Character Encoding

Zdeněk Žabokrtský, Rudolf Rosa

September 8, 2018
Hello world

01001000 01100101 01101100 01101100 01101111 00100000 01010111 01101111 01110010
01101100 01100100
01101100 01100100
• ASCII
• 8-bit extensions
• Unicode
• and some related topics:
  • end of line
  • byte-order mark
  • alternative solution to character encoding – escaping
Exercise

a warm-up exercise:

- 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')
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.*)
a character

- an abstract (Platonic) entity
- no numerical representation nor graphical form
- e.g. “capital A with grave accent”
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 necessarily related to computers

a coded character set:
  • a unique number assigned to each character: code point
  • relevant for a certain purpose (e.g., used in a given language or in group of languages)
  • not necessarily related to computers
Basic notions – Glyph and Font

- a glyph – a visual representation of a character
- a font – a set of glyphs of characters
character encoding
  ▪ the way how (coded) characters are mapped to (sequences of) bytes
  ▪ both in the declarative and procedural sense
- At the beginning there was a word, and the word was encoded in 7-bit ASCII. (well, if we ignore the history before 1950’s)
- ASCII = American Standard Code for Information Interchange
  - 7 bits (0–127)
  - 0–31,127: control characters (Escape, Line Feed)
  - 32–126: space, numerals, upper and lower case characters
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’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!
Intermezzo 1: 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
- Supersets of ASCII, using octets 128–255 (still keeping the 1 character – 1 byte relation)
- 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.
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
- 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/
Intermezzo 2: 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
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
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.
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
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

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

  - \LaTeX

  ```latex
  \usepackage[utf8]{inputenc}
  ```
some editors have their own encoding declaration style, such Emacs’s

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

or VIM’s

```bash
# vim:fileencoding=<encoding-name>
```
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 inevitably, such as more a complex notion of character equivalence – Latin vs. Green 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.

https://ufal.cz/courses/npfl092