El presente documento es parte del material de apoyo para el curso de Estructuras de Datos. Contiene un resumen o anotaciones sobre el contenido de las lecciones y está siendo revisado y editado regularmente. El documento es para uso exclusivo de los estudiantes del curso.

Almacenamiento y codificación de texto

En una computadora digital, toda la información está almacenada por medio de *bits* (dígitos binarios), ya sea en memoria principal o en un dispositivo de almacenamiento externo, como un disco, una unidad de estado sólido o un dispositivo USB.

Un número o un carácter cualquiera debe traducirse a *bits* para ser almacenado. Por ejemplo, un número como 17 puede ser traducido fácilmente usando su representación binaria, de la siguiente manera:

$$17_{10} \equiv 10001_2$$

y almacenarse en un *byte*:

Un *byte* es usualmente un grupo de 8 *bits* tratado como una unidad mínima de almacenamiento. En el diagrama anterior, cada bit está numerado desde la izquierda y se refiere con la potencia de 2 correspondiente.

De esta manera,

$$17_{10} = 2^4 + 2^0 = 16 + 1$$

Observe que la designación de los bits en la expresión anterior es completamente arbitraria. En este caso, se numeran para facilitar la conversión del valor representado a decimal.

Para representar texto, simplemente se hace un mapeo entre los diferentes símbolos y un valor numérico seleccionado a conveniencia. Esto se pueda hacer también para cualquier conjunto discreto de valores.

Un ejemplo del uso de valores numéricos para representar texto es el código Hollerith, utilizado en las tarjetas perforadas. Es interesante notar que este código fue ideado en 1888, muchos años antes de que fueran inventadas las computadoras digitales (Punched card, 2022).

Con el desarrollo de las computadoras y la necesidad de trabajar con conjuntos de caracteres más extensos, se desarrollaron otros sistemas de codificación, como el EBCDIC (*Extended Binary Coded Decimal Interchange Code*), desarrollado por IBM, y más tarde el código

ASCII (*American Standard Code for Information Interchange*). El código ASCII comenzó a desarrollarse a inicios de la década de 1960 con el fin de utilizarse en dispositivos de telecomunicación. Con la introducción de las computadoras personales a mediados de los años 70, el uso del código ASCII se generalizó y pasó a ser de aplicación estandarizada en sistemas informáticos.

Por su origen, el código ASCII no solamente incluía la representación de dígitos y símbolos alfabéticos, sino que consideraba un grupo de caracteres de control, no imprimibles, con el fin de señalar, por ejemplo, el inicio y el final del texto transmitido y otro tipo de información sobre la comunicación.

ASCII	Hex	Symbol	ASCII	Hex	Symbol	ASCII	Hex	Symbol	ASCII	Hex	Symbol
0	0	NUL	16	10	DLE	32	20	(space)	48	30	0
1	1	SOH	17	11	DC1	33	21	!	49	31	1
2	2	STX	18	12	DC2	34	22	"	50	32	2
3	3	ETX	19	13	DC3	35	23	#	51	33	3
4	4	EOT	20	14	DC4	36	24	$	52	34	4
5	5	ENQ	21	15	NAK	37	25	%	53	35	5
6	6	ACK	22	16	SYN	38	26	&	54	36	6
7	7	BEL	23	17	ETB	39	27	'	55	37	7
8	8	BS	24	18	CAN	40	28	(	56	38	8
9	9	TAB	25	19	EM	41	29	)	57	39	9
10	A	LF	26	1A	SUB	42	2A	*	58	3A	:
11	B	VT	27	1B	ESC	43	2B	+	59	3B	;
12	C	FF	28	1C	FS	44	2C	,	60	3C	<
13	D	CR	29	1D	GS	45	2D	-	61	3D	=
14	E	SO	30	1E	RS	46	2E	.	62	3E	>
15	F	SI	31	1F	US	47	2F	/	63	3F	?

