Character Encoding

Zdeněk Žabokrtský, Rudolf Rosa

October 14, 2021

EUROPEAN UNION
European Structural and Investment Fund
Operational Programme Research,
Development and Education

Charles University
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics

Hello world

01001000 01100101 01101100 01101100 01101111 00100000 01010111 01101111 01110010 01101100 01100100

Exercise

- Recall the binary and hexadecimal system and convert first few binary octets to their hexadecimal representation.

Introduction

Outline

- ASCII
- 8-bit extensions
- Unicode
- and some related topics:
  - end of line
  - byte-order mark
  - alternative solution to character encoding – escaping

Problem statement

- Today's computers use binary digits
- No natural relation between numbers and characters of an alphabet $\implies$ convention needed
- No convention $\implies$ chaos
- Too many conventions $\implies$ chaos
- (recall A. S. Tanenbaum: *The nice thing about standards is that you have so many to choose from.*)

Btw why binary computers?

- the answer is actually not that straightforward
- three distinct voltage values certainly possible too (experimental ternary computers assembled in 1950s), as well as any other higher base
- mostly physical and electro-technical reasons, rather than mathematical or information-theoretical reasons:
  - two voltage values technically very easy to distinguish – basically just charge vs. no charge opposition on a transistor's gate (and hence it is fast)
  - only very simple circuitry needed for two-valued logic (and hence it is fast)

Basic notions – Character

a character

- is an abstract notion, not something tangible
- has no numerical representation nor graphical form
- e.g. “capital A with grave accent”
- you need an encoding to associate a character with a numerical representation
- you need a font to associate a character with a concrete visual realization

Basic notions – Character set

a character set (or a character repertoire)

- a set of logically distinct characters
- relevant for a certain purpose (e.g., used in a given language or in group of languages)
- not neccessarily related to computers

a coded character set:

- a unique number (typically non-negative integer) assigned to each character: code point
- relevant for a certain purpose (e.g., used in a given language or in group of languages)
- not neccessarily related to computers

Basic notions – Glyph and Font

- a glyph – a visual representation of a character
- a font – a set of glyphs of characters

Basic notions – Character encoding

character encoding

- the way how (coded) characters are mapped to (sequences of) bytes
- both in the declarative and procedural sense
    - a conversion table
    - a conversion process

8-bit encodings

ASCII

- In the beginning there was the Word. And the Word was encoded in 7-bit ASCII. (well, if we ignore the history before 1950's)

ASCII

- ASCII = American Standard Code for Information Interchange (1963)
  - 7 bits (0–127)
  - 33 control characters (0–31,127) such as Escape, Line Feed, Bell
  - the remaining 95 characters (32–126): printable characters such as space, numerals, upper and lower case characters.

ASCII Code Chart

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

ASCII, cont.

- now with decimal and octal codes (credit: www.pragimtech.com)

ASCII Table

Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
0	0	0		32	20	40	[space]	64	40	100	@	96	60	140	`
1	1	1		33	21	41	!	65	41	101	A	97	61	141	a
2	2	2		34	22	42	"	66	42	102	B	98	62	142	b
3	3	3		35	23	43	#	67	43	103	C	99	63	143	c
4	4	4		36	24	44	$	68	44	104	D	100	64	144	d
5	5	5		37	25	45	%	69	45	105	E	101	65	145	e
6	6	6		38	26	46	&	70	46	106	F	102	66	146	f
7	7	7		39	27	47	'	71	47	107	G	103	67	147	g
8	8	10		40	28	50	(	72	48	110	H	104	68	150	h
9	9	11		41	29	51	)	73	49	111	I	105	69	151	i
10	A	12		42	2A	52	*	74	4A	112	J	106	6A	152	j
11	B	13		43	2B	53	+	75	4B	113	K	107	6B	153	k
12	C	14		44	2C	54	,	76	4C	114	L	108	6C	154	l
13	D	15		45	2D	55	-	77	4D	115	M	109	6D	155	m
14	E	16		46	2E	56	.	78	4E	116	N	110	6E	156	n
15	F	17		47	2F	57	/	79	4F	117	O	111	6F	157	o
16	10	20		48	30	60	0	80	50	120	P	112	70	160	p
17	11	21		49	31	61	1	81	51	121	Q	113	71	161	q
18	12	22		50	32	62	2	82	52	122	R	114	72	162	r
19	13	23		51	33	63	3	83	53	123	S	115	73	163	s
20	14	24		52	34	64	4	84	54	124	T	116	74	164	t
21	15	25		53	35	65	5	85	55	125	U	117	75	165	u
22	16	26		54	36	66	6	86	56	126	V	118	76	166	v
23	17	27		55	37	67	7	87	57	127	W	119	77	167	w
24	18	30		56	38	70	8	88	58	130	X	120	78	170	x
25	19	31		57	39	71	9	89	59	131	Y	121	79	171	y
26	1A	32		58	3A	72	:	90	5A	132	Z	122	7A	172	z
27	1B	33		59	3B	73	;	91	5B	133	[	123	7B	173	{
28	1C	34		60	3C	74	<	92	5C	134	\	124	7C	174	|
29	1D	35		61	3D	75	=	93	5D	135	]	125	7D	175	}
30	1E	36		62	3E	76	>	94	5E	136	^	126	7E	176	~
31	1F	37		63	3F	77	?	95	5F	137	_	127	7F	177	

