COM3110/4115/6115:

Text Processing

Text Encoding

Rob Gaizauskas

Department of Computer Science

University of She eld

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page40) 1 / 40

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

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page1) 2 / 40

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

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page1) 3 / 40

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 de ned by language; but civilization is de ned 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

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page1) 4 / 40

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

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page1) 5 / 40

Languages and Writing Systems (cont)

Writing systems may be classi ed 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

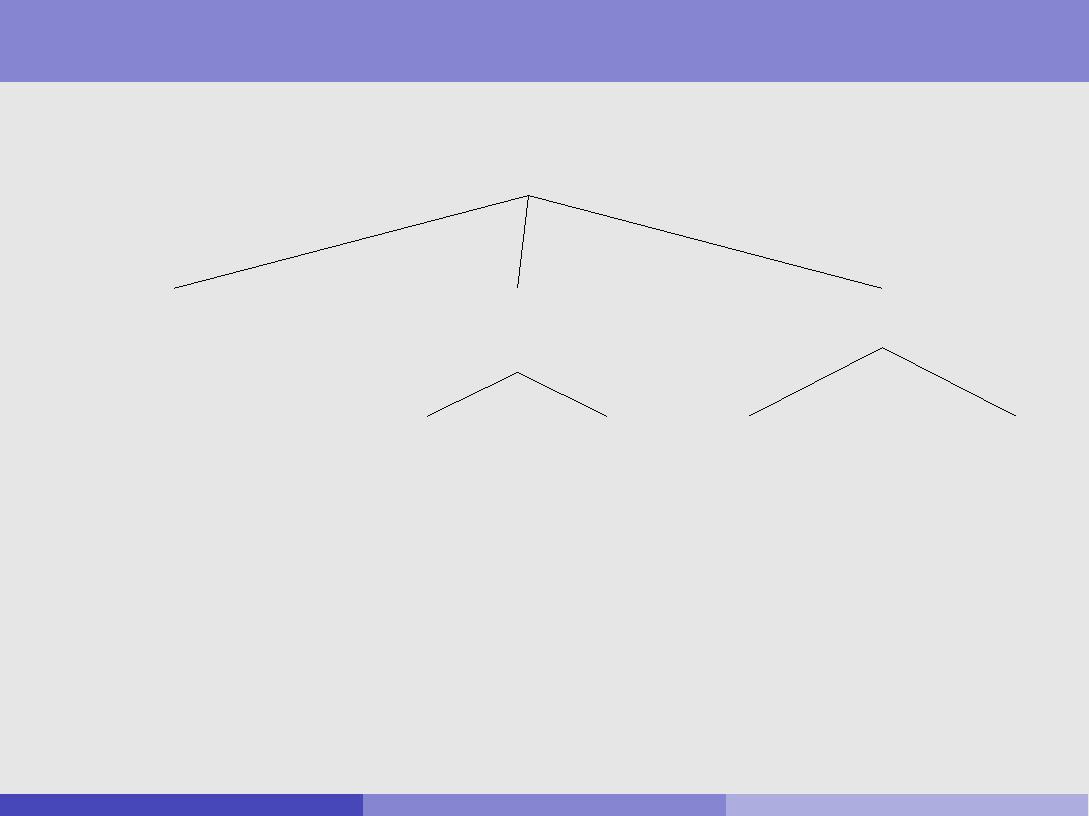
phonological systems show a clear relationship between sounds of the language and symbols

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-classi cations

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page1) 6 / 40



Taxonomy of Writing Systems

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | World Writing Systems | | | |  |  |  |  |  |  |  |
| Logographic | | Syllabic | |  |  | Alphabetic | | | |  |  |  |
| Symbols represent: words | | Symbols represent: syllables | | | | Symbols represent: phonemes | | | | | |  |
|  |  | (consonant + vowel, e.g \ka") | | | |  |  |  |  | Alphabets | |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  | |  |  |  |
| Hanzi (Chinese) | | Syllabaries | | Abugidas | | Abugidas | | Abjads | |  |
| Kanji (Japanese) | |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
| Linear B | |  |  |  |  |  |  |  |  |  |  |  |
| Ethiopic | | Devanagari | | Devanagari | | Arabic | | Latin | |  |
|  |  |  |
|  |  | Inuktitut | | Sinhala | | Sinhala | | Aramaic | | Cyrillic | |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  | Hiragana (Japanese) | | Tagalog | | Tagalog | | Hebrew | | Korean | |  |
|  |  |  |  |  |  |  |  |  |  |  |

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page1) 7 / 40

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. a

A glyph is a representation of a character or characters, as it/they is/are rendered or displayed.

E.g. A, A, A are di erent 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

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page1) 8 / 40

Characters vs Glyphs (Cont)

The relationship between glyphs and characters can be complex

for any character there may be many glyphs representing it (in di erent fonts) a single glyph may correspond to a number of characters.

