Unicode and UTF-8

Lecture 21

A standard for the discrete representation of written text
The Big Picture

glyphs

characters

code points

binary encoding

m, € 好

Cyrillic ef, Euro sign, Tei chou ten

Latin M, Apostrophe

U+0444, U+006D, U+2019

U+20AC, U+5975

U+006D, U+2019

6D, D1 84, E2 82 AC

E2 80 99, E5 A5 BD
The Big Picture

glyphs

characters

code points

binary encoding

m φ € 好

Latin M Cyrillic ef Euro sign Tei chou ten

U+0444 U+20AC U+5975

U+006D U+2019 U+2019

U+006D D1 84 E2 82 AC

U+2019 E2 80 99 E5 A5 BD
Text: A Sequence of Glyphs

- Glyph: “An individual mark on a written medium that contributes to the meaning of what is written.”
  - See foyer floor in main library
- One character can have many glyphs
  - Example: Latin E can be e, e, e, e, e, e...
- One glyph can be different characters
  - A is both (capital) Latin A and Greek Alpha
- One character of text can be made of multiple glyphs
  - An accented letter (é) is two glyphs
  - The ligature of f+i (fi) is two glyphs
Glyphs vs Characters

glyphs

characters

Latin small E

Greek capital alpha

Latin capital A
Security Issue

- Visual homograph: Two different characters that look the same
  - Would you click here: www.paypal.com?
  - Oops! The second ‘a’ is actually CYRILLIC SMALL LETTER A
  - This site successfully registered in 2005

- “Solution”
  - Heuristics that warn users when languages are mixed and homographs are possible
Unicode Code Points

- Each character is assigned a unique code point
- A code point is defined by an integer value, and is also given a name
  - one hundred and nine (ie 109, or 0x6d)
  - LATIN SMALL LETTER M
- Convention: Write code points as U +hex
  - Example: U+006D
- As of June 2014 (version 7.0):
  - Contains 113,000+ code points
  - Covers 123 scripts (and counting...)
Unicode: Mapping to Code Points

glyphs

characters

code points

binary encoding

m  φ  ,  €  好
Latin M  Cyrillic ef  Apostrophe  Euro sign  Tei chou ten
U+006D  U+0444  U+2019  U+20AC  U+5975
D1 84  E2 80 99  E2 82  AC  E5  A5  BD
6D  E2 82 AC  E5 A5 BD
Organization

- Code points are grouped into categories
  - eg, Basic Latin, Cyrillic, Arabic, Cherokee, Currency, Mathematical Operators
- Standard allows for $17 \times 2^{16}$ code points
  - ie, a little more than 1 million
  - U+0000 to U+10FFFF
- Each group of $2^{16}$ called a plane
  - U+nnnnnnnn, same green ==> same plane
- Plane 0 called *basic multilingual plane* (BMP)
  - Has (practically) everything you could need
  - Convention: code points in BMP written U +nnnnn, others written with 5 or 6 hex digits
Basic Multilingual Plane
UTF-8

- Encoding of code point (integer) in a sequence of bytes (octets)
  - Standard: all caps, with hyphen (UTF-8)
- Variable length
  - Some code points require 1 octet
  - Others require 2, 3, or 4
- Consequence: Can not infer number of characters from size of file!
- No endian-ness: just a sequence of octets
  D0 BF D1 80 D0 B8 D0 B2 D0 B5 D1 82...
UTF-8: Code Points & Octets

glyphs

characters

code points

binary encoding

m  φ  ,  €  好

Cyrillic ef  Euro sign  Tei chou ten

Latin M  Apostrophe

U+006D  U+0444  U+2019  U+5975

U+20AC  U+2019

E2 82 AC  E5 A5 BD

D1 84

6D  E2 80 99  E2 82 AC
UTF-8 Encoding Recipe

- **1-byte encodings**
  - First bit is 0
  - Example: 0110 1101 (encodes U+006D)

