

# Codificação de caracteres

Video desta aula

Internamente, um computador só armazena e processa bytes, números inteiros entre 0 e 255. Não é possível armazenar diretamente textos, imagens, sons ou qualquer outra informação que não sejam bytes.

Para armazenar informações mais complexas que os bytes, é necessário **codificar** as mesmas, ou seja, transformá-las em sequências de bytes. Esta página discute as técnicas usadas para transformar as **letras e símbolos de um texto** em bytes, para poder armazená-lo e tratá-lo em um computador.

A figura a seguir mostra as etapas do tratamento de uma letra A pelo computador: o hardware do teclado é responsável por converter a letra digitada em byte(s) na memória do computador; em seguida, o hardware do terminal converte esses bytes em uma representação gráfica na tela.



## Algumas definições

- **Caractere:** é um símbolo da linguagem (letra, dígito ou sinal). Exemplos: A i ç ë ¥ β χ 诶 ☞
- **Conjunto de caracteres** (*charset*): é o conjunto de todos os caracteres suportados por um sistema ou por um padrão de codificação. Exemplo: A-Z, a-z, 0-9, ! @ # \$ % \* ( ) ~ \_ - + = { } [ ] | \ / < . ,
- **Codificação** (*encoding*): é a tradução entre os caracteres e seus respectivos valores numéricos em bytes. Exemplo, A → 65.

Existem diversas codificações de caracteres; a seguir serão apresentadas as mais usuais.

## A codificação ASCII

A codificação de caracteres mais antiga ainda em amplo uso é a **ASCII** (*American Standard Code for Information Interchange*), criada nos anos 1960 a partir de códigos de telegrafia. Sua última atualização ocorreu em 1986, mesmo assim é considerada uma codificação universal.



Praticamente **TODOS** os sistemas computacionais suportam ASCII !

A codificação ASCII abrange o conjunto de caracteres da língua inglesa, sinais gráficos e alguns caracteres de controle (nova linha, tabulação, etc), num total de 128 caracteres. Cada caractere é codificado em um byte, mas ocupa somente 7 bits; o oitavo bit de cada byte era antigamente usado para verificação de paridade.

A codificação ASCII é definida através da famosa **Tabela ASCII**, que é dividida em duas partes:

- 0 - 31 e 127: caracteres de controle (*newline, form feed, tab*, etc), que dependem do terminal utilizado.
- 32 - 126: caracteres imprimíveis (A, B, C, ...), independentes de terminal.

