Character Encodings

Jeremy Kahn

4/9/04

Last edited: 4/8/04

http://students.washington.edu/jgk/talks/char-enc/

Character Encodings: Why should CL care?

- Linguistics ? spoken language
- Writing schemes are linguistically interesting
- But computers don't know “letters” (or “words” or even “numbers”!)
- Computers know *bits* (okay, *bytes*)
- ... we need a mapping from “letter” (or “grapheme”) to bits
- (the wonderful thing about standards...)

Mappings are... important

- Wrong character mapping? corpora are noise
- Corpus as fetched may not be in the format your tools like
- Data interchange formats (e.g. $XML, HTML$) rely heavily on user (at some level) handling encodings correctly
- ... you should be able to read the docs, even if you can't write your own converter

Mappings are... sociologically interesting

- Historical and political:
  - Goes way way back: {U,V,W,u,v,w} all emerged from Roman V
  - *EBCDIC* is one reason nobody likes old *VAX* machines
- We'll explore some more contemporary history shortly...

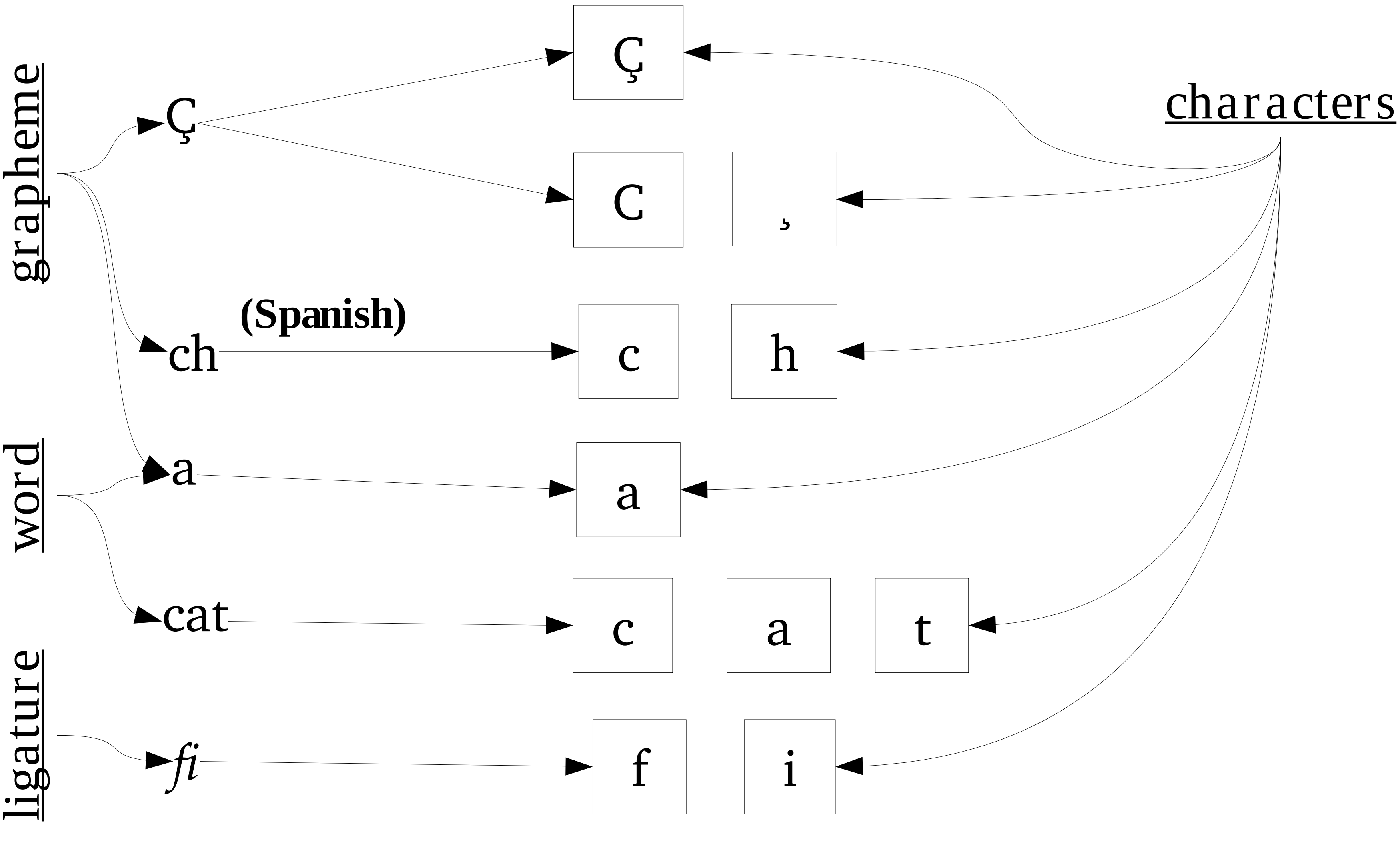
Vocabulary 1

- *Character*
  - “An abstract notion denoting a class of shapes declared to have the same meaning or form”
  - (Think “emic”)
- *Glyph*
  - A specific instance of a character
  - May (or not) include ligatures, serifs, etc
  - (Think “etic”)

Character *vs.* Glyph

- Character
- LATIN CAPITAL LETTER A
- LATIN SMALL LETTER A
- Glyph
- A *A* A A A
- a *a* a a
- ..but a character is not quite a grapheme:
  - "ch" is a grapheme in Spanish, but not English

A character ? a grapheme



Vocabulary 2

- *Character set* (also *character repertoire)*
  - A set of unique *characters*
- *Coded character set*
  - A *character repertoire*, plus a non-negative integer associated with each. This adds ordering, too.
- *Code point* (also *code position*)
  - The integer associated with a *character* in a *coded character set*

Character sets, coded character sets

- Digits (10) {0,1,2,3,4,5,6,7,8,9}
- English alphabet (26) {a,b,c,d,...v,w,x,y,z}
- (7-bit) ASCII (128) *handles English well*
- J?y? Kanji (1,945) *Japanese newspapers*
- Latin-1 (256) *most of W. Europe*