- **2-byte encodings**
  - First byte starts with 110...
  - Second byte starts with 10...
    - Example: 1101 0000 1011 1111
    - Payload: 1101 0000 1011 1111
      = 100 0011 1111
      = 0x043F
    - Codepoint: U+043F
      ie ι, Cyrillic small letter pe
UTF-8 Encoding Recipe

- **Generalization:** An encoding of length k:
  - First byte starts with k 1’s, then 0
    - Example 1110 0110 ==> first byte of a 3-byte encoding
  - Subsequent k-1 bytes each starts with 10
  - Remaining bits are payload

- **Example:**
  - 11100010 10000010 10101100
  - Payload: x20AC (ie U+20AC, €)

- **Consequence:** Stream is self-synchronizing
  - A dropped byte affects only that character
## UTF-8 Encoding Summary

<table>
<thead>
<tr>
<th>Unicode</th>
<th>Byte1</th>
<th>Byte2</th>
<th>Byte3</th>
<th>Byte4</th>
<th>example</th>
</tr>
</thead>
<tbody>
<tr>
<td>U+0000–U+007F</td>
<td>0xxxxxx</td>
<td></td>
<td></td>
<td></td>
<td>'§' U+0024</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>→ 00100100</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>→ 0x24</td>
</tr>
<tr>
<td>U+0080–U+07FF</td>
<td>110yyyx</td>
<td>10xxxxxx</td>
<td></td>
<td></td>
<td>'¢' U+00A2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>→ 11000010, 10100010</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>→ 0xC2, 0xA2</td>
</tr>
<tr>
<td>U+0800–U+FFFF</td>
<td>1110yyyy</td>
<td>10yyyyxx</td>
<td>10xxxxxx</td>
<td></td>
<td>'€' U+20AC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>→ 11100010, 10000010, 10101100</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>→ 0xE2, 0x82, 0xAC</td>
</tr>
<tr>
<td>U+10000–U+10FFFF</td>
<td>11110zzz</td>
<td>10zzyyyy</td>
<td>10yyyyxx</td>
<td>10xxxxxx</td>
<td>'₽' U+024B62</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>→ 11110000, 10100100, 10101101, 10100010</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>→ 0xF0, 0xA4, 0xAD, 0xA2</td>
</tr>
</tbody>
</table>

(from wikipedia)
Your Turn

- For the following UTF-8 encoding, what is the corresponding codepoint(s)?
  - F0 A4 AD A2

- What is the UTF-8 encoding of the following codepoint?
  - U+20AC
Security Issue

- Not all encodings are permitted
  - “overlong” encodings are illegal
  - example: C0 AF
    \[
    = \quad 1100 \ 0000 \quad 1010 \ 1111 \\
    = \quad \text{U+002F (should be encoded 2F)}
    \]

- Classic security bug (IIS 2001)
  - Should reject URL requests with “../..”
    - Scanned for 2E 2E 2F 2E 2E (in encoding)
  - Accepted “..%c0%af..” (doesn’t contain x2F)
    - 2E 2E C0 AF 2E 2E
  - After accepting, server then decoded
    - 2E 2E C0 AF 2E 2E decoded into “../..”

- Moral: Work in “code point” space!
Recall: URL encoding

- Concrete invariant (convention)
  - No space, ;, :, & in representation
  - To represent these characters, use %hh instead (hh is ASCII code in hex)
    - %20 for space
  - Q: What about % in abstract value?
Other (Older) Encodings

- In the beginning...
- Character sets were small
  - ASCII: only 128 characters (ie $2^7$)
  - 1 byte/character, leading bit always 0
- Globalization means more characters...
  - But 1 byte/character seemed so fundamental

- Solutions:
  - Use that leading bit!
    - Text data now looks just like binary data
  - Use more than 1 encoding!
    - Must specify data + encoding used
ASCII: 128 Codes

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

4B = Latin capital K
ISO-8859 family (eg -1 Latin)

