a      'cdef'
subcadena ('abcdef', 3, 4)     equivale a      'cdef'
     longitud = 5 caracteres
               ^^^^^

subcadena ('12 DE OCTUBRE', 4, 5) = DE OC
              ↑
          posición 4
```

Es posible realizar operaciones de concatenación con subcadenas.

```
subcadena ('PATO DONALD',1,4)+ subcadena ('ESTA TIERRA',5,4)
```

equivale a la cadena `'PATO TIE'`.

La aplicación de la función a una subcadena,

```
subcadena (cadena, inicio, fin)
```

puede producir los siguientes resultados:

1. Si *fin* no existe, entonces la subcadena comienza en el mismo carácter inicio y termina con el último carácter.
2. Si *fin* <= 0, el resultado es una cadena vacía.
3. Si *inicio* > *longitud (cadena)*, la subcadena resultante será vacía.
   `subcadena ('MORTIMER', 9, 2)`
   produce una cadena vacía.
4. Si *inicio* <= 0, el resultado es también una cadena vacía.
   `subcadena ('valdez', 0, 4)` *y* `subcadena ('valdez', 8)`
   proporcionan cadenas nulas.

8.5.5. Búsqueda

Una operación frecuente a realizar con cadenas es localizar si una determinada cadena forma parte de otra cadena más grande o buscar la posición en que aparece un determinado carácter o secuencia de caracteres de un texto.

Estos problemas pueden resolverse con las funciones de cadena estudiadas hasta ahora, pero será necesario diseñar los algoritmos correspondientes. Esta función suele ser interna en algunos lenguajes y la definiremos por **indice** o **posicion**, y su formato es

```
indice (cadena, subcadena)
```

o bien

```
posicion (cadena, subcadena)
```

donde *subcadena* es el texto que se trata de localizar.

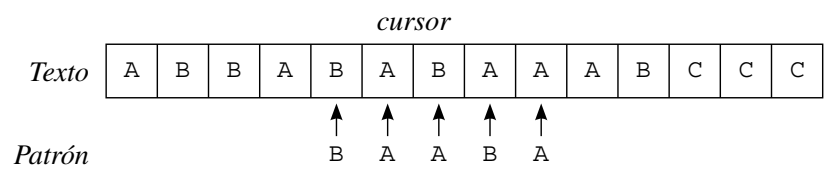
El resultado de la función es un valor entero:

- Igual a *P* >= 1, donde *P* indica la posición del primer carácter de la primera coincidencia de subcadena en cadena.
- Igual a cero, si subcadena es una cadena vacía o no aparece en la cadena.

Así, suponiendo la `cadena C = 'LA CAPITAL ES MADRID'`

```
indice (C, 'CAP')        toma un valor 4
indice (C, ' ES ')       toma un valor 11
indice (C, 'PADRID')     toma un valor 0
```

La función **indice** en su forma más general realiza la operación que se denomina ***coincidencia de patrones*** *(patter-matching).* Esta operación busca una cadena patrón o modelo dentro de una cadena de texto.



Esta operación utiliza un cursor o puntero en la cadena de texto original y va comprobando los sucesivos valores de ambas cadenas: si son distintos, produce un 0, y si no proporciona la posición del primer carácter coincidente.

```
indice ('ABCDE', 'F')          produce 0
indice ('ABXYZCDEF', 'XYZ')    produce 3
```

La función **indice (posicion)** al tomar también un valor numérico entero se puede utilizar en expresiones aritméticas o en instrucciones de asignación a variables numéricas.

```
P ← indice (C, 'F')
```

8.6. OTRAS FUNCIONES DE CADENAS

Existen otras funciones de cadena internas al lenguaje o definidas por el usuario, que suelen ser de utilidad en programación y cuyo conocimiento es importante que conozca el lector:

- *Insertar* cadenas.
- *Borrar* cadenas.

- *Cambiar* cadenas.
- *Convertir* cadenas en números y viceversa.

8.6.1. Insertar

Si se desea insertar una cadena `C` dentro de un texto o cadena más grande, se debe indicar la posición. El formato de la función **`insertar`** es

```
insertar (t, p, s)
```

- `t` texto o cadena *donde* se va a insertar.
- `p` posición *a partir de la cual* se va a insertar.
- `s` *subcadena* que se va a *insertar*.

