COM3110/4115/6115: Text Processing

Text Encoding

Rob Gaizauskas

Department of Computer Science
University of Sheffield

Overview

- Languages and Writing Systems
- Characters vs Glyphs
- The Challenge of Character Encoding
- A Note on Binary/Octal/Hexadecimal Representations
- A (Very) Brief History of Character Encoding
- An Introduction to Unicode

Languages and Writing Systems

- Physical faculties required for spoken language
  - Broca's area, region of brain associated with language
  - vocal apparatus

  arose in homo sapiens between 2 million and 300,000 years ago
- Written language only emerged between 3,500-3,100 BCE in 3 independent centres:
  - amongst the Harappans in the Indus Valley (Indus language)
  - amongst the Sumerians in Mesopotamia (cuneiform)
  - amongst the Egyptians (hieroglyphics)

  though some dispute about whether earlier evidence counts as writing

Languages and Writing Systems (cont)

- Not an accident that the emergence of written language coincides with the stunning acceleration of human culture, science, technology and population size over the last 3000-4000 years.
  - ◇ "Humankind is defined by language; but civilization is defined by writing." (Daniels and Bright, 1996)
- Today some 4000-5000 languages spoken (depending on, e.g. distinction made between language and dialect)
- However, only something like 170-180 writing systems in use + ∼10 undeciphered ones (according to `omniglot.com`)
  - ◇ Not clear how many languages lack writing systems

Languages and Writing Systems (cont)

- What is a writing system (also called a script)?

  "a system of more or less permanent marks used to represent an utterance in such a way that it can be recovered more or less exactly without the intervention of the utterer"

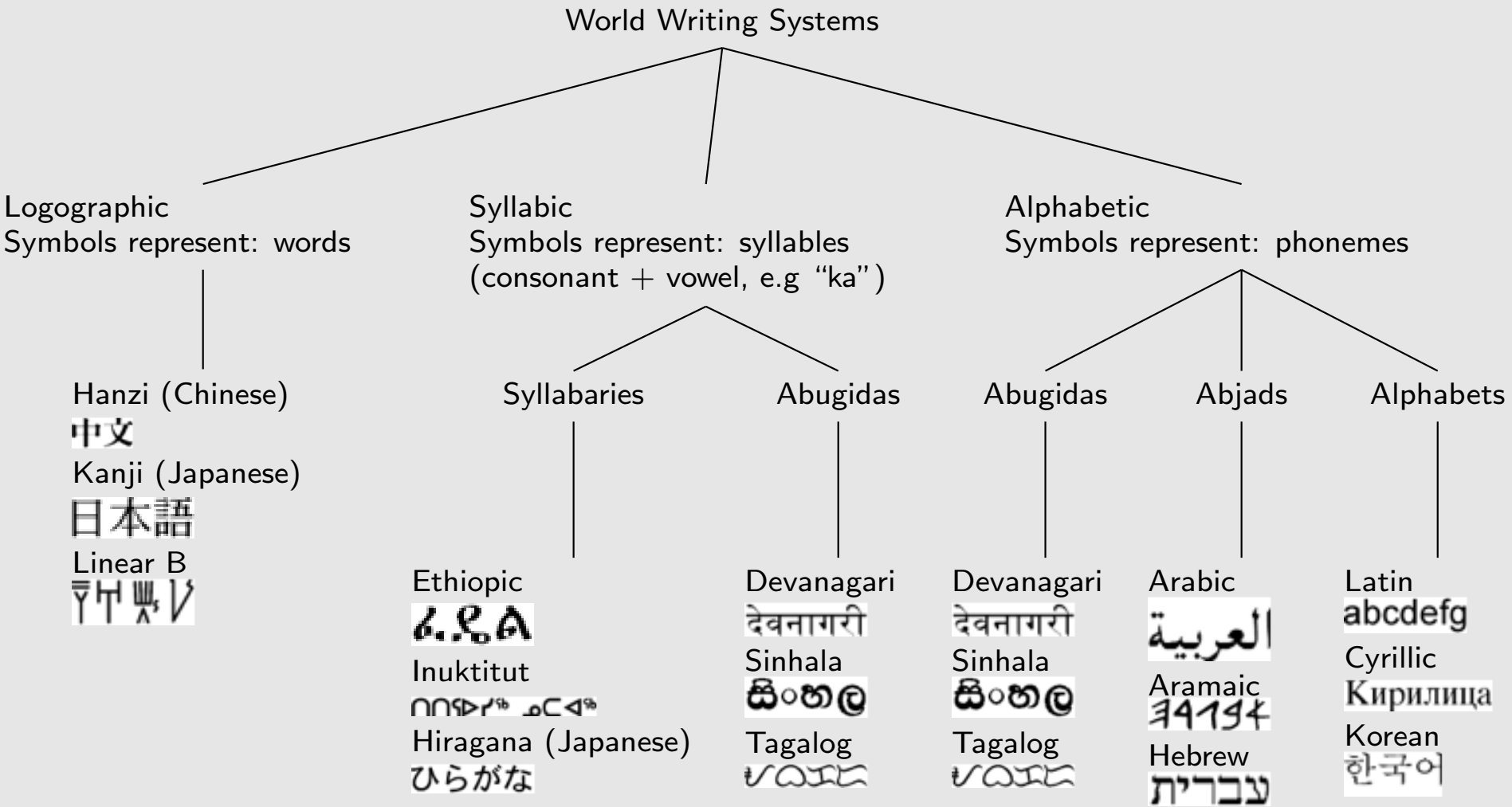
  Peter T. Daniels, *The World's Writing Systems*

- There is a many-to-many mapping between languages and scripts
  - ◇ many languages may share one script
    - e.g. most of the languages of Western Europe use the Roman scipt with some small variations (accents in French, some extra characters in Spanish, the Scandinavian languages)
  - ◇ some languages may have multiple scripts
    - Japanese – kanji (from Chinese); hiragana (for grammatical particles); katakana (for loan words from languages other than Chinese); arabic numerals

Languages and Writing Systems (cont)

- Writing systems may be classified along a number of dimensions:
  - Directionality
    - left-to-right – e.g. English, Russian
    - right-to-left – e.g. Arabic, Hebrew
    - top-to-bottom – e.g. Chinese
    - bottom-to-top – e.g. Mongolian
    - boustrophedon ("ox-turning") – e.g. (some) ancient Greek
  - Historical derivation
  - Relationship between symbols and sounds
    - *phonological* systems show a clear relationship between sounds of the language and symbols
    - *non-phonological* systems do not
- Sound/symbol relationship generally agreed to be the best way to classify languages
  - but no consensus among scholars as to the best set of sub-classifications