<table>
<thead>
<tr>
<th>0-7F match ASCII</th>
<th>reserved (control characters)</th>
<th>A0-FF differ, eg:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>-1 &quot;Western&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-2 &quot;East European&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-9 &quot;Turkish&quot;</td>
</tr>
</tbody>
</table>
Windows Family (eg 1252 Latin)

|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 0x | NUL | SOH | STX | ETX | EOT | ENQ | ACK | BEL | BS  | HT  | LF  | VT  | FF  | CR  | SO  | SI  | 92  |
| 1x | DLE | DC1 | DC2 | DC3 | DC4 | NAK | SYN | ETB | CAN | EM  | SUB | ESC | FS  | GS  | RS  | US  | 92 |
| 2x | SP  | !  | "  | #  | $  | %  | &  | '  | (  | )  | *  | +  | ,  | -  | .  | /  | 92 |
| 3x | 0   | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | :  | ;  | <  | =  | >  | ?  | 92 |
| 4x | @   | A  | B  | C  | D  | E  | F  | G  | H  | I  | J  | K  | L  | M  | N  | O  | 92 |
| 5x | P   | Q  | R  | S  | T  | U  | V  | W  | X  | Y  | Z  | [  | \  | ^  | _  | `  | 92 |
| 6x | a   | b  | c  | d  | e  | f  | g  | h  | i  | j  | k  | l  | m  | n  | o  | 92 |
| 7x | p   | q  | r  | s  | t  | u  | v  | w  | x  | y  | z  | {  | }  | ~  | DEL| 92 |
| 8x | €   | £  | ¥  | £  | ¥  | €  | £  | ¥  | €  | £  | ¥  | £  | ¥  | €  | £  | ¥  | 92 |
| 9x | ©   | a  | «  | —  | —  | —  | —  | —  | —  | —  | —  | —  | —  | —  | —  | —  | 92 |

92 = apostrophe
Early Unicode and UTF-16

- Unicode started as $2^{16}$ code points
  - The BMP of modern Unicode
  - Bottom 256 code points match ISO-8859-1

- Encode every code point in 2 bytes (1 word)
  - Simple, but leads to bloat of ASCII text

- For code points outside of BMP
  - A pair of words (surrogate pairs) carry 20-bit payload split, 10 bits in each word
  - First: $1101\,10\,xx\,xxxx\,xxxx\ (\times D800-DBFF)$
  - Second: $1101\,11\,yy\,yyyy\,yyyy\ (\times DC00-DFFF)$

- U+D800 to U+DFFF are reserved code points in Unicode
  - And now we’re stuck with this legacy, even for UTF-8
Basic Multilingual Plane
UTF-16 and Endianness

- A multi-byte representation must distinguish between big & little endian
- One solution: Specify encoding in name
  - UTF-16BE or UTF-16LE
- Another solution: require byte order mark (BOM) at the start of the file
  - U+FEFF (ZERO WIDTH NO BREAK SPACE)
  - There is no U+FFFE code point
  - So FE FF ➔ BigE, while FF FE ➔ LittleE
  - Not considered part of the text
BOM and UTF-8

- Should we add a BOM to the start of UTF-8 files too?
  - UTF-8 encoding of U+FEFF is EF BB BF

- Advantages:
  - Forms magic-number for UTF-8 encoding

- Disadvantages:
  - Not backwards-compatible to ASCII
  - Existing programs may no longer work
  - eg. In Unix, shebang (#!, ie 23 21) at start of file is significant (file is a script)
    #! /bin/bash
Summary

- **Text vs binary**
  - In pre-historic times: most significant bit
  - Now: data is data

- **Unicode code points**
  - Integers U+0000..U+10FFFF
  - BMP: Basic Multilingual Plane

- **UTF-8**
  - A variable-length, self-synchronizing encoding of unicode code points
  - Backwards compatible with ISO 8859-1, and hence with ASCII too