ASCII	Hex	Symbol	ASCII	Hex	Symbol	ASCII	Hex	Symbol	ASCII	Hex	Symbol
64	40	@	80	50	P	96	60	`	112	70	p
65	41	A	81	51	Q	97	61	a	113	71	q
66	42	B	82	52	R	98	62	b	114	72	r
67	43	C	83	53	S	99	63	c	115	73	s
68	44	D	84	54	T	100	64	d	116	74	t
69	45	E	85	55	U	101	65	e	117	75	u
70	46	F	86	56	V	102	66	f	118	76	v
71	47	G	87	57	W	103	67	g	119	77	w
72	48	H	88	58	X	104	68	h	120	78	x
73	49	I	89	59	Y	105	69	i	121	79	y
74	4A	J	90	5A	Z	106	6A	j	122	7A	z
75	4B	K	91	5B	[	107	6B	k	123	7B	{
76	4C	L	92	5C	\	108	6C	l	124	7C	|
77	4D	M	93	5D	]	109	6D	m	125	7D	}
78	4E	N	94	5E	^	110	6E	n	126	7E	~
79	4F	O	95	5F	_	111	6F	o	127	7F	

Si se examina la representación en binario de cada uno de los valores en una tabla ASCII, se puede observar que los símbolos se encuentran agrupados de manera que su manejo o conversión sea bastante simple. Por ejemplo, cualquier código menor a 32 (20_{16}) es un carácter de control. Para convertir el código ASCII de un dígito a su valor numérico, basta con apagar dos bits en el código correspondiente.

Para convertir un carácter '4' en su valor numérico, por ejemplo:

'4' → 52_{10} → 34_{16} → 00**11** 0100

Se apagan los dos bits de la parte más significativa del *byte*, y tenemos:

00**00** 0100 → 04_{16} → 04_{10}

De la misma manera, es fácil convertir un carácter alfabético a un índice o viceversa con una simple operación, y también permite ordenar los caracteres alfabéticamente sencillamente comparando su valor numérico.

Una hilera de caracteres ASCII puede representarse entonces como una serie de bytes de acuerdo con el valor asociado a cada letra:

H	o	l	a
0100 1000	**0110 1111**	**0110 1100**	**0110 0001**

El código ASCII es un código de **7 *bits***, lo que significa que puede representar hasta 128 valores o símbolos diferentes: 32 caracteres de control y 96 caracteres imprimibles, o visibles. Si se usa un byte para representar un carácter, podrían codificarse hasta 256 símbolos distintos.

Diferentes sistemas operativos dieron un uso distinto a los 128 posibles caracteres con un '1' en el bit más significativo, utilizándolo para letras con marcas diacríticas, como tildes o diéresis, la 'ñ' en español o símbolos gráficos especiales.

En ASCII existe solamente un único esquema de codificación. Cada símbolo o carácter se representa por un valor numérico almacenado en un *byte*.

a ⟷ **97** ⟷ **0110 0001**

El ANSI (*American National Standards Institute*) estableció varios estándares para la codificación de texto en 8 *bits*, incluyendo los códigos ASCII existentes. El estándar ISO 8859-1 (conocido también como Latin-1), por ejemplo, contiene el conjunto de caracteres utilizado en América, Europa central y otros países, y es el código por defecto utilizado por la mayoría de los navegadores *web*. Cada conjunto de caracteres utilizado por diferentes sistemas operativos en diferentes países está asociado a un estándar ANSI particular.

La letra 'ñ', por ejemplo, tiene el código $F1_{16} = 241_{10}$ en el estándar ISO 8859-1. Sin embargo, la tabla de códigos ASCII utilizados tradicionalmente por las computadoras que utilizaban MS-DOS, asigna el valor $A4_{16} = 164_{10}$ al mismo símbolo. Este el código que algunos

usuarios se han acostumbrado a ingresar (empleado la tecla 'alt') cuando el teclado no permite digitarlo directamente. Si se lee un archivo escrito usando la codificación tradicional como si fuera un archivo codificado en ISO 8859-1, el símbolo puede mostrarse diferente, como '±'. Con la codificación ASCII tradicional, se pueden convertir letras de mayúscula a minúscula y viceversa con una simple operación de bits, pero este procedimiento no funciona con la 'ñ', ya que 'Ñ' se encuentra en la posición inmediata superior.

