A Crash Course in Unicode for C++ Developers

Steve Downey

C++ now
2021
MAY 2-7
Aspen, Colorado, USA

ABSTRACT

This talk will give you a crash course so you can understand:

ABSTRACT

This talk will give you a crash course so you can understand:

- what code units, code points, and grapheme clusters are.

ABSTRACT

This talk will give you a crash course so you can understand:

- what code units, code points, and grapheme clusters are.
- how unicode encodings work.

ABSTRACT

This talk will give you a crash course so you can understand:

- what code units, code points, and grapheme clusters are.
- how unicode encodings work.
- how decode and encode algorithms work.

ABSTRACT

This talk will give you a crash course so you can understand:

- what code units, code points, and grapheme clusters are.
- how unicode encodings work.
- how decode and encode algorithms work.
- what Unicode normalization does and what the various forms are for.

ABSTRACT

This talk will give you a crash course so you can understand:

- what code units, code points, and grapheme clusters are.
- how unicode encodings work.
- how decode and encode algorithms work.
- what Unicode normalization does and what the various forms are for.
- what the various Unicode algorithms for text processing are.

  This will give you the tools to understand how not to break your users' text, at least not too much or often.

PRIMUM NON NOCERE: FIRST, DO NO HARM

PRIMUM NON NOCERE: FIRST, DO NO HARM

- The best text handling is not to.

PRIMUM NON NOCERE: FIRST, DO NO HARM

- The best text handling is not to.
- Pushing error handling to the edges is best practice.

PRIMUM NON NOCERE: FIRST, DO NO HARM

- The best text handling is not to.
- Pushing error handling to the edges is best practice.
- std::u8string is not like Python3's string.

THE BASIC PARTS

"CHARACTERS" AREN'T VERY BASIC

What the fundamental units of writing are vary considerably from language to language.

English is one of the simplest, and we have made it simpler in the last century.

Unicode tries to avoid talking about "Characters"

CHARACTER

(1) The smallest component of written language that has semantic value; refers to the abstract meaning and/or shape, rather than a specific shape (see also glyph), though in code tables some form of visual representation is essential for the reader's understanding.

(2) Synonym for abstract character.

(3) The basic unit of encoding for the Unicode character encoding.

(4) The English name for the ideographic written elements of Chinese origin.

http://unicode.org/glossary/#character

ABSTRACT CHARACTER

A unit of information used for the organization, control, or representation of textual data.

http://unicode.org/glossary/#abstract_character

ABSTRACT CHARACTER

A unit of information used for the organization, control, or representation of textual data.

- Not a glyph

http://unicode.org/glossary/#abstract_character

ABSTRACT CHARACTER

A unit of information used for the organization, control, or representation of textual data.

- Not a glyph
- Not a grapheme

http://unicode.org/glossary/#abstract_character

CODE UNITS

CODE UNITS

- `char`

CODE UNITS

- `char`
- `wchar_t`

CODE UNITS

- `char`
- `wchar_t`
- octet

CODE UNITS

- `char`
- `wchar_t`
- octet
- WORD

  What the computer works in at the basic level.

CODE POINTS AND SCALAR VALUES

Unicode assigns each abstract character a `code point`, a number from 0 to 0x10FFFF.

Some code points are currently unassigned, some are guaranteed never to be.

There is a range used for encoding values outside the 16 bit range in UTF-16.

`Scalar Values` excludes that range, so they are code points 0 to 0xD7FF and 0xE000 to 0x10FFFF

http://unicode.org/glossary/#code_point

http://unicode.org/glossary/#unicode_scalar_value

GRAPHEME CLUSTERS, EXTENDED GRAPHEME CLUSTERS

Grapheme clusters are roughly what most users will think of as a letter. A base together with a number of non-spacing marks, such as combining diacritic marks. Extended grapheme clusters may include some spacing marks. These were added because users treat some clusters with spaces as units of text. Text is hard.

G with diaeresis : g̈

Devanagari ssi : षि

ENCODINGS

ENCODINGS

- "The Unicode Standard is the universal character encoding standard for written characters and text."
- Encodings are how the abstract notions of characters and text are made concrete.
- Unicode has defined several over the years.