E.g. \ " is typically a single glyph used to represent \f" followed by \i" in typesetting English

there may be arbitrariness { should e

be stored as two characters and rendered using two glyphs? be stored as two characters and rendered by one glyph?

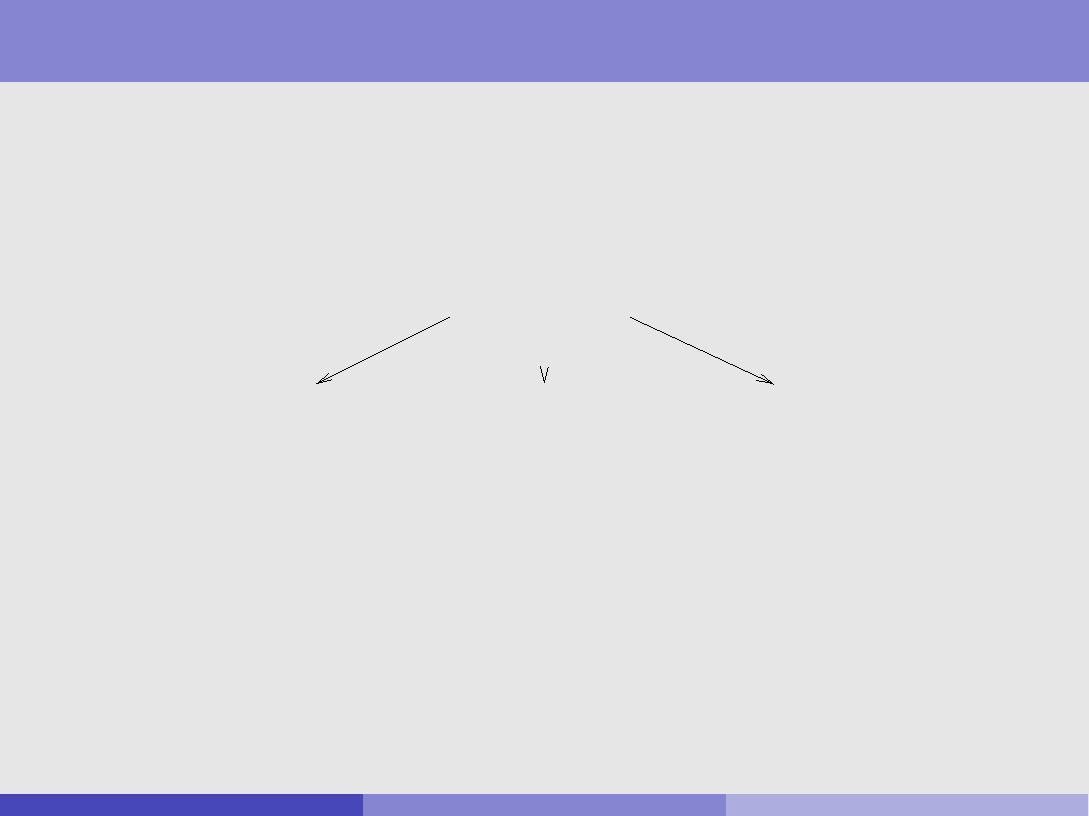
be stored as one character and rendered by one glyph?

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 nal appearance of the text is the responsibility of the rendering process

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page1) 9 / 40



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

THE CAT SAT ON THE MAT

|  |  |  |
| --- | --- | --- |
| ? | ? | ? |
|  |  |  |

0100100010011111001001001000010000100011

Various issues in designing a character encoding scheme:

Generality How many languages are you aiming to deal with?

Character Set Speci cation 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 di erent scheme? (standards)

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page1) 10 / 40

A Note on Binary, Octal & Hexadecimal Representations

One byte = 8 bits.

8 bits can represent 28 = 256 distinct values { 0 - 255. In binary notation, these range from

00000000

00000001

00000010

.

.

.

11111111

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 .