El esquema de codificación ANSI es suficiente en la mayoría de los casos, pero tiene limitaciones importantes. Por ejemplo, los símbolos 'a' y 'á' son diferentes, pero al comparar hileras de texto en español, deberían considerarse equivalentes, ya que la tilde es simplemente una marca ortográfica en la 'a'. En español, la 'n' y la 'ñ' son letras diferentes, pero en otros idiomas, 'ñ' simplemente es una 'n' con una virgulilla. Además, en español, la letra 'ñ' se encuentra entre la 'n' y la 'o', de manera que, si se hace un ordenamiento alfabético considerando solamente el código correspondiente, la posición de las letras sería incorrecta.

En algunos lenguajes de programación, como C++ la función `strlen()`, que calcula la longitud de una hilera, simplemente regresa la cantidad de *bytes* que dicha hilera ocupa en memoria, asumiendo que cada carácter puede ser representado en un *byte*. En algunos casos, esta propiedad puede ser útil para la implementación de algunas funciones de manejo de texto.

En general, los lenguajes de programación no hacen suposición alguna sobre la codificación del texto cuando trabajan con hileras o caracteres. Esto quiere decir que muchas operaciones simplemente asumen una cantidad fija de memoria para cada carácter y manipulan el valor numérico correspondiente para hacer operaciones de comparación o conversión.

Al utilizar diferentes codificaciones (ANSI), se pueden construir tablas de ordenamiento, que indiquen cuál es el valor que debe utilizarse para cada carácter, con el fin de poder hacer comparaciones correctamente.

De esta manera, puede existir una función `c()` tal que:

```
… < c('n') < c('ñ') < c('o') < …
c('a') = c('á') = c('à') = c('ä') = c('ã') = …
```

El uso de esta función permitiría hacer comparaciones de texto y ordenar hileras alfabéticamente (de acuerdo con un ordenamiento establecido para cada alfabeto).

Sin embargo, existen muchos alfabetos diferentes en cientos de idiomas y símbolos de uso especial que no pueden representarse fácilmente utilizando un solo *byte*. Y pueden existir aplicaciones que requieran manejar textos en diferentes alfabetos de manera simultánea.

UNICODE

UNICODE fue creado para estandarizar la representación de texto considerando diferentes lenguas y alfabetos.

Aunque el término "**carácter**" es de uso común, su significado es ambiguo. En su lugar, se usa el término "**grafema**" que representa una unidad indivisible de un sistema de escritura humana. Los grafemas incluyen letras y símbolos diacríticos (signos ortográficos o de acentuación), como las tildes y las diéresis. De la misma manera, se define el término "**fonema**", para referirse a una unidad sonora para diferenciar una palabra de otra en un lenguaje dado. Un fonema, es la unidad mínima de articulación cuyo sonido puede ser vocálico o consonántico. La unión o concatenación de los fonemas produce **morfemas**. La unión de diferentes morfemas da origen a las palabras. En lenguaje escrito, las palabras están formadas por grafemas.

Para representar un grafema en UNICODE, se utilizan uno o varios **puntos de código** (***code points***). Un grafema puede tener un único punto de código o varios en combinación (para incluir marcas ortográficas, por ejemplo). Mientras que en ASCII (o ANSI) sólo existe una estrategia de codificación. En UNICODE, existen varias estrategias.

UCS (***Universal Coded Character Set***, o simplemente *Universal Character Set*) es un estándar de codificación de caracteres desarrollado desde inicios de los años 90 para incluir grafemas y símbolos de diferentes lenguajes y alfabetos. UCS y **UNICODE** son diferentes estándares para definir conjuntos de caracteres que han sido desarrollados en conjunto. Generalmente, es más común hacer referencia al estándar UNICODE.

En cualquier caso, cada grafema (símbolo o carácter) tiene un punto de código que lo identifica. Cada punto de código es simplemente un valor numérico asociado al grafema. Un grafema puede tener un único punto de código o resultar de la combinación de varios puntos de código.