UTF-8

The Encoding

UTF-8 IS UNREASONABLY GOOD

UTF-8 IS UNREASONABLY GOOD

- C string safe

UTF-8 IS UNREASONABLY GOOD

- C string safe
- No aliasing

UTF-8 IS UNREASONABLY GOOD

- C string safe
- No aliasing
- Self syncing

UTF-8 IS UNREASONABLY GOOD

- C string safe
- No aliasing
- Self syncing
- Single errors lose one character

UTF-8 IS UNREASONABLY GOOD

- C string safe
- No aliasing
- Self syncing
- Single errors lose one character
- ASCII compatible

UTF-8 IS UNREASONABLY GOOD

- C string safe
- No aliasing
- Self syncing
- Single errors lose one character
- ASCII compatible
- Start is easy to find

MECHANISM

Variable width multibyte encoding where the first byte encodes the number of bytes used.

UTF-8 BITS

Table 3-6. UTF-8 Bit Distribution

Scalar Value	Range	First Byte	Second Byte	Third Byte	Fourth Byte
00000000 0xxxxxxx	U+0000 - U+007F	0xxxxxxx			
00000yyy yyxxxxxx	U+0080 - U+077F	110yyyyy	10xxxxxx		
zzzzyyyy yyxxxxxx	U+0800 - U+FFFF	1110zzzz	10yyyyyy	10xxxxxx	
000uuuuu zzzzyyyy yyxxxxxx	U+10000 - U+1FFFFF	11110uuu	10uuzzzz	10yyyyyy	10xxxxxx

UTF-8 BYTES

Table 3-7. Well-Formed UTF-8 Byte Sequences

Code Points	First Byte	Second Byte	Third Byte	Fourth Byte
U+0000..U+007F	00..7F			
U+0080..U+07FF	C2..DF	80..BF		
U+0800..U+0FFF	E0	A0..BF	80..BF	
U+1000..U+CFFF	E1..EC	80..BF	80..BF	
U+D000..U+D7FF	ED	80..9F	80..BF	
U+E000..U+FFFF	EE..EF	80..BF	80..BF	
U+10000..U+3FFFF	F0	90..BF	80..BF	80..BF
U+40000..U+FFFFF	F1..F3	80..BF	80..BF	80..BF
U+100000..U+10FFFF	F4	80..8F	80..BF	80..BF

UTF-16

We thought 16 bits were enough for anyone

STILL IN USE

- Windows
- Java

UTF-16 BITS

Table 3-5. UTF-16 Bit Distribution

Scalar Value	UTF-16
xxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxx
000uuuuuxxxxxxxxxxxxxxxx	110110wwwwxxxxxx 110111xxxxxxxxxx

Note: wwww = uuuuu - 1

SURROGATE PAIRS

- The code point range D800..DFFF is used to encode U+10000..U+10FFFF
- There should never be a lone value; always a pair

UTF-32

Simple and never used

UCS-2, UCS-4

The 2 and 4 byte forms before UTF.

WTF-8

WTF-8 (Wobbly Transformation Format – 8-bit)

WTF-16

- JavaScript and Windows file systems don't enforce well formed UTF-16.
- Lone surrogate values happen - they were legal in ancient times, before surrogates were invented.

WTF-8

- Encoding WTF-16 into generalized UTF-8
- Ignore the requirement that only scalar values be encoded

ENCODING/DECODING

ENCODERS AND DECODERS

ENCODERS AND DECODERS

- Encoders take text and output octets

ENCODERS AND DECODERS

- Encoders take text and output octets
- Decoders take octets and output text

ENCODERS AND DECODERS

- Encoders take text and output octets
- Decoders take octets and output text
- By text I mean scalar values

UTF-8

The encoding scheme used for UTF-8 is the bytes are ordered exactly as the form says.