- Latin-2 (256) *most of E. Europe*
- Unicode (1000's) *most of the world (?)*

(How to find a Perl code point)

- Perl uses one datatype, but “under the hood” it can be a number or a string. Usually, you don't want to know, but here we do...
- The `ord` function takes a *character* as an argument, and returns its *code position* (as an integer)
- The `chr` function takes an integer as an argument, and returns the *character* at that *code position*

Vocabulary 3

- *Octet* (sometimes *byte*)
  - Eight bits. Computers “think” in octets. (People don't, at least not without lots of practice.)
  - (not all computer systems define *byte* to be 8 bits!)
- *Encoding* (also *character encoding*)
  - Any algorithmic scheme (often a list) that maps each *code point* (and thus, each *character)* of a *coded character set* to a unique series of *octets*.

Vocabulary 4 (-bidden)

The phrase “*code page*” is widely used, for many different purposes.

- character encoding
- bit orientation
- a variant of prefix encoding

So many, in fact, that it's nearly meaningless.

Therefore, let's avoid “code page”.

Character encodings

A brief history...

ASCII

- Names an encoding and a *character repertoire*
- Handles American English well (but not ¢, £)
  - 128 characters, thus aka 7-bit ASCII – can be represented in 7 bits
  - Most American typewriter characters
  - `0x00` to `0x20` are dedicated to *control characters*
- ... is probably with us forever, like QWERTY.

ASCII as a *coded character set*

[NUL]	0x00
[STX]	0x01
...	
(space)	0x20
!	0x21
...	
A	0x41
B	0x42
...	

`	0x60
a	0x61
b	0x62
...	
z	0x7A
...	
}	0x7D
~	0x7E
[DEL]	0x7F

ASCII as a *character encoding*

Map the code point into a 7-bit integer on the lower-order 7 bits.

[NUL]	`0x00`	(0)000 0000	
...			
A	`0x41`	(0)100 0001	(binary)
...		...	
a	`0x61`	(0)110 0001	
...		...	
~	`0x7E`	0111 1110	
[DEL]	`0x7F`	0111 1111	

ASCII f[l]ounders

Early signs of trouble with ASCII

- Alphabetization
- (whether to even *have* lowercase!)
- Which punctuation belongs in the canonical 128? (currency symbols? Why $, but not ¢?)
- And what about all them funny accented characters?