Taxonomy of Writing Systems



Characters vs Glyphs

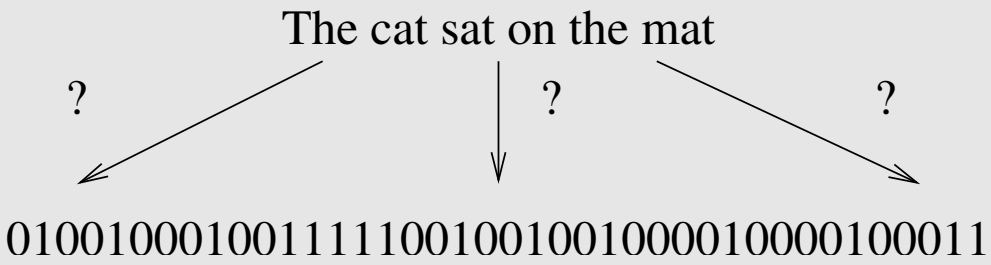
- A character is the smallest component of a writing system that has a semantic value.
  - I.e. changing one character to another – e.g. *bat* → *cat* – changes the meaning of the word in which it occurs
  - The word grapheme (by analogy with phoneme, the smallest sound unit in spoken language) is used
    - sometimes synomously with "character"
    - sometimes to indicate multiple characters that function like a character – e.g. á
- A glyph is a representation of a character or characters, as it/they is/are rendered or displayed.
  - E.g. **A**, A, $\mathcal{A}$ are different glyphs representing the Latin character A
  - A repertoire of glyphs of similar appearance makes up a font
  - Changing font does **not** change the basic meaning of the word(s) whose font is changed
    - Though changing font to italics or bold can change, e.g., emphasis, which is an aspect of meaning

Characters vs Glyphs (Cont)

- The relationship between glyphs and characters can be complex
  - for any character there may be many glyphs representing it (in different fonts)
  - a single glyph may correspond to a number of characters.
    - E.g. "ﬁ" is typically a single glyph used to represent "f" followed by "i" in typesetting English
  - there may be arbitrariness – should é
    - be stored as two characters and rendered using two glyphs?
    - be stored as two characters and rendered by one glyph?
    - be stored as one character and rendered by one glyph?
- When deciding on the set of characters for a language for purposes of representing them in a computer, it is extremely important (e.g. for sorting, indexing purposes) to separate out characters from glyphs
  - the underlying representation of a text should contain the character sequence only
  - the final appearance of the text is the responsibility of the rendering process

The Challenge of Character Encoding

- All data in held in a digital computer is ultimately stored in binary form – i.e. as sequences of 0's and 1's
- Therefore, to store character data, i.e. text, in a digital computer it is necessary to agree an **encoding scheme**



- Various issues in designing a character encoding scheme:
  - ◇ Generality How many languages are you aiming to deal with?
  - ◇ Character Set Specification What is the character set(s) to be encoded?
  - ◇ Hardware Issues What are the minimal units of addressable memory?
  - ◇ Variable/Fixed Width Should every character occupy the same number of bits? Or should more frequently occurring characters occupy fewer?
  - ◇ Interoperability Who do you want to play with? Do they already have a different scheme? (**standards**)

A Note on Binary, Octal & Hexadecimal Representations

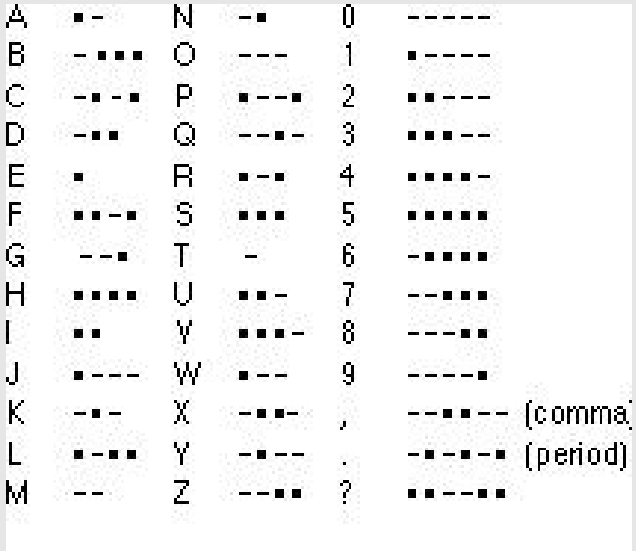
- One byte = 8 bits.
- 8 bits can represent $2^8 = 256$ distinct values – 0 - 255.
- In binary notation, these range from

$$\begin{array}{l} 00000000 \\ 00000001 \\ 00000010 \\ \vdots \\ 11111111 \end{array}$$

- In octal (base 8) notation, they range from: 000 to 377.
  - $o377 = (3 * 64) + (7 * 8) + 7)$.
- In hexadecimal (base 16) notation, they range from 00 to *FF*
  - $xFF = (F * 16) + F$.
- Two bytes = 16 bits, and can represent $2^{16} = 65,536$ distinct values.
- Two byte values most commonly written in hexadecimal; range from 0000 - *FFFF*
  - $d65535 = xFFFF = (15 * 4096) + (15 * 256) + (15 * 16) + 15$

A (Very) Brief History of Character Encoding

- "Standards are a good thing; and the good thing about standards is that there are so many to chose from ..."
- **Telegraphy and the Morse and Baudot Codes**
  - Telegraph invented by Wheatstone + Cooke (UK), 1837
  - Samuel Morse (US) invents simpler system + code, 1838
    - Variable length code, based on 2 values – dot's or dashes