Byte 1 of a 4 byte sequence comes first, followed by 2, 3, and 4

UTF-16BE, UTF-16LE, BYTE ORDER MARKS

BYTE ORDER MARK

- U+FEFF is a valid character code
- U+FFFE is a non character
- We can use these to distinguish the order of octets for 16 bit value
- Used to be used to signal Unicode in UTF-8. – Don't

UTF-16

Choices:

- UTF-16 w/BOM or implied
- UTF-16 Big Endian
- UTF-16 Little Endian

UTF-32

If anyone used it, you could have LE and BE versions and BOMS

LEGACY

Everything before Unicode

SINGLE BYTE

Simple table driven

- Windows 1252, 125x
- ISO-8859-x
- KOI8-R and u
- EBCIDIC
- Private

MULTIBYTE

State machines with "shift" states

- GBK
- Big5
- ISO-2022-JP
- Shift-JIS

TRANSCODING

Unicode is Universal

- Connect an decoder to an encoder -> transcoder
- Short circuits are common

NORMALIZATION, OR THERE'S MORE THAN ONE WAY TO WRITE THAT

Combining characters mean that the same visual text may have more than one representation.

There might be a pre-combined form.

There might be more than one diacritic.

Still - we want to be able to tell if text is "the same."

CANONICAL EQUIVALENCE AND COMPATIBLE EQUIVALENCE

Canonical equivalence means essentially equivalent. Compatible equivalence means some information might be lost

CANONICAL EQUIVALENCE

Å

- Latin Capital Letter A with Ring Above U+00C5
- Angstrom Sign U+212B
- Latin Capital Letter A U+0041 + Combining Ring Above U+030A

COMPATIBLE EQUIVALENCE

Characters		Equivalent To
¼	→	1/4
[NBSP]	→	[SPACE]
i⁹	→	i9
i₉	→	i9
ℌ	→	H

DECOMPOSED AND COMPOSED

Particularly for latin languages, Unicode defines code points for pre-combined characters, such as Latin Capital Letter A with Ring Above before.

There is a choice as whether to prefer composed or decomposed code points.

IMEs tend to produce composed characters.

NFD, NFC, NFKD, NFKC

Form	Description
Normalization Form D (NFD)	Canonical Decomposition
Normalization Form C (NFC)	Canonical Decomposition, followed by Canonical Composition
Normalization Form KD (NFKD)	Compatibility Decomposition
Normalization Form KC (NFKC)	Compatibility Decomposition,followed by Canonical Composition

WHY USE WHICH

WHY USE WHICH

- Use compatible for applications like search.

WHY USE WHICH

- Use compatible for applications like search.
- Use canonical for applications that need strong equality.
  - C++ identifier equivalence will be NFC.

WHY USE WHICH

- Use compatible for applications like search.
- Use canonical for applications that need strong equality.
  - C++ identifier equivalence will be NFC.
- Decomposed makes it easier to find base characters, ignoring diacritics.

WHY USE WHICH

- Use compatible for applications like search.
- Use canonical for applications that need strong equality.
  - C++ identifier equivalence will be NFC.
- Decomposed makes it easier to find base characters, ignoring diacritics.
- All normalization risks loss of information, NFC is least risky.

WHY USE WHICH

- Use compatible for applications like search.
- Use canonical for applications that need strong equality.
  - C++ identifier equivalence will be NFC.
- Decomposed makes it easier to find base characters, ignoring diacritics.
- All normalization risks loss of information, NFC is least risky.
- W3C has pragmatic recommendations
  - https://www.w3.org/TR/charmod-norm/#unicodeNormalization

SUMMARY ALGORITHM

- Fully decompose
  - Canonical decomposition for C
  - Compatible decomposition for K
- For each the order of decomposed code points is defined
- Combining characters are re-ordered according to rules

For NFD or NFKD done

- Recompose with the Canonical Composition Algorithm
  - Replace pairs that have a canonical composition with that composite

TESTING NORMALIZATION

It's much faster and simpler to test if a string is normalized. For every code point the Unicode Database has a property `quick_check` for each normalization form.

QUICK CHECK

Table 9. Description of Quick_Check Values

Values	Abbr	Description
NO	N	The code point cannot occur in that Normalization Form.
YES	Y	The code point is a starter and can occur in the Normalization Form. In addition, for NFKC and NFC, the character may compose with a following character, but it never composes with a previous character.
MAYBE	M	The code point can occur, subject to canonical ordering, but with constraints. In particular, the text may not be in the specified Normalization Form depending on the context in which the character occ