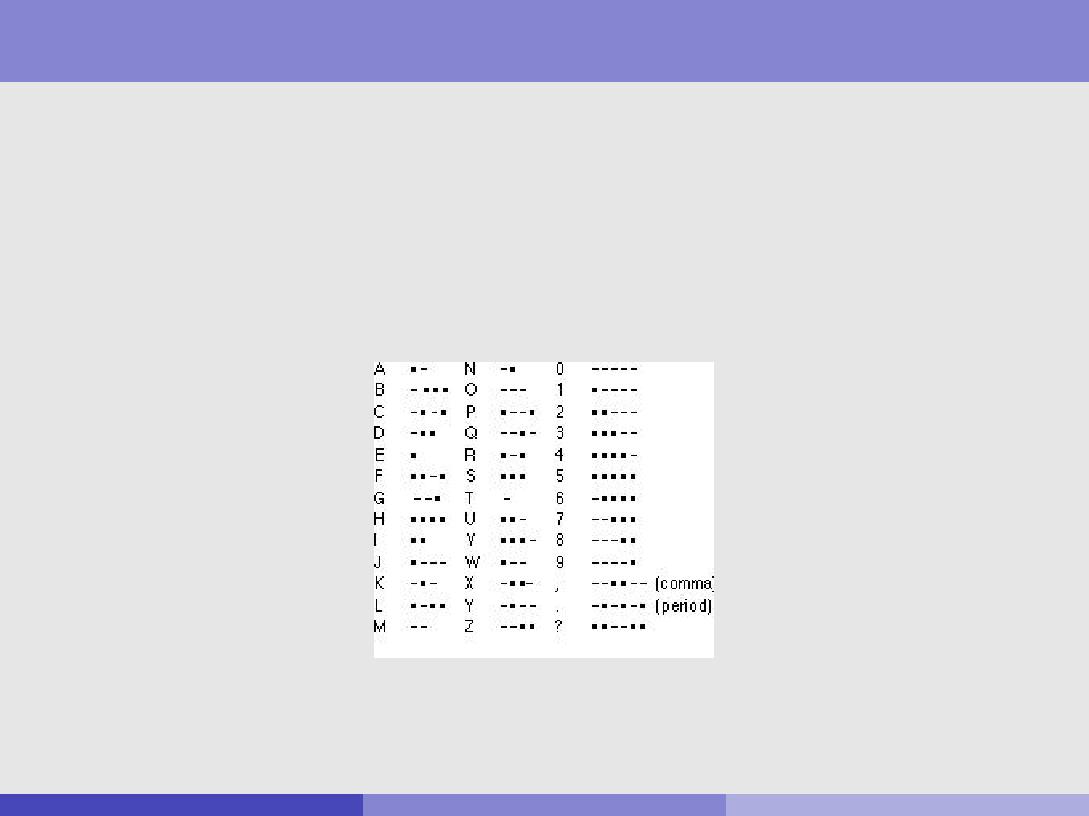
Two bytes = 16 bits, and can represent 216 = 65; 536 distinct values.

Two byte values most commonly written in hexadecimal; range from 0000 -

FFFF

d65535 =xFFFF = (15 4096) + (15 256) + (15 16) + 15

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page1) 11 / 40



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

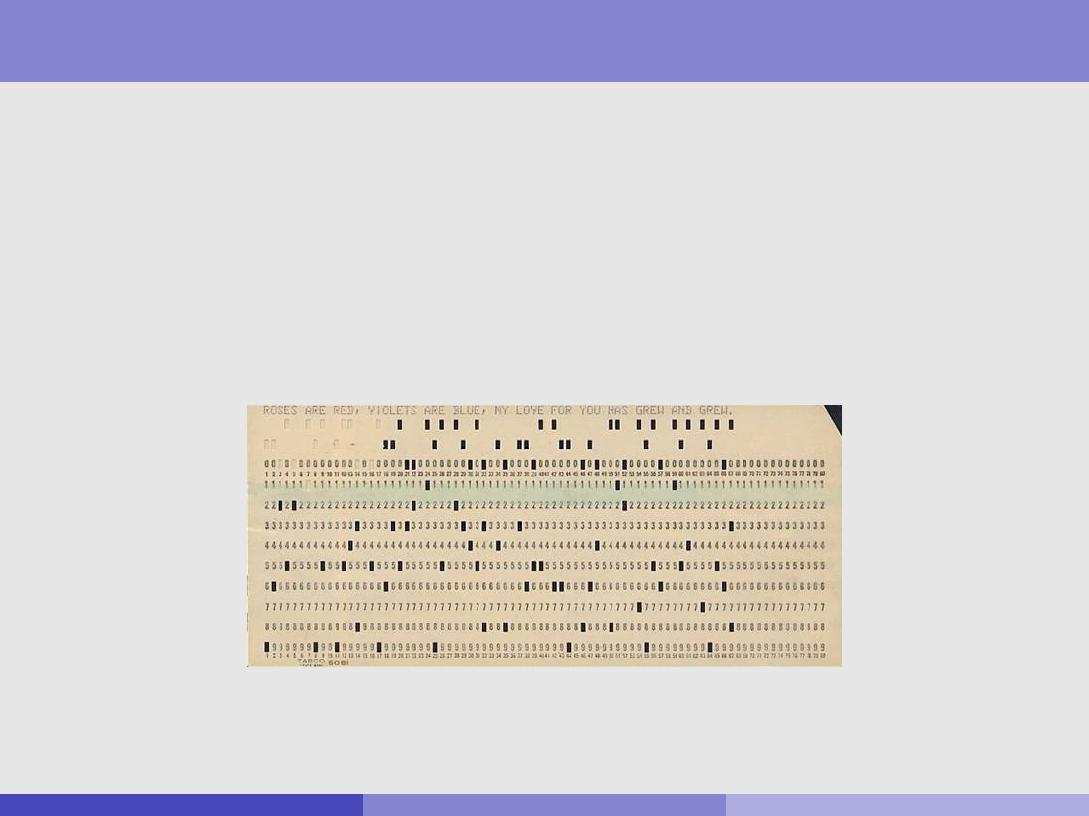
Telegraph invented by Wheatstone + Cooke (UK), 1837 Samuel Morse (US) invents simpler system + code, 1838