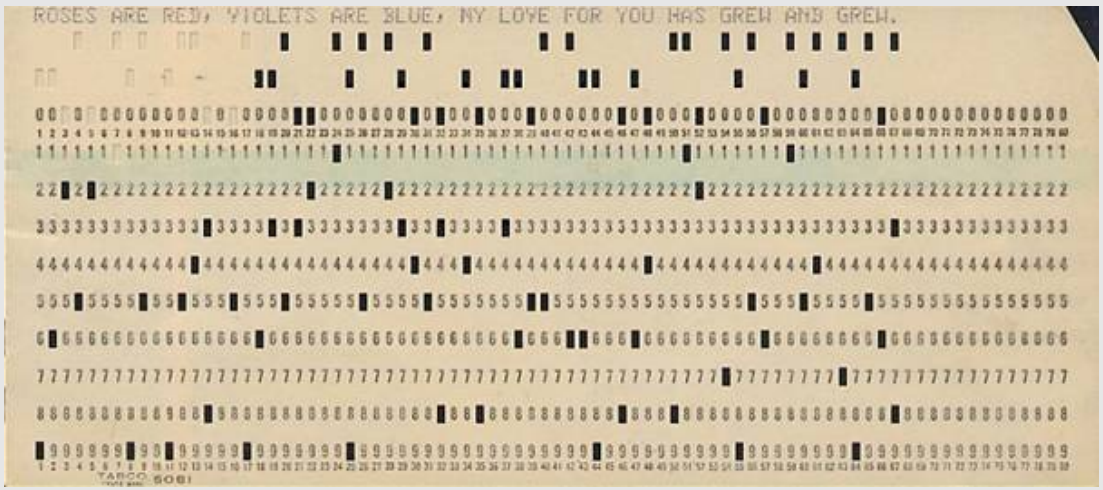


A	·–	N	–·	0	–––––
B	–···	O	–––	1	·––––
C	–·–·	P	·––·	2	··–––
D	–··	Q	––·–	3	···––
E	·	R	·–·	4	····–
F	··–·	S	···	5	·····
G	––·	T	–	6	–····
H	····	U	··–	7	––···
I	··	V	···–	8	–––··
J	·–––	W	·––	9	––––·
K	–·–	X	–··–	,	––··–– (comma)
L	·–··	Y	–·––	.	·–·–·– (period)
M	––	Z	––··	?	··––··

  - Baudot (Fr) invents printing telegraph ("teleprinter"), 1874
    - used first binary code for textual data – 5 bit Baudot code
    - used a shift code to double code space; i.e., $2 * 2^5$

A (Very) Brief History of Character Encoding (cont)

- **Hollerith and Punched Cards**
  - Herman Hollerith (US) invents punched card tabulating machines to process 1890 US census data
  - code based on holes punched in 12 (row) $\times$ 80 (col) cards
  - Hollerith commercialises invention – Tabulating Machine Co (1896) – becomes International Business Machines Corp (IBM) in 1924



From: `http://wvegter.hivemind.net/abacus/CyberHeroes/Hollerith_files/image006.jpg`

A (Very) Brief History of Character Encoding (cont)

- **ASCII**
  - American Standards Association (ASA – later changed to American National Standards Institute [ANSI]) proposed a 7-bit code – the American Standard Code for Information Interchange (ASCII) (1963, finalised 1968)

ASCII

	_0	_1	_2	_3	_4	_5	_6	_7	_8	_9	_A	_B	_C	_D	_E	_F
0_	U+0000 NUL 0	U+0001 SOH 1	U+0002 STX 2	U+0003 ETX 3	U+0004 EOT 4	U+0005 ENQ 5	U+0006 ACK 6	U+0007 BEL 7	U+0008 BS 8	U+0009 HT 9	U+000A LF 10	U+000B VT 11	U+000C FF 12	U+000D CR 13	U+000E SO 14	U+000F SI 15
1_	U+0010 DLE 16	U+0011 DC1 17	U+0012 DC2 18	U+0013 DC3 19	U+0014 DC4 20	U+0015 NAK 21	U+0016 SYN 22	U+0017 ETB 23	U+0018 CAN 24	U+0019 EM 25	U+001A SUB 26	U+001B ESC 27	U+001C FS 28	U+001D GS 29	U+001E RS 30	U+001F US 31
2_	U+0020 SP 32	U+0021 ! 33	U+0022 " 34	U+0023 # 35	U+0024 $ 36	U+0025 % 37	U+0026 & 38	U+0027 ' 39	U+0028 (40	U+0029) 41	U+002A * 42	U+002B + 43	U+002C , 44	U+002D - 45	U+002E . 46	U+002F / 47
3_	U+0030 0 48	U+0031 1 49	U+0032 2 50	U+0033 3 51	U+0034 4 52	U+0035 5 53	U+0036 6 54	U+0037 7 55	U+0038 8 56	U+0039 9 57	U+003A : 58	U+003B ; 59	U+003C < 60	U+003D = 61	U+003E > 62	U+003F ? 63
4_	U+0040 @ 64	U+0041 A 65	U+0042 B 66	U+0043 C 67	U+0044 D 68	U+0045 E 69	U+0046 F 70	U+0047 G 71	U+0048 H 72	U+0049 I 73	U+004A J 74	U+004B K 75	U+004C L 76	U+004D M 77	U+004E N 78	U+004F O 79
5_	U+0050 P 80	U+0051 Q 81	U+0052 R 82	U+0053 S 83	U+0054 T 84	U+0055 U 85	U+0056 V 86	U+0057 W 87	U+0058 X 88	U+0059 Y 89	U+005A Z 90	U+005B [91	U+005C \ 92	U+005D] 93	U+005E ^ 94	U+005F _ 95
6_	U+0060 ` 96	U+0061 a 97	U+0062 b 98	U+0063 c 99	U+0064 d 100	U+0065 e 101	U+0066 f 102	U+0067 g 103	U+0068 h 104	U+0069 i 105	U+006A j 106	U+006B k 107	U+006C l 108	U+006D m 109	U+006E n 110	U+006F o 111
7_	U+0070 p 112	U+0071 q 113	U+0072 r 114	U+0073 s 115	U+0074 t 116	U+0075 u 117	U+0076 v 118	U+0077 w 119	U+0078 x 120	U+0079 y 121	U+007A z 122	U+007B { 123	U+007C | 124	U+007D } 125	U+007E ~ 126	U+007F DEL 127

From: `http://www.gammon.com.au/unicode/`

  - ASCII adopted by all US computer manufacturers except market-leaders IBM – proposed their own code – Extended Binary Coded Decimal Interchange Code (EBCDIC)
  - Remains the most widely used coding scheme for English language text

A (Very) Brief History of Character Encoding (cont)

- **ISO/IEC 8859**
  - ASCII insufficient for Western European languages
  - International Standards Organisation proposed a derivative with 10 codes left for national variants – ISO 646 (1967)
  - Has been rationalised and extended to cover most needs of Western and Eastern Europe – ISO 8859 (revised up until 2001)
  - Despite ISO 8859 many American PC companies built OS-specific extensions of ASCII for European languages that were not compatible with ISO 8859
  - Not maintained extended since 2001 – now subsumed by Unicode/UCS
  - See `https://en.wikipedia.org/wiki/ISO/IEC_8859`.
- Paralleling develops in North America and Europe, efforts to develop standardised character codes have gone on in Asia
  - for Chinese, Japanese and Korean (CJK) codes – e.g JIS
  - for South Asian languages – e.g. ISCII