QUICK CHECK CODE

```
int quickCheck(std::u32string_view source) {
    short lastCanonicalClass = 0;
    int   result             = YES;
    for (int i = 0; i < source.size(); ++i) {
        char32_t ch             = source[i];
        short    canonicalClass = getCanonicalClass(ch);
        if ((lastCanonicalClass > canonicalClass) && (canonicalClass != 0))
            return NO;
        int check = isAllowed(ch);
        if (check == NO) return NO;
        if (check == MAYBE) result = MAYBE;
        lastCanonicalClass = canonicalClass;
    }
    return result;
}
```

https://unicode.org/reports/tr15/#Detecting_Normalization_Forms

CHARACTER PROPERTIES

- getCanonicalclass and isAllowed look up properties in the Unicode Database
- isAllowed checks Derived Normalization Properties for the NF we're checking
- getCanonicalClass gets the combining class, an ordered enum of how characters combine

STREAM-SAFE TEXT FORMAT

Full normalization has some pathologies where long look-ahead and N^2 reads are needed. The example in the standard is a digit, followed by 10,000 umlauts, followed by one dot-below.

The dot-below should be ordered before the umlauts, but may not be available in an IO buffer.

The Unicode standard provides for a stream-safe format where no more than 30 characters need to be examined, and combining grapheme joiner is U+034F used to mark boundaries.

THE UNICODE CHARACTER DATABASE

Unicode is much more than just a set of characters and encodings.

Unicode characters have a rich set of semantics and properties associated with them.

Those are cataloged in the UCD, the Unicode Character Database.

A collection of data files that have the code points, their names, and simple and derived properties of the defined characters.

Derived properties are based on other properties by rule, however stability and correctness concerns means they are also directly defined.

UCD FILES

Well defined directory layout with text files that are intended to be machine processable.
XML files with equivalent information.

UNICODEDATA.TXT

Name
General_Category
Canonical_Combining_Class
Bidi_Class
"Decomposition_Type Decomposition_Mapping"
"Numeric_Type Numeric_Value"
Bidi_Mirrored
Unicode_1_Name (Obsolete as of 6.2.0)
ISO_Comment (Obsolete as of 5.2.0; Deprecated and Stabilized as of 6.0.0)
Simple_Uppercase_Mapping
Simple_Lowercase_Mapping
Simple_Titlecase_Mapping

UNICODEDATA.TXT (SOME LETTERS)

https://www.unicode.org/Public/UCD/latest/ucd/UnicodeData.txt

```
0041;LATIN CAPITAL LETTER A;Lu;0;L;;;;;N;;;;0061;
0042;LATIN CAPITAL LETTER B;Lu;0;L;;;;;N;;;;0062;
0043;LATIN CAPITAL LETTER C;Lu;0;L;;;;;N;;;;0063;
...
0061;LATIN SMALL LETTER A;Ll;0;L;;;;;N;;;0041;;0041
0062;LATIN SMALL LETTER B;Ll;0;L;;;;;N;;;0042;;0042
0063;LATIN SMALL LETTER C;Ll;0;L;;;;;N;;;0043;;0043
...
00C0;LATIN CAPITAL LETTER A WITH GRAVE;Lu;0;L;0041 0300;;;;N;LATIN CAPITAL LETTER A GRAVE;;;00E0;
00C1;LATIN CAPITAL LETTER A WITH ACUTE;Lu;0;L;0041 0301;;;;N;LATIN CAPITAL LETTER A ACUTE;;;00E1;
...
00E0;LATIN SMALL LETTER A WITH GRAVE;Ll;0;L;0061 0300;;;;N;LATIN SMALL LETTER A GRAVE;;00C0;;00C0
00E1;LATIN SMALL LETTER A WITH ACUTE;Ll;0;L;0061 0301;;;;N;LATIN SMALL LETTER A ACUTE;;00C1;;00C1
```