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	<b>NUL</b> (null)	32	20	040	&#32;	Space	64	40	100	&#64;	@	96	60	140	&#96;	`
1	1	001	<b>SOH</b> (start of heading)	33	21	041	&#33;	!	65	41	101	&#65;	A	97	61	141	&#97;	a
2	2	002	<b>STX</b> (start of text)	34	22	042	&#34;	"	66	42	102	&#66;	B	98	62	142	&#98;	b
3	3	003	<b>ETX</b> (end of text)	35	23	043	&#35;	#	67	43	103	&#67;	C	99	63	143	&#99;	c
4	4	004	<b>EOT</b> (end of transmission)	36	24	044	&#36;	\$	68	44	104	&#68;	D	100	64	144	&#100;	d
5	5	005	<b>ENQ</b> (enquiry)	37	25	045	&#37;	%	69	45	105	&#69;	E	101	65	145	&#101;	e
6	6	006	<b>ACK</b> (acknowledge)	38	26	046	&#38;	&	70	46	106	&#70;	F	102	66	146	&#102;	f
7	7	007	<b>BEL</b> (bell)	39	27	047	&#39;	'	71	47	107	&#71;	G	103	67	147	&#103;	g
8	8	010	<b>BS</b> (backspace)	40	28	050	&#40;	(	72	48	110	&#72;	H	104	68	150	&#104;	h
9	9	011	<b>TAB</b> (horizontal tab)	41	29	051	&#41;	)	73	49	111	&#73;	I	105	69	151	&#105;	i
10	A	012	<b>LF</b> (NL line feed, new line)	42	2A	052	&#42;	*	74	4A	112	&#74;	J	106	6A	152	&#106;	j
11	B	013	<b>VT</b> (vertical tab)	43	2B	053	&#43;	+	75	4B	113	&#75;	K	107	6B	153	&#107;	k
12	C	014	<b>FF</b> (NP form feed, new page)	44	2C	054	&#44;	,	76	4C	114	&#76;	L	108	6C	154	&#108;	l
13	D	015	<b>CR</b> (carriage return)	45	2D	055	&#45;	-	77	4D	115	&#77;	M	109	6D	155	&#109;	m
14	E	016	<b>SO</b> (shift out)	46	2E	056	&#46;	.	78	4E	116	&#78;	N	110	6E	156	&#110;	n
15	F	017	<b>SI</b> (shift in)	47	2F	057	&#47;	/	79	4F	117	&#79;	O	111	6F	157	&#111;	o
16	10	020	<b>DLE</b> (data link escape)	48	30	060	&#48;	0	80	50	120	&#80;	P	112	70	160	&#112;	p
17	11	021	<b>DC1</b> (device control 1)	49	31	061	&#49;	1	81	51	121	&#81;	Q	113	71	161	&#113;	q
18	12	022	<b>DC2</b> (device control 2)	50	32	062	&#50;	2	82	52	122	&#82;	R	114	72	162	&#114;	r
19	13	023	<b>DC3</b> (device control 3)	51	33	063	&#51;	3	83	53	123	&#83;	S	115	73	163	&#115;	s
20	14	024	<b>DC4</b> (device control 4)	52	34	064	&#52;	4	84	54	124	&#84;	T	116	74	164	&#116;	t
21	15	025	<b>NAK</b> (negative acknowledge)	53	35	065	&#53;	5	85	55	125	&#85;	U	117	75	165	&#117;	u
22	16	026	<b>SYN</b> (synchronous idle)	54	36	066	&#54;	6	86	56	126	&#86;	V	118	76	166	&#118;	v
23	17	027	<b>ETB</b> (end of trans. block)	55	37	067	&#55;	7	87	57	127	&#87;	W	119	77	167	&#119;	w
24	18	030	<b>CAN</b> (cancel)	56	38	070	&#56;	8	88	58	130	&#88;	X	120	78	170	&#120;	x
25	19	031	<b>EM</b> (end of medium)	57	39	071	&#57;	9	89	59	131	&#89;	Y	121	79	171	&#121;	y
26	1A	032	<b>SUB</b> (substitute)	58	3A	072	&#58;	:	90	5A	132	&#90;	Z	122	7A	172	&#122;	z
27	1B	033	<b>ESC</b> (escape)	59	3B	073	&#59;	:	91	5B	133	&#91;	[	123	7B	173	&#123;	{
28	1C	034	<b>FS</b> (file separator)	60	3C	074	&#60;	<	92	5C	134	&#92;	\	124	7C	174	&#124;	
29	1D	035	<b>GS</b> (group separator)	61	3D	075	&#61;	=	93	5D	135	&#93;	]	125	7D	175	&#125;	}
30	1E	036	<b>RS</b> (record separator)	62	3E	076	&#62;	>	94	5E	136	&#94;	^	126	7E	176	&#126;	~
31	1F	037	<b>US</b> (unit separator)	63	3F	077	&#63;	?	95	5F	137	&#95;	_	127	7F	177	&#127;	DEL

Source: [www.LookupTables.com](http://www.LookupTables.com)

A codificação ASCII ainda é amplamente usada para codificação de textos puros em inglês, como códigos-fonte de programas, páginas HTML, arquivos de configuração, etc.

## Code pages

A codificação ASCII não suporta caracteres acentuados (á é ñ ë) ou caracteres específicos de outras línguas, como ç ¥ ß γ 谡 etc. Pode-se usar o oitavo bit de cada byte para associar caracteres aos valores acima de 127. Isso levou à criação de diversas **tabelas ASCII estendidas** para definir os símbolos de 128 a 255. Cada codificação é denominada uma **code page**; algumas das mais conhecidas são:

- **CP-437**: (code page 437), codificação usada nos primeiros PCs, com caracteres acentuados e gráficos simples (☘ ☙ ☚ ☛).
- **Windows-1252**: codificação usada em sistemas Windows mais antigos; é parte de um conjunto de codificações para diversas linguagens chamado **Windows code pages**.
- **KOI8-R**: cirílico russo (Код Обмена Информацией, 8 бит).
- **BraSCII**: português brasileiro, usada nos anos 1980-90.
- **ISO-8859**: codificações da ISO para diversas línguas.