A (Very) Brief History of Character Encoding (cont)

- So far have discussed codes as developed to meet needs of individual countries. But may need to
  - process multiple languages in one document
  - reuse software for products dealing with multiple languages
- By mid-1980s it became clear there was a need for multilingual coding schemes
- Early efforts
  - some pioneering efforts by Xerox (Xerox Character Code Standard – XCCS) and IBM in early 1980's
  - TRON project (tronweb.super-nova.co.jp/homepage.html)
    - ongoing in Japan (U. Tokyo + Japanese industry) since 1984
    - uses escapes to shift between character planes – "limitlessly extensible"
    - built into BTRON operating system that runs on PCs
- By late 1980's two major efforts underway to build character encoding schemes to embrace all the world's languages – past, present and future
  - Unicode started as initiative of US software industry in 1988
  - The Universal Coded Character Set (UCS) aka ISO/IEC 10646 started in 1989 as an initiative of European and Japanese researchers

A (Very) Brief History of Character Encoding (cont)

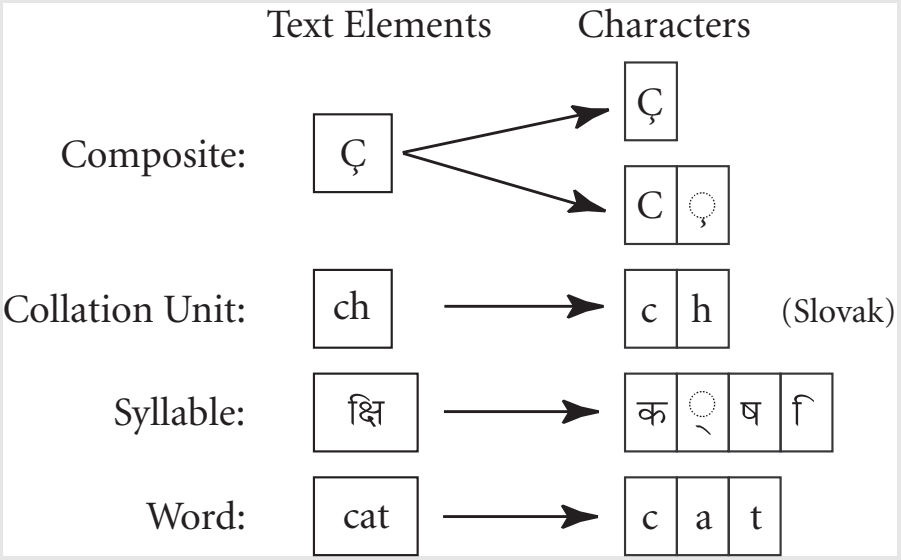
- Unicode (`www.unicode.org`; `en.wikipedia.org/wiki/Unicode`)
  - committed to 16 bit codes and to avoiding escape sequences
  - supported by, e.g. Apple, IBM, Microsoft, Google, ...
  - now part of XML and HTML standards, Java, Perl, Python, ...
  - current version (Version 10.0.0) contains a repertoire of 136,755 characters covering 139 modern and historic scripts, as well as multiple symbol sets
  - has been criticised in East Asia for not meeting technical needs (character coverage insufficient; no way to expand without moving beyond 16 bit codes)
    - proponents: "the universal character encoding scheme for written characters and text" and "the foundation for global software"
    - opponents: "little more than an exercise in Western cultural imperialism"
- ISO/IEC10646 (`en.wikipedia.org/wiki/Universal_Coded_Character_Set`)
  - initially (1990) proposed a scheme involving 128 groups of 256 planes of 256 rows of 256 cells – could encode 679,477,248 characters
  - Supported four byte, two byte and variable length (1-5 byte) encodings
  - Was opposed by Unicode as too complex/requiring too much storage – wanted a two byte scheme
  - Later Unicode was forced to admit 16 bytes was insufficient and added "surrogate pairs" (see below)
- Unicode and UCS have now converged, are maintained together, and are code-for-code identical

Unicode: Architectural Context – Text Processes

- *"The Unicode Standard is the universal character encoding standard for written characters and text. It defines a consistent way of encoding multilingual text that enables the exchange of text data internationally and creates the foundation for global software."* (version 10.0.0, p. 1) (Unicode Standard Version 3.0, p .1)
- No character encoding scheme is an end in itself: it is a means to enable useful text processing on computers.
- Basic low-level text processes computers expected to support include:
  - rendering characters visible (including ligatures, contextual forms, and so on)
  - breaking lines while rendering (including hyphenation)
  - modifying appearance, such point size, kerning, underlining, slant and weight (light, demi, bold, etc.)
  - determing units such as "word" and "sentence"
  - interacting with users in processes such as selecting and highlighting text
  - accepting keyboard input and editing stored text through insertion and deletion
  - comparing text in operations such as in sorting or or determining the sort order of two strings
  - analysing text content for, e.g. spell checking, hyphenation, parsing morphology (finding word roots/stems/affixes)
  - treating text as bulk data for, e.g., compression, transmission

Unicode: Architectural Context – Text Elements, Code Values and Text Processes

- Unfortunately, for each text process, languages differ in what constitutes a text element.
  - ◇ e.g. In Spanish "ll" sorts between "l" and "m" (i.e. should be treated as a text element), but in rendering it is best treated as two "l" elements.
- Thus, the first challenge in designing an encoding scheme for a language is to agree on the set of **abstract characters** to be encoded – those characters that will be assigned a **code value**.
- For English this seems straightforward (but, e.g., should "A" and "a" get 2 code values or 1?); other languages are not so simple.