The first-born: Latin-1

- Expands to 256 codepoints
- Handles Western Europe rather nicely.
- Other names:
  - ISO-8859-1
  - aka “*8-bit ASCII*”, but this is not PC, as we'll see

Latin-1 as a coded character set

- Integers 0 to 127 (`0x00` to `0x7F`) same as ASCII.
- Integers 128 to 255 (`0x80` to `0xFF`) add most of the Western European characters, e.g.:

– ¡ *(Spanish)*	`0xA1`
– £ *(British English)*	`0xA3`
– Ä *(German)*	`0xC4`
– Þ *(Icelandic)*	`0xDE`
– ê *(French)*	`0xEA`

Latin-1 as a *character encoding*

Still easy: map the *code point* to an 8-bit integer on the entire *octet*.

A	(`0x41`)	0100 0001
...		(note the leading zero now!)
b	(`0x62`)	0110 0010
...		
ê	(`0xEA`)	1110 1010
...		
þ	(`0xFE`)	1111 1110
...		

Trouble with Latin-1

- Alphabetization
  - Even worse than before. Now all accented characters sort *after* all unaccented ones.
- Extensibility and inclusion
  - 256 code points just aren't very many for multi-lingual systems. Where is there room for détente? [or is it *detente*?]

Other heirs to ASCII

- Latin-2 (ISO-8859-2) “East” European (e.g. ?)
  - Polish, Czech, etc
- Latin-3 (ISO-8859-3) “South” European (e.g. ?)
  - Esperanto, Maltese, Turkish, etc.
- Latin-4 (ISO-8859-4) “North” European (e.g. ? , ?)
  - Estonian, Baltic, Lithuanian, Greenlandic, Lappish
- Latin-5 (ISO-8859-9)
  - Latin-1 minus Icelandic plus Turkish
- Latin-6 (ISO-8859-10)
  - Squeeze in Latvian and all of Nordic

Too many cooks?

- What if you want French and Icelandic at the same time?
  - >256 characters needed
  - Encoding collisions in the upper 128 codepoints

- There's just not enough room for this many characters.

Non-Roman Alphabets

- Cyrillic (ISO-8859-5)
  - Russian, Tajik, etc.
- Arabic (ISO-8859-6)
  - (doesn't include ligated forms – would quadruple!)
- Greek (ISO-8859-7)
- Hebrew (ISO-8859-8)
- All these still(!) retain the original ASCII values for the lower 128 code points.
- What about French and Greek?

Further problems with non-Roman alphabets

- Lots of new characters
- Mostly non-overlapping with US English
- Mostly non-overlapping with each other
- Complex ligating behaviors
- Arabic and Hebrew have a handedness problem:
  - Their text is written .tfel-ot-thgir

The real monster – East Asian encodings

- Chinese
  - Big-5
  - GB
- Korean
  - Johab
  - Wan-Sung
- Japanese
  - Shift-JIS

Difficulties with East Asian writing systems

- Number of unique characters
  - Chinese *hanzi*, Japanese *kanji*, Korean *hanja*, Vietnamese *ch?Hán* all number in the <u>thousands</u> (at the <u>low</u> end estimate)
  - Korean *hangul* need at least another 1300!
  - Where do all these go?
- Ordering
  - Alphabetization now nearly meaningless
  - What's a natural order for these characters?

Solutions for East Asia

- Wider characters
  - Use more octets. -- but this is space constrained!
- Shift encodings
  - A control character indicates when to “turn on” wide character mode. Difficult to randomly-access.
- Prefix encodings
  - Zipfian principle: common characters get shorter encodings...
- All these have problems...

Unicode

A proposed multi-lingual solution

Unicode

- As a first stab, consider Unicode a *coded character set* with a very large range of integers available.