## Codificações ISO-8859

Nos anos 1980, para tentar organizar a profusão de codepages ASCII estendidas, a ISO propôs o conjunto de padrões **ISO-8859**, que define codificações ASCII estendidas para diversas linguagens, como por exemplo:

- [ISO-8859-1](#): Europa ocidental (francês, espanhol, italiano, alemão, etc)
- [ISO-8859-15](#): revisão do ISO-8859-1, contendo o € e outros símbolos
- [ISO-8859-2](#): Europa central (Bósnio, Polonês, Croata, etc)
- [ISO-8859-6](#): árabe simplificado
- [ISO-8859-7](#): grego

As codificações ISO-8859 se tornaram um padrão mundial e ainda são amplamente usadas em muitos sistemas, sendo gradualmente substituída pela codificação Unicode em sistemas mais recentes. Elas são compatíveis com a codificação ASCII, pois representam cada caractere com **somente um byte** e respeitam as definições ASCII dos caracteres de 0 a 127.



Programas que manipulem caracteres ISO devem usar variáveis `unsigned char`, para poder representar valores de 0 a 255.

## Caracteres multibyte

O maior problema das codificações ISO-8859 é o uso de somente **um byte por caractere**, o que limita cada *code page* a 256 caracteres. Essa limitação impede a representação completa de línguas asiáticas e do árabe, por exemplo.

Para representar conjuntos com mais de 256 caracteres é necessário usar **caracteres multibyte**, ou seja, com mais de um byte. Por exemplo, se usarmos 2 bytes por caractere é possível representar até  $2^{16} = 65.536$  caracteres distintos na mesma tabela, sem precisar trocar de *code page*.

Vários padrões de codificação multibyte foram propostos, como:

- [ISO-2022-CJK](#): chinês, japonês, coreano
- [Shift-JIS](#): japonês (Windows)
- [GB 18030](#): padrão oficial chinês
- [Big5](#): chinês tradicional (Taiwan)
- **Unicode**

Alguns destes padrões definem todos os caracteres com uma quantidade fixa de bits (16 ou 32), enquanto outros definem caracteres com tamanho variável (8, 16 ou 32 bits).

## Unicode

O padrão [Unicode](#) define um imenso conjunto de caracteres e os modos de codificação dos mesmos. Atualmente, existem cerca de [140.000 caracteres](#) definidos em Unicode, para todas as línguas conhecidas (inclusive [Klingon!](#)), além de símbolos e emojis. Eles ocupam pouco mais de 10% da capacidade total desse padrão.

Em Unicode, cada caractere possui um código numérico único, chamado *code point*, que pode ser representado de diversas formas. Por exemplo, o *code point* do emoji 🦁 vale 128540 (1F61C<sub>h</sub>) e pode ser representado como:

- `U+1f61c` : em hexadecimal
- `&#x1f61c;` ou `&#128540;` : em páginas Web (decimal ou hexadecimal)
- `\u1f61c` : em algumas linguagens de programação

Caracteres em Unicode podem ser codificados (representados em bytes) de diversas formas:

- **UTF-8**: *8-bit Unicode Transformation Format*, usa de 1 a 4 bytes por caractere. É usado no Linux, Windows

10 e outros sistemas recentes.

- **UTF-16**: usa 2 ou 4 bytes por caractere; muito usado nas APIs dos sistemas Windows, em Java, Python e PHP.
- **UTF-32**: usa sempre 4 bytes por caractere. É pouco usado na prática.

## A codificação UTF-8

UTF-8 é certamente a codificação multibyte **mais utilizada hoje em dia**, por ser plenamente compatível com a codificação ASCII e por ser econômica em espaço.

Em UTF-8, cada caractere Unicode é codificado usando de 1 a 4 bytes, conforme o número de bits de seu *code point*:

Caractere	Code point	Em binário	bits
A	41 <sub>h</sub> (65)	100 0001	7
ç	E7 <sub>h</sub>	1110 0111	8
©	C2A9 <sub>h</sub>	1100 0010 1010 1001	16
☐	1F600 <sub>h</sub>	1 1111 0110 0000 0000	17

A regra de codificação de cada caractere é escolhida conforme o número de bits usados pelo seu *code point*:

# de bits do caractere	formato codificado	bytes	Uso
até 7 bits	0xxx - xxxx	1	tabela ASCII
8-11 bits	110x - xxxx 10xx - xxxx	2	caracteres estendidos
12-16 bits	1110 - xxxx 10xx - xxxx 10xx - xxxx	3	caracteres estendidos
17-21 bits	1111 - 0xxx 10xx - xxxx 10xx - xxxx 10xx - xxxx	4	caracteres estendidos

Pode-se observar que bytes os bytes iniciando em 0... sempre representam caracteres ASCII. Então, um texto codificado em UTF-8 contendo somente caracteres com códigos entre 0 e 127 equivale a um texto codificado em ASCII padrão.