Exercise

Given that A's code point in ASCII is 65, and a's code point is 97.

- What is the binary representation of 'A' in ASCII? (and what's its hexadecimal representation)
- What is the binary representation of 'a' in ASCII?

Is it clear now why there are the special characters inserted between upper and lower case letters?

ASCII, cont.

- ASCII's main advantage – simplicity: one character – one byte
- ASCII's main disadvantage – no way to represent national alphabets
- Anyway, ASCII is one of the most successful software standards ever developed!

How to represent the end of line

- “newline” == “end of line” == “EOL”
- ASCII symbols LF (line feed, 0x0A) and/or CR (carriage return, 0x0D), depending on the operation system:
  - LF is used in UNIX systems
  - CR+LF used in Microsoft Windows
  - CR used in Mac OS

A "how-many" question

- ASCII is clearly not enough for Czech
- but how many additional characters do we actually need for Czech?

Another "how-many" question

How many questions would be needed if we want to keep several languages in the same code space?

- find pieces of text from the following languages: Czech, French, German, Spanish, Greek, Icelandic, Russian (at least a few paras for each)
- store them into plain text files
- count how many different signs in total appear in the files
- try to solve it using only a bash command pipeline (hint: you may use e.g. '`grep -o .`' or `sed 's/./&\n/g'`)

8-bit encodings

- Supersets of ASCII, using octets 128–255 (still keeping the 1 character – 1 byte relation)
- International Standard Organisation: ISO 8859 (1980's)
- West European Languages: ISO 8859-1 (ISO Latin 1)
- For Czech and other Central/East European languages: anarchy
  - ISO 8859-2 (ISO Latin 2)
  - Windows 1250
  - KOI-8
  - Brothers Kamenický
  - other proprietary "standards" by IBM, Apple etc.

How to inspect the raw content of a file?

- The encoding of a text file must be known in order to display the text correctly.
- Is there an encoding-less way to view a file?
- Yes, you can view the hexadecimal codes of characters: `hexdump -C`

Unicode

Unicode

- The Unicode Consortium (1991)
- the Unicode standard defined as ISO 40646
- nowadays: all the world's living languages
- highly different writing systems: Arabic, Sanscrit, Chinese, Japanese, Korean
- ambition: 250 writing systems for hundreds of languages
- Unicode assigns each character a unique code point
- example: "LATIN CAPITAL LETTER A WITH ACUTE" goes to U+00C1
- Unicode defines a character set as well as several encodings

Common Unicode encodings

- UTF-32
  - 4 bytes for any character
- UTF-16
  - 2 bytes for each character in Basic Multilingual Plane
  - other characters 4 bytes
- UTF-8
  - 1-6 bytes per character

UTF-8 and ASCII

- a killer feature of UTF-8: an ASCII-encoded text is encoded in UTF-8 at the same time!
- the actual solution:
  - the number of leading 1's in the first byte determines the number of bytes in the following way:
    - zero ones (i.e., 0xxxxxxx): a single byte needed for the character (i.e., identical with ASCII)
    - two or more ones: the total number of bytes needed for the character
  - continuation bytes: 10xxxxxx
- a reasonable space-time trade-off
- but above all: this trick radically facilitated the spread of Unicode

UTF-8 and Czech

- We are lucky with Czech: characters of the Czech alphabet consume at most 2 bytes

Exercise: does this or that character exist in Unicode?