```
insertar ('ABCDEFGHI', 4, 'XXX') = 'ABCXXXDEFGHI'
insertar ('MARIA O', 7, 'DE LA ') = 'MARIA DE LA O'
```

Algoritmo de inserción

Si su lenguaje no posee definida esta función, se puede implementar con el siguiente algoritmo:

```
inicio
  insertar(t,p,s) = subcadena(t,1,p-1)& S &
    subcadena(t,p,longitud(t)-p+1)
fin
```

Veámoslo con un ejemplo: `insertar ('ABCDEFGHI' 4, 'XXX')`

donde `t = 'ABCDEFGHI' y S = 'XXX' p = 4`

```
subcadena (t,1,p-1) = subcadena (t,1,3) = ABC
subcadena (t,p,longitud(t)-p+1) = subcadena (t,4,9-4+1) =
subcadena (t,4,6) = DEFGHI
```

por consiguiente,

```
insertar ('ABCDEFGHI',4'XXX')= 'ABC'+'XXX'+'DEFGHI'='ABCXXXDEFGHI'
```

8.6.2. Borrar

Si se desea eliminar una subcadena que comienza en la posición `p` y tiene una longitud `l` se tiene la función **`borrar`**.

```
borrar (t, p, l)
```

- `t` texto o cadena de donde se va a eliminar una subcadena,
- `p` posición a partir de la cual se va a borrar (eliminar),
- `l` longitud de la subcadena a eliminar,

```
borrar ('supercalifragilístico', 6, 4) = 'superfragilístico'
borrar ('supercalifragilístico', 3, 10) = 'sugilístico'
```

Algoritmo borrar

Si no se posee la función estándar **borrar**, será preciso definirla. Ello se consigue con el algoritmo,

```
inicio
  borrar (t,p,1) = subcadena (t,1,p-1) &
    subcadena (t,p+1,longitud(t)-p-1+1)
fin
```

8.6.3. Cambiar

La operación insertar trata de sustituir en un texto *t* la primera ocurrencia de una subcadena `S1` por otra `S2`. Este es el caso frecuente en los programas de tratamiento de textos, donde a veces es necesario sustituir una palabra cualquiera por otra (... en el archivo `"DEMO"` sustituir la palabra "ordenador" por "computadora"), acomodando las posibles longitudes diferentes. La función que realiza la operación de insertar tiene el formato

cambiar *(t, S1, S2)*

- *t* texto donde se realizarán los cambios.
- *S1* subcadena a sustituir.
- *S2* subcadena nueva.

```
cambiar ('ABCDEFGHIJ', 'DE', 'XXX') = 'ABCXXXFGHIJ'
```

Si la subcadena `S1` no coincide exactamente con una subcadena de `t`, no se produce ningún cambio y el texto o cadena original no se modifica

```
cambiar ('ABCDEFGHIJK', 'ZY', 'XXX') = 'ABCDEFGHIJK'
```

Algoritmo cambio

Si no se dispone de esta función como estándar, es posible definir un algoritmo haciendo uso de las funciones analizadas.

```
cambiar (t, S1, S2)
```

El algoritmo se realiza llamando a las funciones **indice**, **borrar** e **insertar**.

```
procedimiento cambiar(t, S1, S2)
inicio
  j ← indice(t, S1)
  t ← borrar(t, j, longitud(S1))
  insertar(t, j, S2)
fin
```

La primera instrucción, `j ← indice(s, S1)`, calcula la posición donde se debe comenzar la inserción, que es, a su vez, el primer elemento de la subcadena `S1`.

La segunda instrucción

```
t ← borrar(t, j, longitud(S1))
```

borra la subcadena `S1` y la nueva cadena se asigna a la variable de cadena `t`.

La tercera instrucción inserta en la nueva cadena `t` —original sin la cadena `S1`— la subcadena `S2` a partir del carácter de posición `j`, como se había previsto.

8.6.4. Conversión de cadenas/números

Existen funciones o procedimientos en los lenguajes de programación (**val** y **str** en BASIC, **val** y **str** en Turbo Pascal) que permiten convertir un número en una cadena y viceversa.

En nuestro algoritmo los denotaremos por **valor** y **cad**.

valor *(cadena)*	*convierte la cadena en un número; siempre que la cadena fuese de dígitos numéricos*
cad *(valor)*	*convierte un valor numérico en una cadena*

EJEMPLOS