Un **esquema de codificación** (*encoding scheme*) es una forma de representar los diferentes puntos de código en una serie de *bytes*. Cada esquema de codificación tiene ventajas y desventajas. La diferencia más significativa entre los diferentes esquemas es el uso de una cantidad fija o variable de *bytes* para cada punto de código.

Varios esquemas tienen el prefijo **UTF** (***Unicode Transformation Format***) y un número que se refiere al número de bits usado para cada punto de código.

UTF-32

Este esquema de codificación usa un tamaño fijo de 32 *bits* (4 *bytes*) para cada punto de código. Aunque usar un tamaño fijo es una ventaja para la implementación de varias funciones de procesamiento de texto, el esquema tiende a desperdiciar mucho espacio, ya que ignora el hecho de que algunos símbolos son de uso más frecuente que otros.

	Hola, mundo!
ASCII	48 6F 6C 61 2C 20 6D 75 6E 64 6F 21
UTF-32	00 00 00 48 00 00 00 6F 00 00 00 6C 00 00 00 61 00 00 00 2C 00 00 00 20 00 00 00 6D 00 00 00 75 00 00 00 6E 00 00 00 64 00 00 00 6F 00 00 00 21

En UNICODE, los caracteres ASCII tienen un punto de código que es el mismo de la codificación original, por lo que los bits más significativos siempre son cero.

UTF-16

Es un esquema similar al anterior, pero que utiliza 16 *bits* en lugar de 32 para cada símbolo o grafema. UTF-16 puede utilizar 2 o 4 *bytes* para representar cada símbolo. Una diferencia importante entre UTF-16 y el esquema UCS-2 (*Universal Character Set* de 2 *bytes*) es que el último usa siempre un tamaño fijo de 2 *bytes*. Un inconveniente de UTF-32 y UTF-16 es que no son directamente compatibles con ASCII, como en el ejemplo anterior. Esto quiere decir que una **secuencia de *bytes*** UTF-32 (o UTF-16) no es igual a su equivalente en ASCII.

UTF-8

Es el esquema de codificación más popular. Utiliza de 1 hasta 4 *bytes* para cada símbolo, empleando solamente un *byte* para los puntos de código con el valor más bajo. Puntos de código con un valor más alto pueden ocupar 2 o más *bytes*.

La principal ventaja de este esquema de codificación es que los caracteres ASCII pueden ser representados exactamente de la misma manera, utilizando un *byte* cada uno.

El *bit* más significativo permite saber si cada punto de código está codificado en más de un *byte*.

Considere el siguiente ejemplo, para el texto **“pingües ñúes”**:

Grafema (carácter, símbolo)	**Punto de código**	
	Dec	Hex
e	101	0065
g	103	0067
i	105	0069
n	110	006E
ñ	241	00F1
p	112	0070
s	115	0073
ú	250	00FA
ü	252	00FC

La hilera puede codificarse de varias maneras:

	pingües ñúes
ASCII (CP437)	70 69 6E 67 81 65 73 20 A4 A3 65 73
ANSI (ISO 8859-1)	70 69 6E 67 FC 65 73 20 F1 FA 65 73
UTF-8	70 69 6E 67 C3 BC 65 73 20 C3 B1 C3 BA 65 73
UTF-16	FE FF 00 70 00 69 00 6E 00 67 00 FC 00 65 00 73 00 20 00 F1 00 FA 00 65 00 73

El texto ASCII corresponde con el código ASCII extendido (*extended ASCII*) empleado por las computadoras IBM-PC (y eventualmente todas las computadoras compatibles soportadas por la versión occidental y europea de MS-DOS). En otros sistemas operativos o en versiones de MS-DOS utilizadas en otras regiones, algunos caracteres podrían usar un código diferente o no existir del todo.

El texto ANSI corresponde con la codificación ISO 8859-1, que es la codificación por defecto utilizada por la mayoría de los *browsers*. En Windows, antes del uso generalizado de UNICODE, se usaba la codificación Windows-1252, que es similar y a menudo se confunde con ISO-8859-1 (Comparing Characters in Windows-1252, ISO-8859-1, ISO-8859-15, 2022).