- check http://shapecatcher.com/

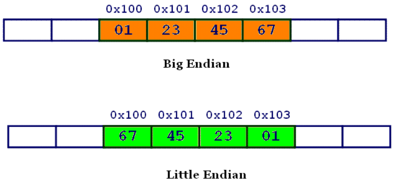
Misc

Byte order mark (BOM)

- BOM = a Unicode character: U+FEFF
- a special Unicode character, possibly located at the very beginning of a text stream
- optional
- used for several different purposes:
  - specifies byte order – endianess (little or big endian)
  - specifies (with a high level of confidence) that the text stream is encoded in one of the Unicode encodings
  - distinguishes Unicode encodings
- BOM in the individual encodings:
  - UTF-8: 0xEF,0xBB,0xBF
  - UTF-16: 0xFE followed by 0xFF for big endian, the other way round for little endian
  - UTF-32 – rarely used

If you can’t recall endianess

- Little and big endian are two ways of storing multibyte data-types (int, float, etc).
- In little endian machines, last byte of binary representation of the multibyte data-type is stored first.
- suppose an integer stored in 4 bytes:



CREDIT: https://www.geeksforgeeks.org/

Exercise

- using any text editor, store the Czech word *žlutý* into a text file in UTF-8
- using the `iconv` command, convert this file into four files corresponding the these encodings:
  - cp1250
  - iso-8859-2
  - utf-16
  - utf-32
- look at the size of these 5 files (using e.g. `ls * -l`) and explain all size differences
- use `hexdump` to show the hexadecimal (“encoding-less”) content of the files
- check out what the `file` command guesses

Exercise on character identity

- Create a UTF-8 encoded file containing the Latin letter "A", the Greek letter "A", and the Cyrilic letter "A", and view the file using `hexdump -C`.
- This might be a source of confusion when working with multilingual data.

Some myths and misunderstandings about character encoding

The following statements are wrong:

- ASCII is an 8-bit encoding.
- Unicode is a character encoding.
- Unicode can only support 65,536 characters.
- UTF-16 encodes all characters with 2 bytes.
- Case mappings are 1-1.
- This is just a plain text file, no encoding.
- This file is encoded in Unicode.
- It is the filesystem who knows the encoding of this file.
- File encoding can be absolutely reliably detected by this utility.

Detection of a file's encoding

100% accuracy impossible, but

- in some situations some encodings can be rejected with certainty
  - e.g. Unicode encodings do not allow some byte sequences
- if you have a prior knowledge (or expectation distribution) concerning the language of the text, then some encodings might be highly improbable
  - e.g. ISO-8859-1 improbable for Czech
- BOM can help too
- rule of thumb: many modern solutions default to UTF-8 if no encoding is specified
- the `file` command works reasonably well in most cases

Specification of a file's encoding – encoding declaration

- however, "reasonably well" is not enough, we need certainty
- for most plain-text-based file formats (including source codes of programming languages) there are clear rules how encodings should be specified
  - HTML4 vs HTML5

    ```
    <meta http-equiv="Content-Type" content="text/html;charset=ISO-8859-2">
    ```

    ```
    <meta charset="iso-8859-2">
    ```

    (btw notice the misnomer: "charset" stands for an encoding here, not for a character set (explain why))
  - XML

    ```
    <?xml version="1.0" encoding="UTF-8"?>
    ```

  - LaTeX

    ```
    \usepackage[utf8]{inputenc}
    ```

Encoding declaration, cont.

- some editors have their own encoding declaration style, such Emacs's

  ```
  # -*- coding: <encoding-name> -*-
  ```

  or VIM's

  ```
  # vim:fileencoding=<encoding-name>
  ```

Exercise

Try to fool the `file` command

- try to construct a file whose encoding is detected incorrectly by `file`

Summary

1. In spite of some relicts of chaos in the real world, the problem of character encoding has been solved almost exhaustively, esp. compared to the previous 8-bit solutions.
2. However, some new complexity has been induced (more or less inevitably), such as more a complex notion of character equivalence – Latin vs. Greek Vs. Cyrilic capital letter A.
3. Whenever possible, try to stick to Unicode (with UTF-8 being its prominent encoding).
4. Make sure you perfectly understand how Unicode is handled in your favourite programming languages and in your editors.