Unicode: Design Principles

- Underpinning Unicode are 10 design principles (version 10.0.0):

Principle	**Statement**
Universality	The Unicode Standard provides a single, universal repertoire
Efficiency	Unicode text is simple to parse and process
Characters, not glyphs	Unicode encodes Characters, not glyphs
Semantics	Characters have well-defined semantics
Plain text	Unicode characters represent plain text
Logical order	Default for memory representation is logical order
Unification	Unicode unifies duplicate characters within scripts across languages
Dynamic composition	Accented forms can be dynamically composed
Stability	Characters, once assigned, cannot be reassigned and key properties are immutable
Convertibility	Accurate convertibility is guaranteed between Unicode and other widely accepted standards

Unicode: Design Principles

- **Universality**
  - Unicode encodes a single, very large set of characters, encompassing all the characters needed for worldwide use.
  - intended to be universal in coverage, containing all characters for textual representation in all modern writing systems, in most historic writing systems, and for symbols used in plain text.
  - designed to meet the diverse needs of business, educational, liturgical and scientific users
  - out of scope:
    - writing systems for which insufficient information is available to enable reliable encoding of characters
    - writing systems that have not become standardized through use
    - writing systems that are nontextual in nature.
- **Efficiency** – designed to allow efficient implementations
  - no escape characters or shift states – each character has same status
  - all encoding forms self-synchronising: limited backup required to find character when randomly accessing a string
    - in UTF-16, at most one byte back-up when dealing with surrogate pairs
    - in UTF-8, at most three bytes backup
  - characters of a script grouped together as far as possible

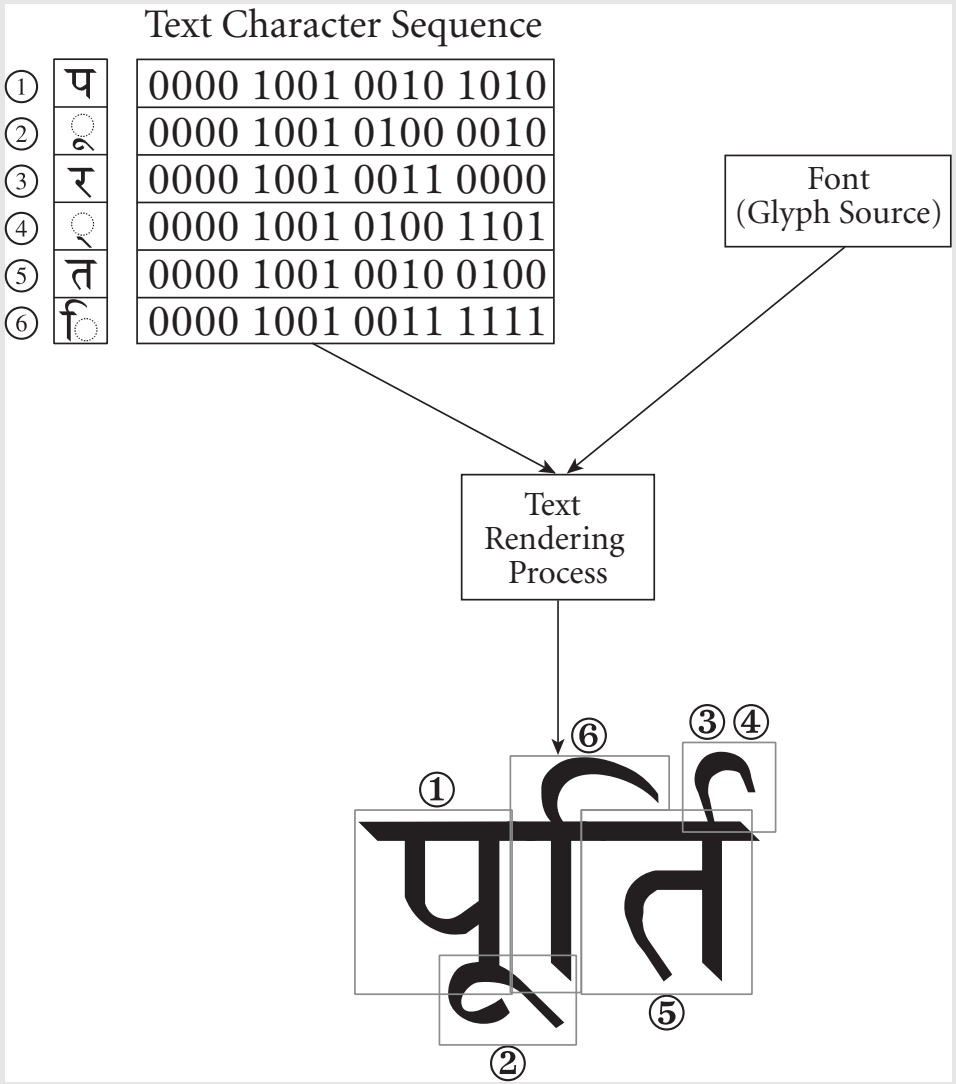
Unicode: Design Principles (cont)

- **Characters, not Glyphs**
  - Unicode distinguishes
    - characters: smallest components of written language with semantic value
    - glyphs: the shapes characters can have when displayed
  - Unicode standard deals *only* with character codes

Glyphs	Unicode Characters
A 𝐴 𝒜 𝔸 A 𝘈 𝒜 A	U+0041 LATIN CAPITAL LETTER A
a 𝐚 𝒂 𝕒 a a 𝒂 𝒶 𝐚	U+0061 LATIN SMALL LETTER A
п *n* *ū*	U+043F CYRILLIC SMALL LETTER PE
ه ه ه ه	U+0647 ARABIC LETTER HEH
fi fi	U+0066 LATIN SMALL LETTER F + U+0069 LATIN SMALL LETTER I

Unicode: Design Principles (cont)

- **Characters, not Glyphs (cont)**
  - ◇ Glyph shapes, how they are selected and rendered are the responsibility of font vendors (font = set of glyphs)



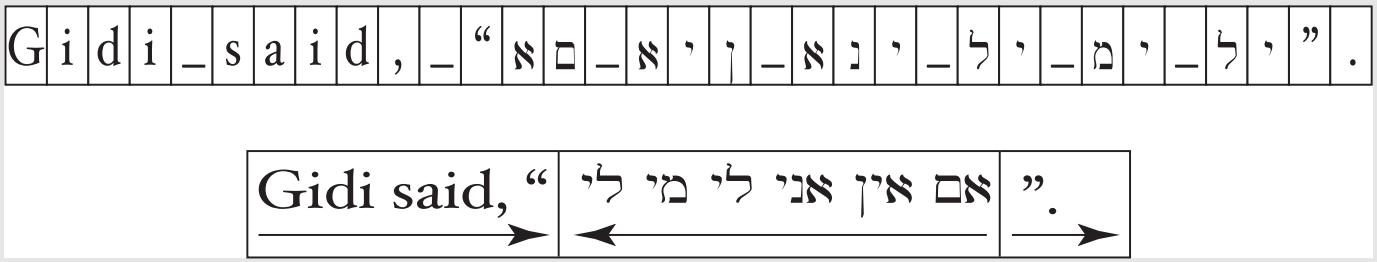
Unicode: Design Principles (cont)