Variable length code, based on 2 values { dot's or dashes

Baudot (Fr) invents printing telegraph (\teleprinter"), 1874

used rst binary code for textual data { 5 bit Baudot code used a shift code to double code space; i.e., 2 25

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page1) 12 / 40



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

Hollerith and Punched Cards

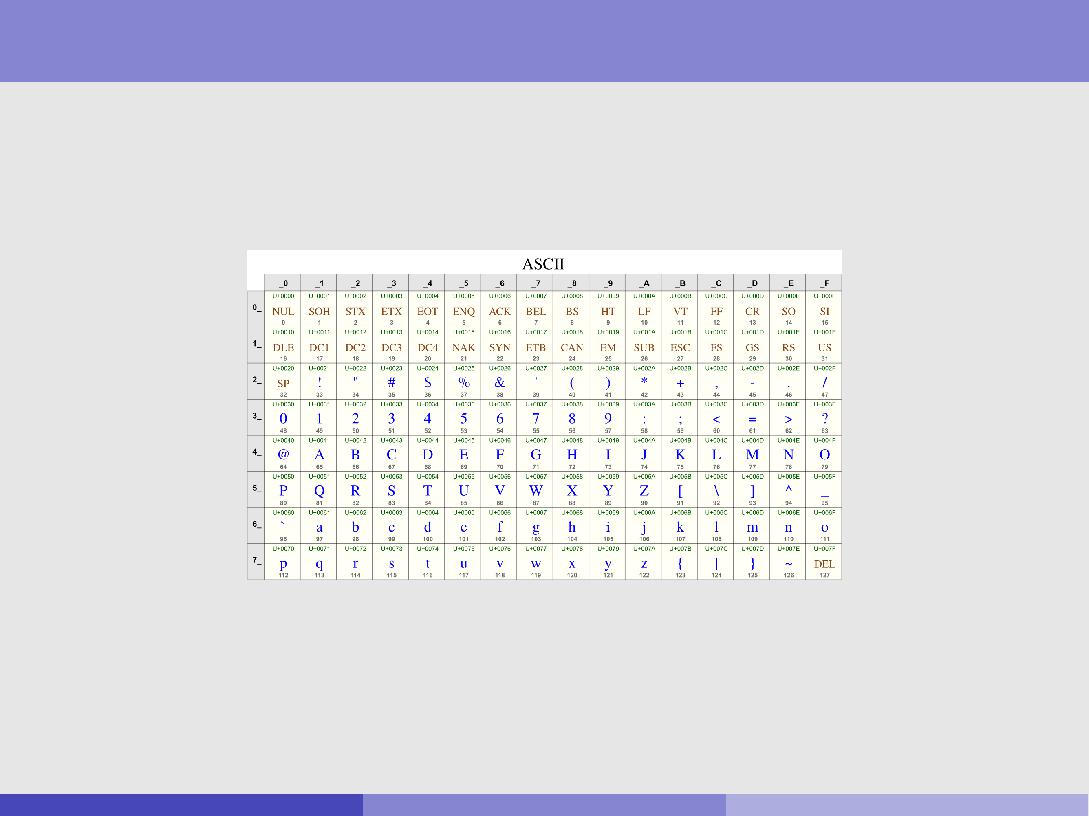
Herman Hollerith (US) invents punched card tabulating machines to process 1890 US census data

code based on holes punched in 12 (row) 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>

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page1) 13 / 40



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, nalised 1968)

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

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page1) 14 / 40

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

ISO/IEC 8859

ASCII insu cient 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-speci c 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, e orts 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

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page1) 15 / 40

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 e orts

some pioneering e orts 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 e orts 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

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page1) 16 / 40

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 insu cient; 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 insu cient and added \surrogate pairs" (see below)

Unicode and UCS have now converged, are maintained together, and are

code-for-code identical

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page1) 17 / 40

Unicode: Architectural Context { Text Processes

\The Unicode Standard is the universal character encoding standard for written characters and text. It de nes 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