Além disso, todos os bytes iniciando em 10... são bytes de continuação da codificação de um caractere multibyte. Isso significa que é fácil localizar o início de cada caractere no texto, mesmo na presença de erros.



**Dica:** pode-se visualizar o conteúdo de um arquivo em hexadecimal ou binário usando o comando `xxd`.

O mecanismo de codificação de *code points* Unicode em UTF-8 funciona da seguinte forma:

1. Dado um caractere, verifica-se quantos bits são necessários para armazenar seu código em UTF-8. Por exemplo, o caractere ☐ (*code point* U+1f600) precisa de 17 bits: 1F600 → 1 1111 0110 0000 0000.
2. Para 17 bits é necessário codificar usando 4 bytes (faixa 17-21 bits)
3. Distribui-se os bits do código numérico do caractere nos espaços disponíveis:

Code point (hex)	1	f	6	0	0		
Code point (bin)	0001	1111	0110	0000	0000		
Encoding format	1111-0xxx	10xx-xxxx	10xx-xxxx	10xx-xxxx			
Code point (bin)	000	01 1111	01 1000	00 0000			
Encoded character	1111 0000	1001 1111	1001 1000	1000 0000			
	f	0	9	f	9	8	8

4. Com isso, a codificação de U+1f600 em UTF-8 resulta nos 4 bytes `f0 9f 98 80`.
5. A decodificação (de UTF-8 para o *code point*) se efetua fazendo o caminho inverso.

Alguns arquivos codificados em UTF-\* podem apresentar em seus dois primeiros bytes um valor chamado BOM (*Byte Order Mark*), que define em que ordem os bytes de cada caractere devem ser considerados: *big endian* ou *little endian*. o campo BOM não é necessário em UTF-8, mas pode estar presente às vezes:

Bytes	Encoding Form
00 00 FE FF	UTF-32, big-endian
FF FE 00 00	UTF-32, little-endian
FE FF	UTF-16, big-endian
FF FE	UTF-16, little-endian
EF BB BF	UTF-8



**Dica:** no Linux, pode-se digitar caracteres Unicode usando as seguintes teclas: `ctrl + shift + u`, código hexadecimal, `enter`

## Comparando as codificações

O quadro a seguir compara a representação da string “equação” usando algumas das codificações estudadas. No caso da codificação ASCII, considera-se a letra sem acento ou cedilha; a representação UTF-16 usa dois bytes de cabeçalho BOM (*Byte Order Mark*).

Codificação	BOM	e	q	u	a	ç	ã	o	
ASCII		65	71	75	61	63	61	6f	00
ISO-8859-1		65	71	75	61	e7	e3	6f	00
UTF-8		65	71	75	61	c3 a7	c3 a3	6f	00
UTF-16 (be)	fe ff	00 65	00 71	00 75	00 61	00 e7	00 e3	00 6f	00 00

## Conversão de codificações

O comando `file` do UNIX informa o tipo de codificação usado em um arquivo de texto:

```
$ file exemplo.*
exemplo.c:      C source, ISO-8859 text
exemplo.html:  HTML document, ASCII text
exemplo.txt:   UTF-8 Unicode text
```

A conversão de codificação de um arquivo de texto pode ser feita com utilitários específicos, como o `iconv` no Linux:

```
iconv -f ISO-8859-15 -t UTF-8 < input.txt > output.txt
```

Além disso, os editores de texto geralmente permitem escolher a codificação ao salvar o arquivo. Por exemplo, no VI:

```
:set fileencoding=utf8
:w myfilename
```

## Mais informações

Sobre codificações e Unicode:

- <https://www.cl.cam.ac.uk/~mgk25/unicode.html>
- <https://www.ime.usp.br/~pf/algoritmos/apend/unicode.html>
- <https://www.cprogramming.com/tutorial/unicode.html>
- <https://begriffs.com/posts/2019-05-23-unicode-icu.html>
- <http://kunststube.net/encoding/>

## Exercícios

1. Use o programas `file` e `iconv` para fazer as seguintes conversões:
  1. o arquivo `exemplo.c` para UTF-8
  2. o arquivo `exemplo.c` para ASCII

From:

<https://wiki.inf.ufpr.br/maziero/> - **Prof. Carlos Maziero**

Permanent link:

[https://wiki.inf.ufpr.br/maziero/doku.php?id=c:codificacao\\_de\\_caracteres](https://wiki.inf.ufpr.br/maziero/doku.php?id=c:codificacao_de_caracteres)

Last update: **2023/08/01 20:13**