```
valor ('12345') = 12345
cad (12345) = '12345'
```

Otras funciones importantes relacionadas con la conversión de caracteres en números y de números en caracteres son

código *(un_caracter)*	*devuelve el código ASCII de un carácter*
car *(un_codigo)*	*Devuelve el carácter asociado en un código ASCII*

ACTIVIDADES DE PROGRAMACIÓN RESUELTAS

8.1. *Se desea eliminar los blancos de una frase dada terminada en un punto. Se supone que es posible leer los caracteres de la frase de uno en uno.*

Solución

Análisis

Para poder efectuar la lectura de la frase, almacena ésta en un array de caracteres `(F)` —esto es posible en lenguajes como Pascal; en BASIC sería preciso recurrir a enojosas tareas de operaciones con funciones de cadenas `MID$`, `LEFT$` o `RIGHT$`—, de modo que `F[i]` contiene el carácter *i*-ésimo de la frase dada. Construiremos una nueva frase sin blancos en otro array `G`.

Algoritmo

Los pasos a dar para la realización del algoritmo son:

- Inicializar contador de letras de la nueva frase `G`.
- Leer el primer carácter.
- Repetir.
  Si el primer carácter no es en blanco, entonces
  escribir en el lugar siguiente del array G,
  leer carácter siguiente de la frase dada.
  Hasta que el último carácter se encuentre.
- Escribir la nueva frase —`G`— ya sin blancos.

Tabla de variables:

F array de caracteres de la frase dada.
G array de caracteres de la nueva frase.
I contador del array F.
J contador del array G.

Pseudocódigo:

```
algoritmo blanco
inicio
  I ← 1
  J ← 0
  F[i] ← leercar() {leercar es una función que permite la lectura de un carácter}
  repetir
    si F[I] <> ' ' entonces
      J ← J+1
      G[I] ← F[I]
    fin_si
    I ← I+1
    F[i] ← leercar()
  hasta_que F[I] = '.'
  //escritura de la nueva frase G
  desde I ← 1 hasta J hacer
    escribir(G[I])    //no avanzar linea
  fin_desde
fin
```

8.2. *Leer un carácter y deducir si está situado antes o después de la letra "m" en orden alfabético.*

Solución

Análisis

La comparación de datos de tipo carácter se realiza mediante los códigos numéricos ASCII, de modo que una letra estará situada antes o después de ésta si su código ASCII es menor o mayor. La propia computadora se encarga de realizar la comparación de datos tipo carácter de acuerdo al código ASCII, siempre que los datos a comparar sean de tipo carácter. Por ello se deben declarar de tipo carácter las variables que representan las comparaciones.

```
Variables C: caracter
```

Pseudocódigo

```
algoritmo caracter
var
  carácter : C
inicio
  leer(C)
  si C < 'M' entonces
    escribir(C, 'esta antes que M en orden alfabético')
  si_no
    escribir(C, 'esta despues que M en orden alfabético')
  fin_si
fin
```

8.3. *Leer los caracteres y deducir si están en orden alfabético.*

Solución

Tabla de variables

```
CAR1, CAR2: caracter
```

Pseudocódigo

```
algoritmo comparacion
var
  carácter : CAR1, CAR2
inicio
  leer(CAR1, CAR2)
  si CAR1 <= CAR2 entonces
    escribir('en orden')
  si_no
    escribir('desordenados')
  fin_si
fin
```

8.4. *Leer una letra de un texto. Deducir si está o no comprendida entre las letras mayúsculas I-M inclusive.*

Solución

Variables

```
LETRA: caracter.
```

Pseudocódigo

```
algoritmo
var
  carácter : LETRA
inicio
  leer(LETRA)
  si (LETRA > = 'I') y (LETRA =< 'M') entonces
    escribir('esta comprendida')
  si_no
    escribir('no esta comprendida')
  fin_si
fin
```

8.5. *Contar el número de letras "i" de una frase terminada en un punto. Se supone que las letras pueden leerse independientemente.*

Solución

En este algoritmo el contador de letras sólo se incrementa cuando se encuentran las letras "i" buscadas.

Pseudocódigo