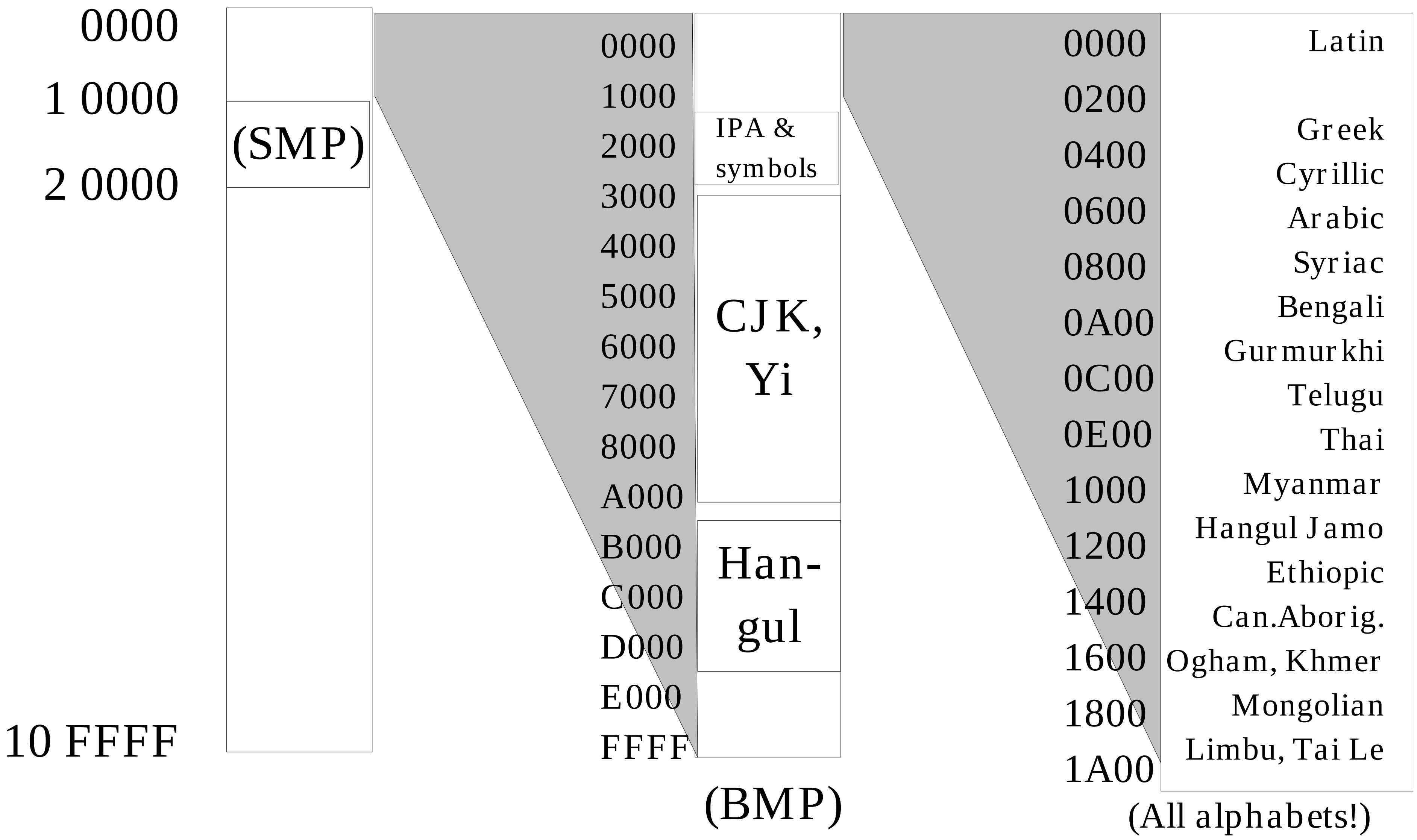
What's the big deal?

- Efficient (simple)
- Plain text
  - Good for corpus work
- Logical order
  - Handedness begone!
- Unification
- Convertibility

Size of Unicode coded character set

- Unicode defines more than a million *code points*.
- We'll only look at the *Basic Multilingual Plane*, which is the first 65,536...

Unicode is big



0000
1 0000
2 0000
10 FFFF

(SMP)

0000
1000
2000
3000
4000
5000
6000
7000
8000
9000
A000
B000
C000
D000
E000
FFFF

IPA & symbols

CJK, Yi

Han-gul

(BMP)

0000
0200
0400
0600
0800
0A00
0C00
0E00
1000
1200
1400
1600
1800
1A00

Latin
Greek
Cyrillic
Arabic
Syriac
Bengali
Gurmurkhi
Telugu
Thai
Myanmar
Hangul Jamo
Ethiopic
Can.Aborig.
Ogham, Khmer
Mongolian
Limbu, Tai Le

(All alphabets!)