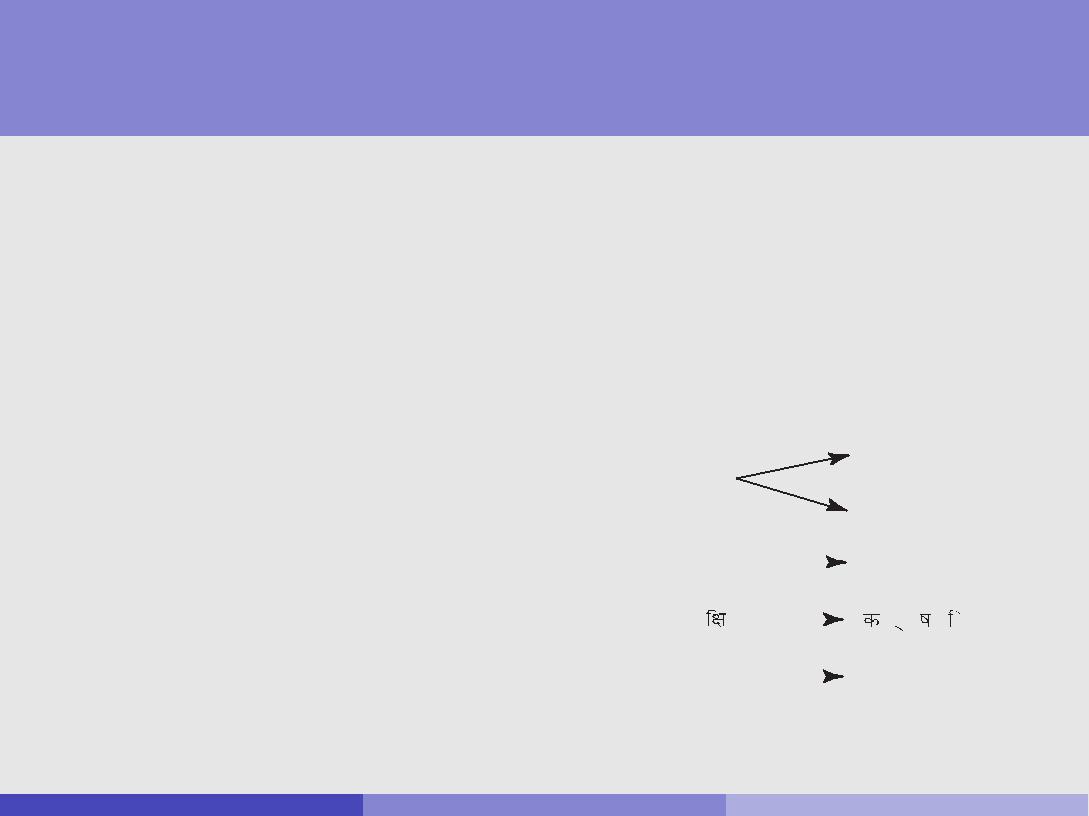
order of two strings

analysing text content for, e.g. spell checking, hyphenation, parsing morphology ( nding word roots/stems/a xes)

treating text as bulk data for, e.g., compression, transmission

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page1) 18 / 40

In contrast, a character encoding standard provides a single set of fundamental units of encoding, to which it uniquely assigns numerical code points. These units, called *assigned*



Unicode: Architectural Context { Text Elements, Code

|  |  |  |
| --- | --- | --- |
| *General Structure* | *11* | *2.1 Architectural Context* |
| Values and Text Processes |  |  |

language depend upon the specific text process; a text element for spell-checking may have different boundaries from a text element for sorting purposes. For example, in the phrase

“the quick brown fox,” the sequence “fox” is a text element for the purpose of spell-check-

Unfortunately, for each text process,ing. languages di er in what constitutes a text element.

e.g. In Spanish \ll" sorts between*characters*, \l"arethe andsmallest\m"interpretable(i.eunits.shouldofstoredtextbe.Texttreatedelementsareasthen arepre-

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | sented by a sequence of one or more characters. | | | | | | |  |  |  |  |  |  |  |  |  |
| text element), but in rendering it is best treated as two \l" elements. | | | | | | | | | | | | | | | |  |
|  | *Figure 2-1* illustrates the relationship between several different types of text elements and | | | | | | | | | | | | | | |  |
| Thus, the rst challenge in | the characters used to represent those text elements. | | | | | | |  |  |  |  |  |  |  |  |  |
|  | **Figure 2-1.** Text Elements and Characters | | | | | | | | | |  |  |  |  |  |
| designing an encoding scheme |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| for a language is to agree on the | |  | Text Elements | | | | | Characters | | | | | | |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| set of abstract characters to |  |  |  |  |  |  |  |  | Ç |  |  |  |  |  |  |  |
|  | Composite: |  | Ç |  |  |  |  |  |  |  |  |  |  |
| be encoded { those characters |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  | C | @¸ |  |  |  |  |  |  |
| that will be assigned a code |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| value. |  | Collation Unit: |  | ch |  | |  |  | c | h |  |  | (Slovak) | |  |  |
|  |  |  |  |  |  |  |
| For English this seems |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | Syllable: |  |  |  |  |  |  |  | @ |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
| straightforward (but, e.g., |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| should \A" and \a" get 2 code | | Word: |  | cat | |  |  |  | c | a |  | t |  | |  |  |
|  |  |  |  |  |
| values or 1?); other languages | The design of the character encoding must provide precisely the set of characters that | | | | | | | | | | | | | | |  |

