Character Encoding

Zdeněk Žabokrtský, Rudolf Rosa

February 18, 2023
Motivation

(credit: https://serpstat.com/blog/how-to-set-the-site-encoding-correctly/)
Hello world

01001000 01100101 01101100 01101100 01101111 00100000
01010111 01101111 01110010 01101100 01100100

01001000 01000101 01101100 01101100 01101111 00100000
01010111 01101111 01110010 01101100 01100100
• Recall the binary and hexadecimal system and convert first few binary octets to their hexadecimal representation.
Introduction
• 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
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 (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 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
  - a conversion table
  - a conversion process
8-bit encodings
• 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 = 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.
• now with decimal and octal codes (credit: www.pragimtech.com)

### ASCII Table

<table>
<thead>
<tr>
<th>Dec</th>
<th>Hex</th>
<th>Oct</th>
<th>Char</th>
<th>Dec</th>
<th>Hex</th>
<th>Oct</th>
<th>Char</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>[space]</td>
<td>32</td>
<td>20</td>
<td>40</td>
<td>@</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>!</td>
<td>64</td>
<td>40</td>
<td>100</td>
<td>96</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>*</td>
<td>66</td>
<td>42</td>
<td>102</td>
<td>98</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>#</td>
<td>67</td>
<td>43</td>
<td>103</td>
<td>99</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
<td>$</td>
<td>68</td>
<td>44</td>
<td>104</td>
<td>100</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>%</td>
<td>69</td>
<td>45</td>
<td>105</td>
<td>101</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>6</td>
<td>&amp;</td>
<td>70</td>
<td>46</td>
<td>106</td>
<td>102</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>7</td>
<td>'</td>
<td>71</td>
<td>47</td>
<td>107</td>
<td>103</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>10</td>
<td>(</td>
<td>72</td>
<td>48</td>
<td>110</td>
<td>104</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>11</td>
<td>)</td>
<td>73</td>
<td>49</td>
<td>111</td>
<td>105</td>
</tr>
<tr>
<td>10</td>
<td>A</td>
<td>12</td>
<td>#</td>
<td>74</td>
<td>4A</td>
<td>112</td>
<td>106</td>
</tr>
<tr>
<td>11</td>
<td>B</td>
<td>13</td>
<td>+</td>
<td>75</td>
<td>4B</td>
<td>113</td>
<td>107</td>
</tr>
<tr>
<td>12</td>
<td>C</td>
<td>14</td>
<td>=</td>
<td>76</td>
<td>4C</td>
<td>114</td>
<td>108</td>
</tr>
<tr>
<td>13</td>
<td>D</td>
<td>15</td>
<td>-</td>
<td>77</td>
<td>4D</td>
<td>115</td>
<td>109</td>
</tr>
<tr>
<td>14</td>
<td>E</td>
<td>16</td>
<td>/</td>
<td>78</td>
<td>4E</td>
<td>116</td>
<td>110</td>
</tr>
<tr>
<td>15</td>
<td>F</td>
<td>17</td>
<td>\</td>
<td>79</td>
<td>4F</td>
<td>117</td>
<td>111</td>
</tr>
<tr>
<td>16</td>
<td>10</td>
<td>20</td>
<td>0</td>
<td>80</td>
<td>50</td>
<td>120</td>
<td>112</td>
</tr>
<tr>
<td>17</td>
<td>11</td>
<td>21</td>
<td>1</td>
<td>81</td>
<td>51</td>
<td>121</td>
<td>113</td>
</tr>
<tr>
<td>18</td>
<td>12</td>
<td>22</td>
<td>2</td>
<td>82</td>
<td>52</td>
<td>122</td>
<td>114</td>
</tr>
<tr>
<td>19</td>
<td>13</td>
<td>23</td>
<td>3</td>
<td>83</td>
<td>53</td>
<td>123</td>
<td>115</td>
</tr>
<tr>
<td>20</td>
<td>14</td>
<td>24</td>
<td>4</td>
<td>84</td>
<td>54</td>
<td>124</td>
<td>116</td>
</tr>
<tr>
<td>21</td>
<td>15</td>
<td>25</td>
<td>5</td>
<td>85</td>
<td>55</td>
<td>125</td>
<td>117</td>
</tr>
<tr>
<td>22</td>
<td>16</td>
<td>26</td>
<td>6</td>
<td>86</td>
<td>56</td>
<td>126</td>
<td>118</td>
</tr>
<tr>
<td>23</td>
<td>17</td>
<td>27</td>
<td>7</td>
<td>87</td>
<td>57</td>
<td>127</td>
<td>119</td>
</tr>
<tr>
<td>24</td>
<td>18</td>
<td>30</td>
<td>8</td>
<td>88</td>
<td>58</td>
<td>130</td>
<td>120</td>
</tr>
<tr>
<td>25</td>
<td>19</td>
<td>31</td>
<td>9</td>
<td>89</td>
<td>59</td>
<td>131</td>
<td>121</td>
</tr>
<tr>
<td>26</td>
<td>1A</td>
<td>32</td>
<td>10</td>
<td>90</td>
<td>5A</td>
<td>132</td>
<td>122</td>
</tr>
<tr>
<td>27</td>
<td>1B</td>
<td>33</td>
<td>11</td>
<td>91</td>
<td>5B</td>
<td>133</td>
<td>123</td>
</tr>
<tr>
<td>28</td>
<td>1C</td>
<td>34</td>
<td>12</td>
<td>92</td>
<td>5C</td>
<td>134</td>
<td>124</td>
</tr>
<tr>
<td>29</td>
<td>1D</td>
<td>35</td>
<td>13</td>
<td>93</td>
<td>5D</td>
<td>135</td>
<td>125</td>
</tr>
<tr>
<td>30</td>
<td>1E</td>
<td>36</td>
<td>14</td>
<td>94</td>
<td>5E</td>
<td>136</td>
<td>126</td>
</tr>
<tr>
<td>31</td>
<td>1F</td>
<td>37</td>
<td>15</td>
<td>95</td>
<td>5F</td>
<td>137</td>
<td>127</td>
</tr>
</tbody>
</table>
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!
• “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?
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'`)