UNICODEDATA.TXT (SOME DIGITS)

```
0030;DIGIT ZERO;Nd;0;EN;;0;0;0;N;;;;;
0031;DIGIT ONE;Nd;0;EN;;1;1;1;N;;;;;
0032;DIGIT TWO;Nd;0;EN;;2;2;2;N;;;;;
0033;DIGIT THREE;Nd;0;EN;;3;3;3;N;;;;;
...
00B2;SUPERSCRIPT TWO;No;0;EN;<super> 0032;;2;2;N;SUPERSCRIPT DIGIT TWO;;;;
00B3;SUPERSCRIPT THREE;No;0;EN;<super> 0033;;3;3;N;SUPERSCRIPT DIGIT THREE;;;;
...
0660;ARABIC-INDIC DIGIT ZERO;Nd;0;AN;;0;0;0;N;;;;;
0661;ARABIC-INDIC DIGIT ONE;Nd;0;AN;;1;1;1;N;;;;;
0662;ARABIC-INDIC DIGIT TWO;Nd;0;AN;;2;2;2;N;;;;;
...
1FBF0;SEGMENTED DIGIT ZERO;Nd;0;EN;<font> 0030;0;0;0;N;;;;;
1FBF1;SEGMENTED DIGIT ONE;Nd;0;EN;<font> 0031;1;1;1;N;;;;;
```

DERIVEDCOREPROPERTIES.TXT

Code points, or code point ranges with a particular property, followed by a comment describing the code points by their general category, how many are in the range, and what their names are.

DERIVEDCOREPROPERTIES.TXT (XID_START)

https://www.unicode.org/Public/UCD/latest/ucd/DerivedCoreProperties.txt

```
# Derived Property: XID_Start
#  ID_Start modified for closure under NFKx
#  Modified as described in UAX #15
#  NOTE: Does NOT remove the non-NFKx characters.
#        Merely ensures that if isIdentifer(string) then isIdentifier(NFKx(string))
#  NOTE: See UAX #31 for more information

0041..005A    ; XID_Start # L&  [26] LATIN CAPITAL LETTER A..LATIN CAPITAL LETTER Z
0061..007A    ; XID_Start # L&  [26] LATIN SMALL LETTER A..LATIN SMALL LETTER Z
00AA          ; XID_Start # Lo       FEMININE ORDINAL INDICATOR
00B5          ; XID_Start # L&       MICRO SIGN
00BA          ; XID_Start # Lo       MASCULINE ORDINAL INDICATOR
```

ALGORITHMS

Many problems with text handling have standard imperfect solutions. Be aware of these so as not to reinvent worse wheels.

BIDIRECTIONAL

Some text is written right to left. In those scripts digits are still layed out left to right. Mixed language text is common also. The BiDi algorithm describes how to break apart and format mixed, or bidirectional text.

HIGH LEVEL

- Break into paragraphs
- Identify character types and their embedding
- Resolve embedding levels
- Reorder the lines

LINE BREAKING

A.K.A. Word Wrapping.

Produces a list of "break opportunities" for a given text.

Classifies characters as mandatory breaks, optional break before, optional break after, prohibiting breaks, or kinds of breaks, and characters that must not be broken.

The only surprising thing is how many reasonable ways and places there are to separate lines.

https://www.unicode.org/reports/tr14/

TEXT SEGMENTATION

Separating text into significant user perceived elements:

- User perceived characters
- Words
- Sentences

The default algorithms can also be tailored, or extended with dictionary information, such as for Chinese or Japanese.

https://www.unicode.org/reports/tr29/

GRAPHEME CLUSTER BOUNDARIES

User perceived characters.

The default units for word and sentence boundaries, and relevant for line breaks.

Grapheme clusters can be detected by Unicode regular expressions that are straightforward to generate.

At least once someone else has gone to all of the classification work.

Less straightforward for words and sentences, but possible.

WORD BOUNDARIES

```
The quick ("brown") fox can't jump 32.3 feet, right?
```

Seach term	Match
brown	Y
brow	N
"brown"	Y
("brown")	Y
␣("brown")␣	Y

https://www.unicode.org/reports/tr29/#Word_Boundaries

SENTENCE BOUNDARIES

Complicated because text can embed itself, making it difficult to analyze.

"Is this a sentence?" Steve asked.

Nonetheless there are patterns that people will accept as reasonable.

The Unicode Database provides properties that classify characters as likely sentence breaks, and defines likely sentences by pairs of characters with those properties.

https://www.unicode.org/reports/tr29/#Sentence_Boundaries

THE FUTURE FOR C++

C++ 20	char8_t
C++ 23 *	Unicode Identifiers
	Literal Encoding
	Portable Source Code
	Encoding / Decoding Ranges *
	Text *
C++ 26 *	Text
	Algorithms w/Ranges

* The future is uncertain and the end is always near.