Observe que, en este ejemplo, los códigos de los caracteres ANSI son los mismos valores que los puntos de código correspondientes en UNICODE.

Conversión de puntos de Código a UTF-8					
Rango de puntos de código		**byte 1**	**byte 2**	**byte 3**	**byte 4**
U+0000	U+007F	0xxxxxxx			
U+0080	U+07FF	110xxxxx	10xxxxxx		
U+0800	U+FFFF	1110xxxx	10xxxxxx	10xxxxxx	
U+10000	U+10FFFF	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

El lenguaje de programación Java representa las hileras internamente con UTF-16, lo que facilita el recorrido y la búsqueda dentro de las hileras en memoria.

Bibliografía

ASCII. (18 de agosto de 2022). Obtenido de Wikipedia: https://en.wikipedia.org/wiki/ASCII

ASCII table. (18 de agosto de 2022). Obtenido de theasciicode.com.ar: https://theasciicode.com.ar/

Bedell, C. (18 de agosto de 2022). *What are big-endian and little-endian?* Obtenido de TechTarget: https://www.techtarget.com/searchnetworking/definition/big-endian-and-little-endian#:~:text=Big%2Dendian%20is%20an%20order,the%20sequence)%20is%20stored%20first.

Character encoding. (22 de agosto de 2022). Obtenido de Wikipedia: https://en.wikipedia.org/wiki/Character_encoding

Code page 437. (24 de agosto de 2022). Obtenido de Wikipedia: https://en.wikipedia.org/wiki/Code_page_437

Comparing Characters in Windows-1252, ISO-8859-1, ISO-8859-15. (24 de agosto de 2022). Obtenido de i18nqa: https://www.i18nqa.com/debug/table-iso8859-1-vs-windows-1252.html#:~:text=ISO%2D8859%2D1%20(also,assigned%20to%20these%20code%20points.

GeeksforGeeks. (18 de agosto de 2022). *Little and Big Endian Mystery*. Obtenido de https://www.geeksforgeeks.org/little-and-big-endian-mystery/

HTML Unicode (UTF-8) Reference. (22 de agosto de 2022). Obtenido de w3schools: https://www.w3schools.com/charsets/ref_html_utf8.asp

ISO/IEC 8859-1. (18 de agosto de 2022). Obtenido de Wikipedia: https://en.wikipedia.org/wiki/ISO/IEC_8859-1

Jorquera, Z. (18 de agosto de 2022). *What Is Little-Endian And Big-Endian Byte Ordering?* Obtenido de Section: https://www.section.io/engineering-education/what-is-little-endian-and-big-endian/

Larson, Q. (24 de agosto de 2022). *What is UTF-8? UTF-8 Character Encoding Tutorial*. Obtenido de freeCodeCamp: https://www.freecodecamp.org/news/what-is-utf-8-character-encoding/

Punched card. (18 de agosto de 2022). Obtenido de Wikipedia: https://en.wikipedia.org/wiki/Punched_card

The EBCDIC character set. (18 de agosto de 2022). Obtenido de IBM Documentation: https://www.ibm.com/docs/en/zos-basic-skills?topic=mainframe-ebcdic-character-set

Unicode Character "ñ" (U+00F1). (22 de agosto de 2022). Obtenido de compart: https://www.compart.com/en/unicode/U+00F1

Unicode, UTF8 & Character Sets: The Ultimate Guide. (22 de agosto de 2022). Obtenido de Smashing Magazine: https://www.smashingmagazine.com/2012/06/all-about-unicode-utf8-character-sets/

Universal Coded Character Set. (22 de agosto de 2022). Obtenido de Wikipedia: https://en.wikipedia.org/wiki/Universal_Coded_Character_Set

UTF-8. (24 de agosto de 2022). Obtenido de Wikipedia: https://en.wikipedia.org/wiki/UTF-8

What Is ANSI Format? (18 de agosto de 2022). Obtenido de Techwalla: https://www.techwalla.com/articles/what-is-ansi-format

Windows-1252. (24 de agosto de 2022). Obtenido de Wikipedia: https://en.wikipedia.org/wiki/Windows-1252