- **Semantics**
  - character property tables are provided for use in, e.g., parsing and sorting algorithms
  - properties include: numeric, spacing, combination, directionality

- **Plain Text**
  - *plain text* is a pure sequence of character codes
    - underlying content stream to which formatting is applied
    - public, standardised, universally readable
  - *fancy/rich text* is plain text plus additional information, such as font size, colour, hypertext links, etc.
    - extra info can be embedded (SGML, HTML, XML, TeX), or in parallel store with links to plain text (word processors)
    - may be implementation-specific, proprietary
  - Unicode encodes plain text only, where
    - "plain text must contain enough information to permit the text to be rendered legibly, and nothing more" (Unicode Standard, Version 10.0.0)

Unicode: Design Principles (cont)

- **Logical Order**
  - Unicode text is stored in logical order – generally the same as the order it would be input via a keyboard
  - In some cases this may differ from display order



  - Directionality property of characters generally sufficient to render plain text, but not always
    - Unicode does contain characters to specify change of direction

- **Unification**
  - duplicate encoding of the same character across languages is avoided in general, e.g.
    - punctuation
    - “Y” (French *i grecque*, German *ypsilon*, English *wye* all represented by U+0057)
  - sometimes this principle is violated to support compatibility with existing standards

Unicode: Design Principles (cont)

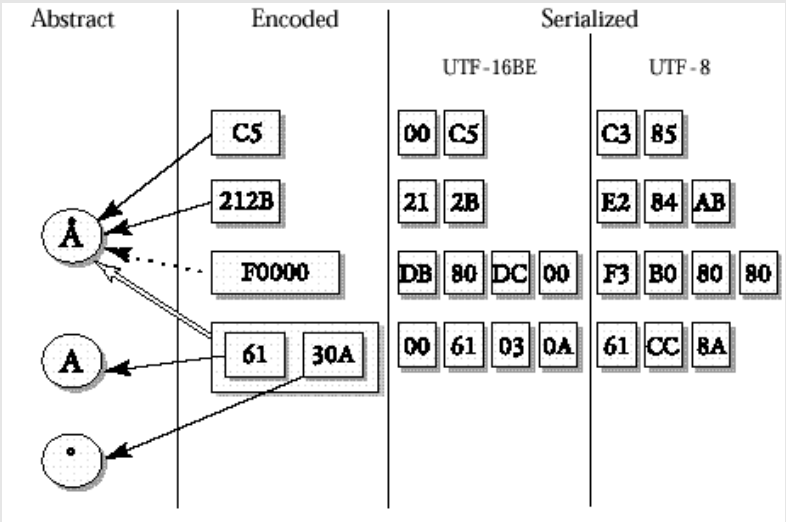
- **Dynamic Composition**
  - by separating codes, for e.g. base characters and accents, Unicode supports the dynamic composition of various forms
  - this process is open-ended – new forms not currently used may be created

- **Stability**
  - Certain aspects of the Unicode Standard must be absolutely stable between versions – so that implementers and users can be guaranteed text data, once encoded will preserve meaning
  - Means once Unicode characters are assigned, their code point assignments cannot be changed, nor can characters be removed
  - Unicode character names also never changed, so that they can be used as valid dentifiers across versions
  - While characters are retained there is a mechanism for deprecating characters whose use is to be discouraged

- **Convertibility**
  - Character identity is preserved for interchange with a number of different base standards (national, international, vendor-specific)

Unicode: Encoding Model

- The Unicode model may be first approximated by a three level model consisting of:
  - ◇ an abstract character repertoire
  - ◇ their mapping to a set of integers also called the codespace
    - a particular integer in this set is called a code point
    - an abstract character is assigned to a code point and is then referred to as an encoded character
    - the Unicode codespace consists of the integers from 0 to $10FFFF_{16}$, comprising 1,114,112 code points
  - ◇ their encoding forms + byte serialization

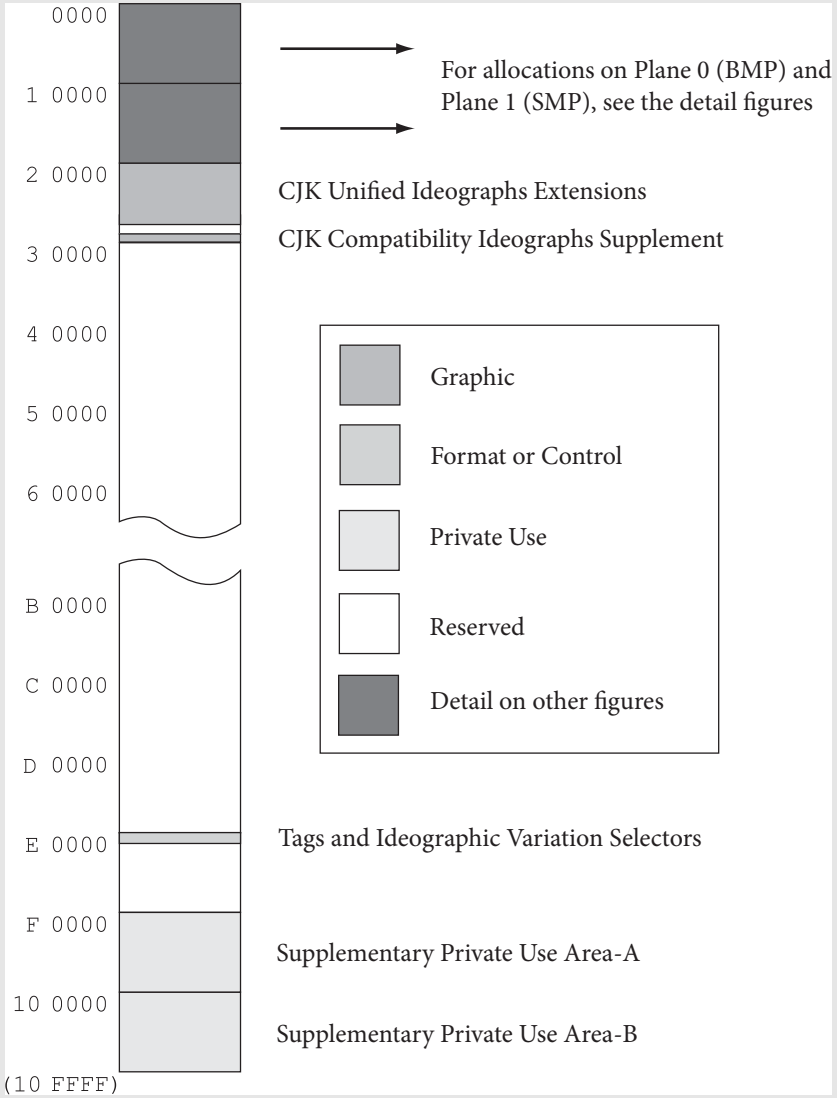


Unicode: Encoding Model (cont)