```
algoritmo letras_i
var
  entero : N
  carácter : LETRA
inicio
  N ← 0
  repetir
    LETRA ← leercar()
    si LETRA = 'i' entonces
      N ← N+1
    fin_si
  hasta_que LETRA = '.'
  escribir('La frase tiene', N, 'letras i')
fin
```

8.6. *Contar el número de vocales de una frase terminada en un punto.*

Solución

Pseudocódigo

```
algoritmo vocales
var
  entero : NUMVOCALES
  carácter : C
inicio
  repetir
    C ← leercar()      {la función leercar permite la lectura de caracteres independientes}
    si C = 'a' o C = 'e' o C = 'i' o C = 'o' o C = 'u' entonces
      NUMVOCALES ← NUMVOCALES+1
    fin_si
  hasta_que C = '.'
  escribir('El numero de vocales es =', NUMVOCALES)
fin
```

8.7. *Se desea contar el número de letras "a" y el número de letras "b" de una frase terminada en un punto. Se supone que es posible leer los caracteres independientemente.*

Solución

Método 1

```
algoritmo letras_a_b
var
  entero : NA, NB
  carácter : C
inicio
  NA ← 0
  NB ← 0
  repetir
    C ← leercar()
    si C = 'a' entonces
      NA ← NA+1
    fin_si
    si C = 'b' entonces
      NB ← NB+1
    fin_si
  hasta_que C = '.'
  escribir('Letras a =', NA, 'Letras b=', NB)
fin
```

Método 2

```
algoritmo letras_a_b
var
  entero : NA, NB
  carácter : C
inicio
  NA ← 0
  NB ← 0
  repetir
    C ← leercar()
    si C = 'a' entonces
      NA ← NA+1
```

```
      si_no
        si C = 'b' entonces
          NB ← NB+1
        fin_si
      fin_si
    hasta_que C = '.'
  fin
```

Método 3

```
  algoritmo letras_a_b
  var
    entero : NA, NB
    carácter : C
  inicio
    NA ← 0
    NB ← 0
    repetir
      C ← leercar()
      según_sea C hacer
        'a': NA ← NA+1
        'b': NB ← NB+1
      fin_según
    hasta_que C = '.'
  fin
```

8.8. *Leer cien caracteres de un texto y contar el número de letras "b".*

Solución

Tabla de variables

```
entero : I, NE
caracter : C
```

Pseudocódigo

```
algoritmo letras_b
var
  entero : I, NE
  carácter : C
inicio
  NE ← 0
  desde I ← 1 hasta 100 hacer
    C ← leercar()
    si C = 'b' entonces
      NE ← NE+1
    fin_si
  fin_desde
  escribir('Existen', NE, 'letras b')
fin
```

8.9. *Escribir una función convertida (núm,b) que nos permita transformar un número entero y positivo en base 10 a la base que le indiquemos como parámetro. Comprobar el algoritmo para las bases 2 y 16.*

```
algoritmo Cambio_de_base
  var entero: num, b
  inicio
    escribir('Déme número')
    leer(num)
```

```
    escribir('Indique base')
    leer(b)
    escribir(convertir(num,b),'es el número', num, 'en base',b)
  fin