|  |  |  |
| --- | --- | --- |
| are not so simple. | allows programmers to design applications capable of implementing a variety of text pro- |  |
| cesses in the desired languages. Therefore, the text elements encountered in most text pro- |  |
|  |  |
|  | cesses are represented as sequences of character codes. See Unicode Standard Annex #29, |  |

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page1) 19 / 40

Unicode: Design Principles

Underpinning Unicode are 10 design principles (version 10.0.0):

|  |  |
| --- | --- |
| Principle | Statement |
|  |  |
| Universality | The Unicode Standard provides a single, |
|  | universal repertoire |
|  |  |
| E ciency | Unicode text is simple to parse and process |
|  |  |
| Characters, not glyphs | Unicode encodes Characters, not glyphs |
|  |  |
| Semantics | Characters have well-de ned semantics |
|  |  |
| Plain text | Unicode characters represent plain text |
|  |  |
| Logical order | Default for memory representation is logical order |
|  |  |
| Uni cation | Unicode uni es 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 |
|  |  |

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page1) 20 / 40

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 scienti c users

out of scope:

writing systems for which insu cient 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.

E ciency { designed to allow e cient implementations

no escape characters or shift states { each character has same status

all encoding forms self-synchronising: limited backup required to nd 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

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page1) 21 / 40



Unicode: Design Principles (cont)

Characters, not Glyphs

*General Structure*Unicode distinguishes *16* *2.2* *Unicode Design Principles*

characters: smallest components of written language with semantic value

glyphs: the shapes characters can have when displayed

Unicode standard**Figure**deals**2-2**only**.**CharacterswithcharactVersuscodesGlyphs

**Glyphs** **Unicode Characters**

U+0041 latin capital letter a

U+0061 latin small letter a

U+043F cyrillic small letter pe

U+0647 arabic letter heh

U+0066 latin small letter f

* U+0069 latin small letter i

word; the glyphs in *Figure 2-2* show independent, final, initial, and medial forms. Sequences

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page1) 22 / 40



resentation), variations in typographic design of the fonts used, and formatting

Unicode: Design Principlesinformation(point(cont)size,superscript, subscript, and so on). The results on screen or paper can differ considerably from the prototypical shape of a letter or character, as shown in

*Figure 2-3*.

|  |  |  |
| --- | --- | --- |
|  | **Figure 2-3.** Unicode Character Code to Rendered Glyphs |  |
| Characters, not Glyphs (cont) |  |

Glyph shapes, how they are selected and rendered are the responsibility of font vendors (font = set of glyphs)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | Text Character Sequence | | | | | | | | |  |  |  |  |
|  |  |  | 0000 1001 0010 1010 | | | | |  |  |  |  |  |  |  |  |
|  |  |  | 0000 1001 0100 0010 | | | | |  |  |  |  |  |  |  |  |
|  |  |  | 0000 1001 0011 0000 | | | | |  |  |  |  |  |  | Font |  |
|  |  |  |  |  |  |  |  |  |  |  |  | (Glyph Source) | | |  |
|  |  |  | 0000 1001 0100 1101 | | | | |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  | 0000 1001 0010 0100 | | | | |  |  |  |  |  |  |  |  |
|  |  |  | 0000 1001 0011 1111 | | | | |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  | Text | | |  |  |  |  |  |  |
|  |  |  |  |  |  | Rendering | | | |  |  |  |  |  |  |
|  |  |  |  |  |  | Process | | | |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

For the Latin script, this relationship between character code sequence and glyph is rela-

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page1) 23 / 40

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-speci c, 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)

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page1) 24 / 40



acters in one direction, such as from left to right, right to left, or top to bottom. In other

circumstances, text is displayed or printed in an order that differs from a single linear pro-Unicode:gressionDesign.Someof thePrinciplesclearestexamples are(cont)situations where a right-to-left script (such as

Arabic or Hebrew) is mixed with a left-to-right script (such as Latin or Greek). For exam-ple, when the text in *Figure 2-4* is ordered for display the glyph that represents the first

LogicalcharacterOrderofthe English text appears at the left. The logical start character of the Hebrew