- Encoded characters are alternate ways the same text element may be assigned an integer code in the Unicode code value space
  - C5 is the precomposed static form of "A-ring"
  - 212B is the code for Angstrom unit
  - F0000 is a hypothetical "private use" surrogate pair
  - 61 30A are "A" and "ring" – to be dynamically composed

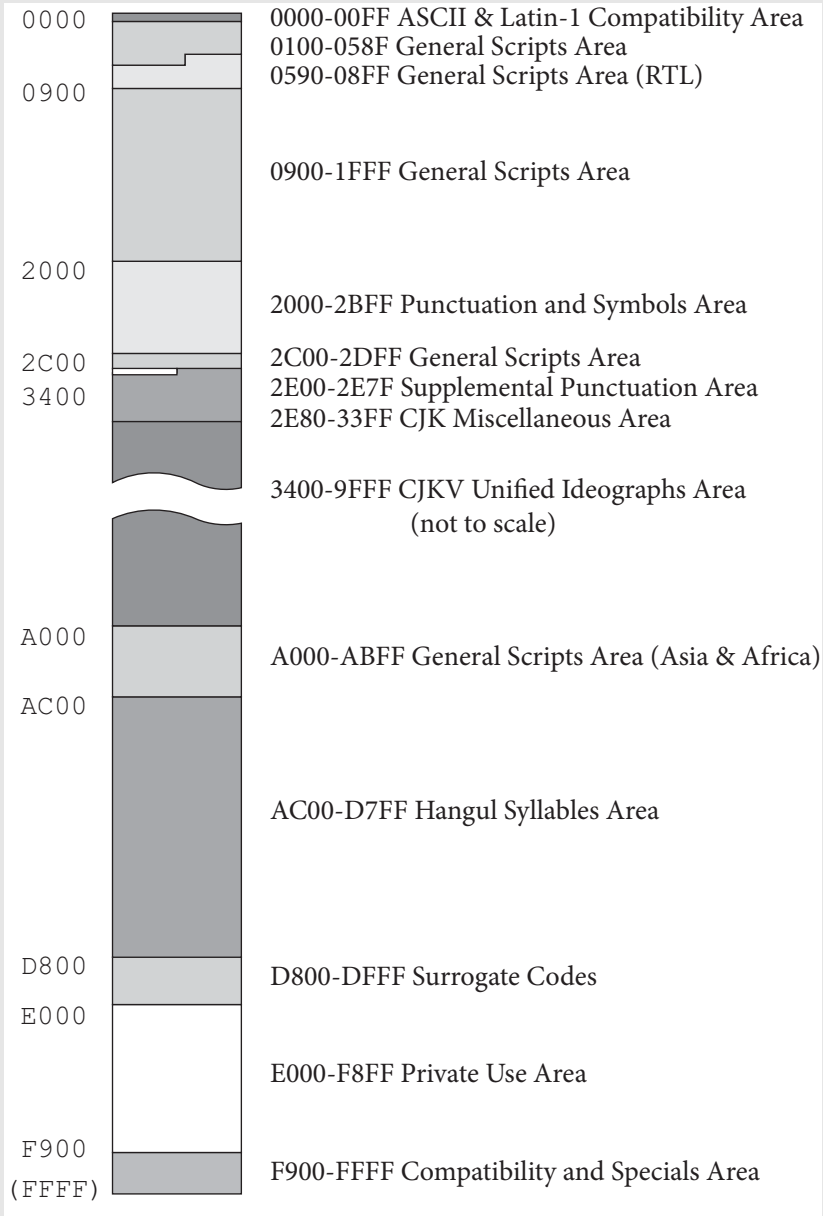
Unicode: Allocation Areas

- Unicode codespace of $10FFFF_{16}$ codepoints is divided into planes each consisting of FFFF (64k decimal) code points
- The first plane (codepoints 0-FFFF), is called the basic multilingual plane(BMP)



Unicode: Allocation Areas (cont)

- Codespace assignment attempts to follow certain principles – e.g. scripts grouped together, in order of accepted standards
- No guarantee of correlation with sort order or case



Encoding Forms and Encoding Schemes

- Once a mapping from an abstract character set to a set of integers has been defined two further mappings need to be specified
  - Character Encoding Form – a mapping from a set of integers to a set of sequences of code units of specified width (e.g. 8-bit bytes).
  - Character Encoding Scheme – a mapping from a set of sequences of code units to a serialized sequence of bytes
- Often character encoding forms and character encoding schemes have the same names – because some default serialization is assumed – but they should not be confused.
- The most common character encoding forms are:
  - UTF-32 (Unicode (or UCS) Transformation Format-16)
  - UTF-16 (Unicode (or UCS) Transformation Format-16)
  - UTF-8 (Unicode (or UCS) Transformation Format-8)

Character Encoding Forms: UTF-32

- UTF-32 (Unicode Transformation Format-32)
  - Each Unicode code point is represented directly by a single 32-bit (4 byte) code unit
  - advantages:
    - Simple: allows all Unicode code points to be mapped into 1 fixed length code units (bytes)
    - Note: a 32-but code could handle up to 4.29 billion distinct characters
  - disadvantages:
    - files containing only Latin texts are four times as large as they are in single byte encodings, such as ASCII or ISO Latin-1 or UTF-8
    - not backwards/forwards compatible with ASCII – so programs that expect a single-byte character set won't work on a UTF-32 file *even if it only contains Latin text*.

Character Encoding Forms: UTF-16

- UTF-32 (Unicode Transformation Format-32)
  - original/default Unicode encoding form – characters are assigned a 16-bit value, except for surrogate pairs which consist of two 16-bit values
  - advantages:
    - allows all Unicode code points to be mapped into 2 code units (bytes)
  - disadvantages:
    - files containing only Latin texts are twice as large as they are in single byte encodings, such as ASCII or ISO Latin-1
    - not backwards/forwards compatible with ASCII – so programs that expect a single-byte character set won't work on a UTF-16 file *even if it only contains Latin text*.