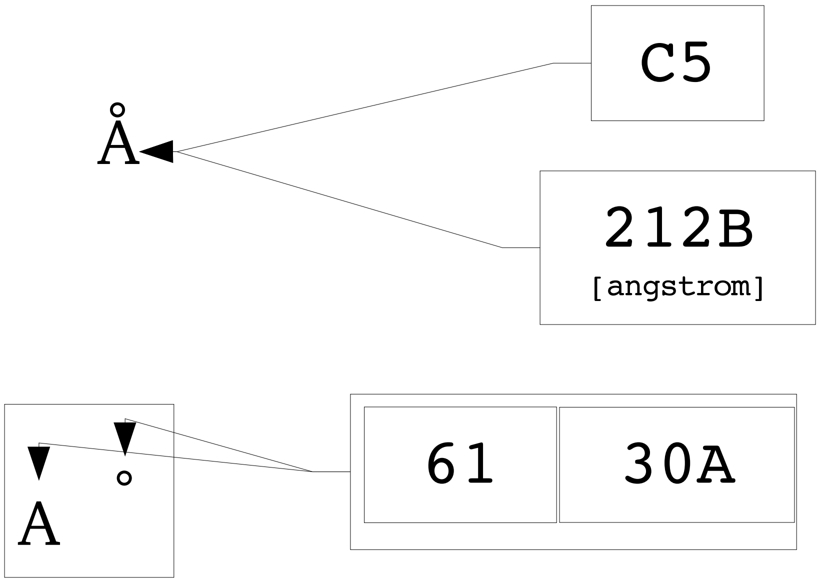
Separates CCS from CE

Unicode separates:

- The *coded character set*
- The *character encoding*

Allows user to choose compromises to make

Code point to encoding(s) (Unicode)



Character	Code point(s)	UTF-16	UTF-8
Å	C5	00 C5	C3 85
	212B [angstrom]	21 2B	E2 84 AB
A ˚	61 30A	00 41 03 0A	00 41 CC 8A

UCS Transformation Format (UTF)

UTF-16

- Very common, sometimes mislabeled “Unicode”
- Encodes each *character* within the BMP into 2 *octets*
- Character semantics and boundaries very simple

UTF-8

- Uses variable number of *octets* to encode the BMP
- Prefix mapping approach, skewed towards ASCII
- (link to discussion of the UTF-8 prefix map)

Unicode extras (1)

- Adopted clever ideas from other systems:
  - Handedness and combining characters
  - Johab Hangul decompositional encoding is used within Unicode
- UTF-16 has 2 variants: big-endian and little-endian.
  - Long and complicated history
  - “Byte Order Mark” (BOM) and its use

Character Encoding extras

- “ASCIIbetical” vs. Alphabetical
  - What are the implications?
  - How can they be resolved?
- CPAN: Sort::ArbBiLex
  - Allows “natural” alphabetic sorts

Unicode tools

Does my browser support UTF-8? How well?

- http://www.columbia.edu/kermit/utf8.html
- Includes the classic “I can eat glass, it doesn't hurt me.”

Got glyphs? Look for fonts here:

- http://www.alanwood.net/unicode/fonts.html

Platform-specific tools

Unix, Linux, and Mac OS X:

- *Emacs* MULE has good support for UTF-8
- One of several clever editors for Unicode: http://www.yudit.org/

Mac OS 9:

- http://www.hclrss.demon.co.uk/unicode/fonts_mac.html

Windows:

- get the Arial MS Unicode font!

Perl and Unicode

Perl is natively UTF-8 (rev 5.8 and up).

I/O layers work well:

```
open $fh, “<:latin1”, $file
   or die “couldn't open $file:$!\n”;
```

Good module support for Unicode from CPAN:

– Unicode::String

- `latin2eight.pl`, `eight2sixteen.pl`

Perl and encodings

- Encode
  - For handling encodings outside of Unicode
- Unicode::String
  - Object-oriented; straightforward
- Unicode::UCD
  - Works best if you're using 5.8+
- XML::Parser
  - Implicitly uses XML's dependence on encodings

Further readings

Links and tips for better understanding of character encodings and Unicode

Further readings (Unicode)

- *The Unicode Standard* (v. 3.0, 4.0), The Unicode Consortium
  - See also http://www.unicode.org for v. 4.0.1
- *Programming Perl*, *3rd Edition*, Larry Wall *et al*. (aka "The Camel Book")
  - See especially chapter 15, "Unicode"
- Simon Cozens has a great talk, which inspired this one:
  - http://www.netthink.co.uk/downloads/unicode.pdf

Further readings (other encodings)

- East Asian encodings:
  - *CJKV Information Processing*, Ken Lunde (aka “The Blowfish Book”)
- ISO-8859 (alphabetic) encodings:
  - http://czyborra.com/charsets/iso8859.html
- UW Library (!):
  - http://www.lib.washington.edu/help/catalog/unicode/unicodehelp.html