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | text, however, is represented by the Hebrew glyph closest to the right margin. The succeed- | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |  |
| Unicode text is stored in logical order { generally the same as the order it | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |  |
| ing Hebrew glyphs are laid out to the left. | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |  |
|  | would be input via a keyboard | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |  |
|  |  |  |  |  |  |  |  |  |  | **Figure 2-4.** Bidirectional Ordering | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |  |
| In some cases this may di er from display order | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Directionality property of characters generally su cient to render plain text, In logical order, numbers are encoded with most sig ificant digit first, but are displayed in

butdifferentnot writingalwaysdirections. As shown in *Figure 2-5* these writing directions do not always correspondUnicodetothedoeswritingcontaindirectioncharacteofthesu**r**roundingstospecifytext. Tchangefirstexampleofdirectionshows N’Ko, a right-to-left script with digits that also render right to left. Examples 2 and 3 show

Hebrew and Arabic, in which the numbers are rendered left to right, resulting in bidirec-

Uni cation

tional layout. In left-to-right scripts, such as Latin and Hiragana and Katakana (for Japa-

duplicatenese),numbersencodingfollowtheofpredominantthesame leftcharacter-to-rightdirectionacross oflanguagesthescript,asisshownavoidedin in general,Examples e4.andg. 5. When Japanese is laid out vertically, numbers are either laid out verti-

cally or may be rotated clockwise 90 degrees to follow the layout direction of the lines, as punctuation

shown in Example 6.

\Y" (French i grecque, German ypsilon, English wye all represented by U+0057)

|  |  |  |
| --- | --- | --- |
|  | **Figure 2-5.** Writing Direction and Numbers |  |
| sometimes this principle is violated to support compatibility with existing |  |
|  | standards |  |

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page1) 25 / 40

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 denti ers 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 di erent base standards (national, international, vendor-speci c)

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page1) 26 / 40



Unicode: Encoding Model

The Unicode model may be rst 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 10FFFF16, comprising 1,114,112 code points

their encoding forms + byte serialization

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page1) 27 / 40

Unicode: Encoding Model (cont)

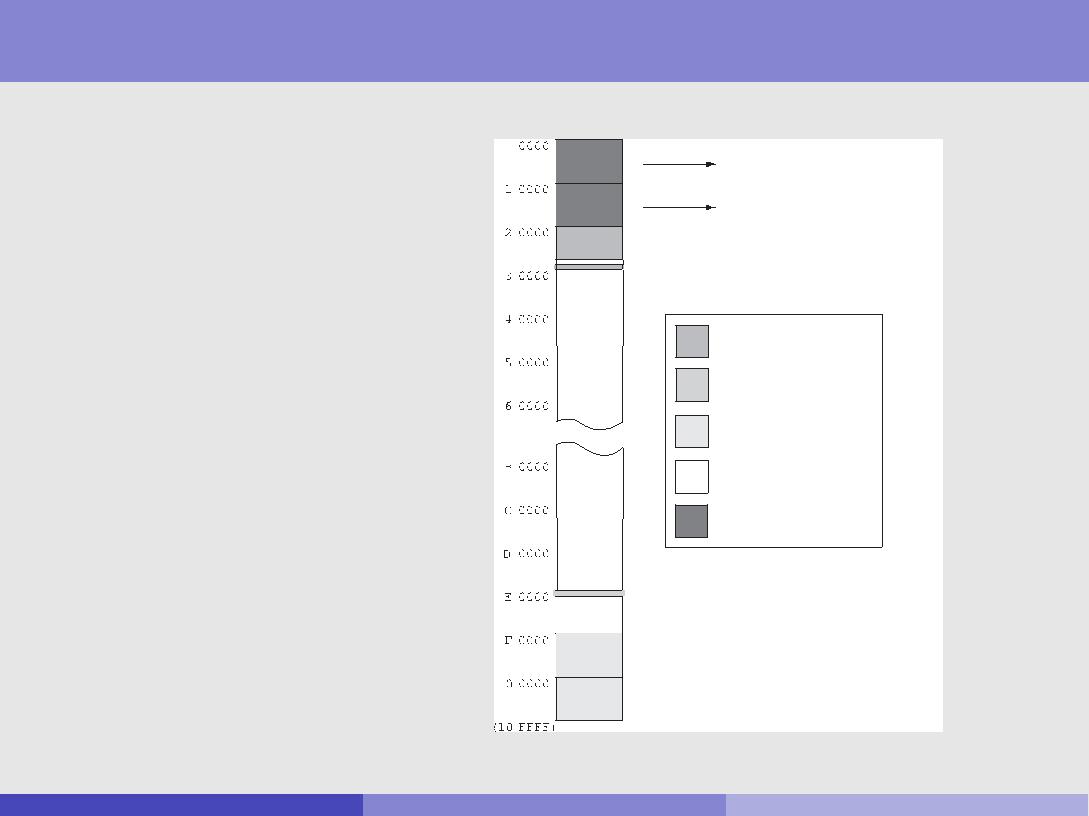
Encoded characters are alternate ways the same text element may be assigned an integer code in the Unicode code value space

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

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page1) 28 / 40



This section provides a more detailed summary of the way characters are allocated in the Unicode: Allocation AreasUnicode Standard. *Figure 2-13* gives an overall picture of the allocation areas of the Uni-code Standard, with an emphasis on the identities of the planes. The following subsections

discuss the allocation details for specific planes.