cadena función convertir(E entero: num,b)
  var entero: r
      carácter: c
      cadena: unacadena
  inicio
    unacadena ← ''
    si num > 0 entonces
      mientras num > 0 hacer
        r ← num MOD b
        si r > 9 entonces
          c ← car(r+55)
          si_no
          c ← car(r + codigo('0'))
        fin_si
        unacadena ← c + unacadena
        num ← num div b
      fin_mientras
    si_no
      unacadena ← '0'
  fin_si
  devolver(unacadena)
fin_función
```

CONCEPTOS CLAVE

- Cadena.
- Cadena nula.
- Comparación de cadenas.
- Concatenación.
- Funciones de biblioteca.
- Literal de cadena.
- Longitud de la cadena.
- String.
- Variable de cadena.

RESUMEN

Cada lenguaje de computadora tiene su propio método de manipulación de cadenas de caracteres. Algunos lenguajes, tales como C++ y C, tienen un conjunto muy rico de funciones de manipulación de cadenas. Otros lenguajes, tales como FORTRAN, que se utilizan predominantemente para cálculos numéricos, incorporan características de manipulación de cadenas en sus últimas versiones. También lenguajes tales como LISP, que está concebido para manipular aplicaciones de listas proporciona capacidades excepcionales de manipulación de cadenas.

En un lenguaje como C o C++, las cadenas son simplemente *arrays* de caracteres terminados en caracteres nulos ("\0") que se pueden manipular utilizando técnicas estándares de procesamiento de *arrays* elemento por elemento. En esencia, las cadenas en los lenguajes de programación modernos tienen, fundamentalmente, estas características:

1. Una cadena (*string*) es un array de caracteres que en algunos casos (C++) se termina con el carácter `NULO (NULL)`.
2. Las cadenas se pueden procesar siempre utilizando técnicas estándares de procesamiento de arrays.
3. En la mayoría de los lenguajes de programación existen muchas funciones de biblioteca para procesamiento de cadenas como una unidad completa. Internamente estas funciones manipulan las cadenas carácter a carácter.
4. Algunos caracteres se escriben con un código de escape o secuencia de escape, que consta del carác-

ter escape (\) seguido por un código del propio carácter.
5. Un carácter se representa utilizando un único byte (8 bits). Los códigos de caracteres estándar más utilizados en los lenguajes de programación son **ASCII** y **Unicode**.
6. El código ASCII representa 127 caracteres y el código ASCII ampliado representa 256 caracteres. Mediante el código Unicode se llegan a representar numerosos lenguajes internacionales, además del inglés, como el español, francés, chino, hindi, alemán, etc.
7. Las bibliotecas estándar de funciones incorporadas a los lenguajes de programación incluyen gran cantidad de funciones integradas que manipulan cadenas y que actúan de modo similar a los algoritmos de las funciones explicadas en el capítulo. Este es el caso de la biblioteca de cadenas del lenguaje C o la biblioteca *string.h* de C++.
8. Algunas de la funciones de cadena típicas son: *longitud de la cadena*, *comparar cadenas*, *insertar cadena*, *copiar cadenas*, *concatenar cadenas*, etc.
9. El lenguaje C++ soporta las cadenas como arrays de caracteres terminado en el carácter nulo representado por la secuencia de escape "\0".
10. Los lenguajes orientados a objetos Java y C# soportan las cadenas como objetos de la clase `String`.

EJERCICIOS

8.1. Escribir un algoritmo para determinar si una cadena especificada ocurre en una cadena dada, y si es así, escribir un asterisco (*) en la primera posición de cada ocurrencia.

8.2. Escribir un algoritmo para contar el número de ocurrencias de cada una de las palabras `'a'`, `'an'` y `'and'` en las diferentes líneas de texto.

8.3. Contar el número de ocurrencias de una cadena especificada en diferentes líneas de texto.

8.4. Escribir un algoritmo que permita la entrada de un nombre consistente en un nombre, un primer apellido y un segundo apellido, en ese orden, y que imprima a continuación el último apellido, seguido del primer apellido y el nombre. Por ejemplo: `Luis Garcia Garcia` producirá: `Garcia Garcia Luis`.

8.5. Escribir un algoritmo que elimine todos los espacios finales en una cadena determinada. Por ejemplo: `'J. R. GARCIA '` se deberá transformar en `'J. R. GARCIA'`.

8.6. Diseñar un algoritmo cuya entrada sea una cadena `S` y un factor de multiplicación `N`, cuya función sea generar la cadena dada `N` veces. Por ejemplo:

'¡Hey!', 3

se convertirá en

```
'¡Hey! ¡Hey! ¡Hey!'
```

8.7. Diseñar un algoritmo que elimine todas las ocurrencias de cada carácter en una cadena dada a partir de otra cadena dada. Las dos cadenas son:

- `CADENA1` es la cadena donde deben eliminarse caracteres.
- `LISTA` es la cadena que proporciona los caracteres que deben eliminarse.

```
CADENA = 'EL EZNZZXTX'
LISTA = 'XZ'
```

la cadena pedida es `'EL ENT'`.

8.8. Escribir un algoritmo que convierta los números arábigos en romanos y viceversa (I = 1, V = 5, X = 10, L = 50, C = 100, D = 500 y M = 1000).

8.9. Diseñar un algoritmo que mediante una función permita cambiar un número n en base 10 a la base b, siendo b un número entre 2 y 20.

8.10. Escribir el algoritmo de una función que convierta una cadena en mayúsculas y otra que la convierta en minúsculas.

8.11. Diseñar una función que informe si una cadena es un palíndromo (una cadena es un palíndromo si se lee igual de izquierda a derecha que de derecha a izquierda).