**Another ”how-many” question**
8-bit encodings

- 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 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
• 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
• 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/
What if you want to remove accents from text

- sometimes you need to work with both accented and non-accented characters
- no ad-hoc mapping dict from accented to non-accented chars is needed
- a standard solution:

```python
import unidecode
print(unidecode.unidecode("žšč"))
```

`zsc`
Working with more exotic scripts

- Example: if you need to debug a code that works with a script that you cannot read, even very simple tasks (such as visual checking whether two strings are identical) become uneasy

- a possible solution: use Unicode descriptions of characters for "reading" them

```python
import unicodedata as ucd
test = "žšč" # try to insert e.g. some Georgian or Malayalam or so
"for cha in test:
    print(f"character: {cha}\tdescription: {ucd.name(cha,'unknown')}\n"

character: ž  description: LATIN SMALL LETTER Z WITH CARON
character: š  description: LATIN SMALL LETTER S WITH CARON
character: č  description: LATIN SMALL LETTER C WITH CARON
```
Misc
• 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:

  ```
  Big Endian
  0x100 0x101 0x102 0x103
  01 23 45 67
  
  Little Endian
  0x100 0x101 0x102 0x103
  67 45 23 01
  ```

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 "А", and view the file using `hexdump -C`.
• This might be a source of confusion when working with multilingual data.
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
• e.g. the following looks perfectly OK unless you have some knowledge of Czech:
  Příliš lhu»oučký kůň úpěl dábelské ódy.

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}
• some editors have their own encoding declaration style, such as Emacs’s
  # -*- coding: <encoding-name> -*-
  
or VIM’s
  # vim: fileencoding=<encoding-name>
Try to foil the `file` command

- try to construct a file whose encoding is detected incorrectly by `file`
1. In spite of some relics 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.

https://ufal.cz/courses/npfl124