An Aside on Surrogate Pairs

- Unicode UTF-16 text consists of sequences of 16-bit character codes
  - in principle this allows for the encoding of 65,536 distinct values (enough for all scripts currently used)

- Since this may not be sufficient for all existing and possible future scripts, an extension mechanism has been built in, allowing two 16-bit values to represent a single character – these are called **surrogate pairs**
  - the first value in the pair is the **high surrogate**
  - the second value in the pair is the **low surrogate**

- To avoid introducing an escape character to indicate the beginning of a surrogate pair, codes in the range U+D800 to U+DFFF are reserved for use in surrogate pairs (i.e. 2048 codes are sacrificed)
  - U+D800 - U+DBFF are used for the high surrogate
  - U+DBFF - U+DFFF are used for the low surrogate

An Aside on Surrogate Pairs (continued)

- Unicode scalar value (aka "code point") N in the range 0 to $10FFFF_{16}$ is assigned to a character sequence S as follows:

$$N = U \qquad \text{If } S \text{ is a single, non-surrogate value } \langle U \rangle$$

$$N = (H - D800_{16}) * 400_{16} + (L - DC00_{16}) + 10000_{16} \qquad \text{If } S \text{ is a surrogate pair } \langle H, L \rangle$$

- The reverse mapping from a Unicode scalar value S to a surrogate pair is given by:

$$\begin{aligned} H &= (S - 10000_{16})/400_{16} + D800_{16} \text{ (high surrogate)} \\ L &= (S - 10000_{16})\%400_{16} + DC00_{16} \text{ (low surrogate)} \end{aligned}$$

- Using this mechanism Unicode can represent more than 1 million distinct characters ($10FFFF_{16} = 1{,}114{,}111_{10}$)

Character Encoding Forms: UTF-8

- UTF-8 (Unicode Transformation Format-8)
  - maps a Unicode scalar value onto 1 to 4 bytes (see table)
    - 1st byte indicates number of bytes to follow
    - one byte sufficient for ASCII code values (1..127)
    - two bytes sufficient for most non-ideographic scripts
    - four bytes needed only for surrogate pairs
  - advantages:
    - existing ASCII files are already UTF-8
    - most broadly supported encoding form today
  - disadvantages:
    - ideographic (mostly Asian) languages require 3 bytes/character – so UTF-8 encodings are larger for Asian languages than UTF-16 and most existing encodings

Scalar Value	**UTF-16**	**UTF-8 Bit Distribution**			
		1st Byte	**2nd Byte**	**3rd Byte**	**4th Byte**
000000000xxxxxxx	000000000xxxxxxx	0xxxxxxx			
00000yyyyyxxxxxx	00000yyyyyxxxxxx	110yyyyy	10xxxxxx		
zzzzyyyyyyxxxxxx	zzzzyyyyyyxxxxxx	1110zzzz	10yyyyyy	10xxxxxx	
uuuuuzzzzyyyyyyxxxxxx	110110wwwwzzzzyy+ 11011yyyyxxxxxx	11110uuu[a]	10uuzzzz	10yyyyyy	10xxxxxx

[a] Where uuuuu = wwww + 1 (to account for addition of 10000_{16} in computing scalar value of surrogate pair)

Character Encoding Schemes

- For code units wider than 1 byte, platform discrepancies mean byte order must be specified in encoding to ensure portability
- Thus, for two byte encoding forms like UTF-16, there are two possible byte orderings
  - In the UTF-16BE big-endian character encoding scheme
    `<004D 0061 0072 006B>` is serialized as
    `<00 4D 00 61 00 72 00 6B>`
  - In the UTF-16LE little-endian character encoding scheme
    `<004D 0061 0072 006B>` is serialized as
    `<4D 00 61 00 72 00 6B 00>`

Summary

- Character encoding is necessary for the representation of texts in digital form
- Standards must be agreed if such data is to be widely shared
- Many standards have emerged over the years, even for single languages and languages as relatively straightforward as English
- Developing a single multilingual encoding standard will permit
  - processing of multiple languages in one document
  - reuse of software for products dealing with multiple languages (especially Web-related software)
- Unicode is emerging as a global standard, though not without contention
- Key features of Unicode include:
  - a commitment to encode plain text only
  - attempt to unify characters across languages, modulo support for existing standards
  - use of well-defined character property tables to associate additional information with characters
  - an encoding model which clearly separates:
    - the abstract character repertoire
    - their mapping to a set of integers
    - their encoding forms + byte serialization

Summary (cont)

- Unicode model involves specifying
  - the abstract character repetoire to be encoded
  - a mapping from the characters to the integers in the range 0 - $10FFFF_{16}$
  - a mapping from integers in this range onto sequences of bytes
- Basic Unicode only uses integers in the range 0 - $FFFF_{16}$
  However, pairs of Unicode values can be used, via the **surrogate pair** mechanism, to extend the code space up to $10FFFF_{16}$
  All encodings for existing languages are in the range 0 - $FFFF_{16}$
- The most common encoding forms for Unicode values are
  - UTF-32 a fixed with encoding with one code point per 4 bytes
  - UTF-16, a fixed width encoding, which straightforwardly maps integers in the range 0 - $FFFF_{16}$ onto 2 bytes
    - UTF-16 comes in 2 flavours, UTF-16BE/LE, to support big-endian/little-endian processors
  - UTF-8, a variable width (1-4 byte) encoding which allows current ASCII-based applications/data to function transparently in a Unicode framework
- Despite its complexity, UTF-8 is currently the most widely supported encoding in programming languages (Java, Perl) and markup languages (XML)

References

- Crystal, D. The Cambridge Encyclopedia of Language. Cambridge University Press, 1987.
- Daniels, P. T. and Bright, W. eds, The World's Writing Systems. Oxford University Press, 1996.
- Steven J. Searle. "A Brief History of Character Codes in North America, Europe and Asia". Available at: `http://tronweb.super-nova.co.jp/characcodehist.html`. Site last visited 06/11/17.
- Unicode Consortium. The Unicode Standard Version 10.0.0. Available at `www.unicode.org`. Site last visited 06/11/17.
- Unicode Technical Report # 17: Character Encoding Model. Available at `www.unicode.org`. Site last visited 06/11/17.