Unicode codespace of 10FFFF16 codepoints is divided into planes each consisting of FFFF (64k decimal) code points

The rst plane (codepoints 0-FFFF), is called the basic multilingual plane(BMP)

**Figure 2-13.** Unicode Allocation

For allocations on Plane 0 (BMP) and

Plane 1 (SMP), see the detail figures

CJK Unified Ideographs Extensions

CJK Compatibility Ideographs Supplement

Graphic

Format or Control

Private Use

Reserved

Detail on other figures

Tags and Ideographic Variation Selectors

Supplementary Private Use Area-A

Supplementary Private Use Area-B

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page1) 29 / 40



*Figure 2-14* shows the Basic Multilingual Plane (BMP) in an expanded format to illustrate

the allocation substructure of that plane in more detail. This section describes each alloca-

Unicode: Allocation Areastionarea,(cont)intheorderoftheir location on the BMP.

**Figure 2-14.** Allocation on the BMP

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

0000

0900

2000

2C00

3400

A000

AC00

D800

E000

F900

(FFFF)

0000-00FF ASCII & Latin-1 Compatibility Area

0100-058F General Scripts Area

0590-08FF General Scripts Area (RTL)

0900-1FFF General Scripts Area

2000-2BFF Punctuation and Symbols Area

2C00-2DFF General Scripts Area

2E00-2E7F Supplemental Punctuation Area

2E80-33FF CJK Miscellaneous Area

3400-9FFF CJKV Unified Ideographs Area

(not to scale)

A000-ABFF General Scripts Area (Asia & Africa)

AC00-D7FF Hangul Syllables Area

D800-DFFF Surrogate Codes

E000-F8FF Private Use Area

F900-FFFF Compatibility and Specials Area

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page1) 30 / 40

Encoding Forms and Encoding Schemes

Once a mapping from an abstract character set to a set of integers has been de ned two further mappings need to be speci ed

Character Encoding Form { a mapping from a set of integers to a set of sequences of code units of speci ed 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)

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page1) 31 / 40

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 xed length code units (bytes)

Note: a 32-but code could handle up to 4.29 billion distinct characters

disadvantages:

les 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 le even if it only contains Latin text.

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page1) 32 / 40

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:

les 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 le even if it only contains Latin text.

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page1) 33 / 40

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 su cient 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 rst 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 sacri ced)

U+D800 - U+DBFF are used for the high surrogate U+DBFF - U+DFFF are used for the low surrogate

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page1) 34 / 40

An Aside on Surrogate Pairs (continued)

Unicode scalar value (aka \code point") N in the range 0 to 10FFFF16 is assigned to a character sequence S as follows:

N = U If S is a single, non-surrogate value hUi

N = (H D80016) 40016 + If S is a surrogate pair hH; Li

(L DC0016) + 1000016

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

* = (S 1000016)=40016 + D80016 (high surrogate)
  + = (S 1000016)%40016 + DC0016 (low surrogate)

Using this mechanism Unicode can represent more than 1 million distinct characters ( 10FFFF16 = 1,114,11110)

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page1) 35 / 40

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 su cient for ASCII code values (1..127)

two bytes su cient for most non-ideographic scripts four bytes needed only for surrogate pairs

advantages:

existing ASCII les 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+ | 11110uuua | 10uuzzzz | 10yyyyyy | 10xxxxxx |
|  | 11011yyyyxxxxxx |  |  |  |  |

* Where uuuuu = wwww + 1 (to account for addition of 1000016 in computing scalar value of surrogate pair)

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page1) 36 / 40

Character Encoding Schemes

For code units wider than 1 byte, platform discrepancies mean byte order must be speci ed 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>

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page1) 37 / 40

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-de ned 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

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page1) 38 / 40

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 - 10FFFF16 a mapping from integers in this range onto sequences of bytes

Basic Unicode only uses integers in the range 0 - FFFF16 However, pairs of Unicode values can be used, via the surrogate pair mechanism, to extend the code space up to 10FFFF16

All encodings for existing languages are in the range 0 - FFFF16 The most common encoding forms for Unicode values are

UTF-32 a xed with encoding with one code point per 4 bytes

UTF-16, a xed width encoding, which straightforwardly maps integers in the range 0 - FFFF16 onto 2 bytes

UTF-16 comes in 2 avours, 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)

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page1) 39 / 40

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.

Rob Gaizauskas (U. She eld) [COM3110/4115/6115: Text Processing](#page1) 40 / 40