Mini-Lecture on Character Sets and Unicode

Godmar Back

Virginia Tech

August 30, 2022

VIRGINIA TECH

Motivation

- Character sets are easily one of the most confusing aspects of writing application code and interacting with computer systems
- Examples of where understanding of character sets is necessary include
  - Web servers/web applications (form processing, HTTP responses)
  - Processing files (copying, conversion, validation, display...)
  - Writing i18n code that is robust and correct
- This minilecture is intended to give a understanding of what Unicode is about and the consequences this entails for you as a programmer
- It's nowhere near to covering everything about Unicode or character sets

VIRGINIA TECH

Before we talk about character sets, let's talk about bytes

- A byte is a unit of digital information.
- An octet is a byte consisting of 8 bits ("8-bit byte"), which allows us to represent 256 possible values, in unsigned interpretation the integers from 0..255 (decimal) or 0x00..0xff (hex).
  - Historically, there were systems using smaller or larger bytes
  - In C, `uint8_t` is guaranteed to be 8 bits, but `unsigned char` is not in general (it's `CHAR_WIDTH` bits).
  - POSIX says that `CHAR_WIDTH` is 8 bits.
- Upshot: there is wide consensus what is meant when talking about bytes/octet, and streams of bytes: `48 65 6c 6c 6f 20 43 53 33 32 31 34`
- Bytes generally do not have an a priori interpretation other than the unsigned value associated with the bit pattern
- We typically ignore bit order (which bit is most/least significant) - this is a lower-layer concern (serial protocol, memory controller)
- Multibyte integers are subject to endianness e.g. do we interpret `01 02` as $1 \cdot 256 + 2 = 258$ or $2 \cdot 256 + 1 = 513$.

VIRGINIA TECH

Characters and Character Sets

- Characters are abstract entities from some kind of alphabet
- Consider this set of things (Source: freepik.com/flaticon.com)
- {  ,  ,  ,  ,  ,  }
- We may call these characters and associate names with them:
- apple, tree, flower, pretzel, ball, house
- Note: we haven't used numbers yet

VIRGINIA TECH

Character Encoding Example

- To work with abstract characters, we must encode them somehow in a way computers can understand them
- Possible idea: assign consecutive numbers
-  $= 0$,  $= 1$,  $= 2$,  $= 3$,  $= 4$,  $= 5$
- uses the integers $[0 \ldots 5]$. On a computer, this would require 3 bits. All characters would take up 3 bits in this encoding. 6 and 7 would not be used.
- This is not the only possible encoding.

VIRGINIA TECH

Alternative Character Encoding

- Encode characters as either one or two groups of 2 bits.
-  = 00,  = 01,  = 10
-  = 11 00,  = 11 01,  = 11 10
- apple, tree, and flower would require 2 bits in this encoding, pretzel, ball, and house would require 4 bits
- 00 01 10 11 00 means apple, tree, flower, pretzel
- 00 11 11 would be ill-formed
- When would such a variable-length encoding be a win?

VIRGINIA TECH

Character Sets in the Real World

- There used to be many character sets that were of importance: ASCII, ISO-8859-1, ISO-8859-2, ...
- Typically, these character sets were not defined in a manner that separates the abstract entities ("characters") from their representation/encoding
- They are all of only historical interest right now, because Unicode was defined as character set to replace all existing ones
- This is not to say that you may not encounter legacy data somewhere...
- This is also not to say that you shouldn't understand ASCII
  - Type `man ascii`

VIRGINIA TECH

The Unicode Standard

- https://home.unicode.org/
- A universal character set that includes enough abstract character definitions to express all major languages in the world (and then some).
- Unicode 14.0 defines 144,697 characters called "code points," and can accommodate up to 1,114,112 code points/characters in the future.
  - Code points are written using a number called a Unicode scalar value, like so: U+0041 but they also have a name
- Good news: many of these characters correspond to a single grapheme (intuitively, a letter or symbol used in a world language)
  - Unicode Character "A" (U+0041) is Latin Capital Letter A
  - Unicode Character "Ä" (U+00C4) is Latin Capital Letter A with Diaeresis
  - Unicode Character (U+1F385) is Father Christmas 
- Bad news: that's not always true. Unicode Character (U+0308) Combining Diaeresis means: "Put an umlaut over the preceding character," so the sequence U+0041 U+0308 is one grapheme Ä that may be indistinguishable from the grapheme expression U+00C4.

VIRGINIA TECH

The Unicode Standard

- Now that we have defined what the Unicode character set is, the big question will be: How should we encode Unicode characters?
  - while in a program's memory (in variables, a programming language's "strings", etc.)
  - while in transit or stored on disk
- Fundamental trade-off: ease of processing vs efficiency of storage
- To represent 1,114,112 code points, we would need 21 bits ($2^{21} = 2,097,152$) in an encoding that uses the same number of bits for each character
- Will talk about three encodings designed for text in Western languages (note: others exist, e.g. GB18030 optimizes encoding for Chinese characters).

VIRGINIA TECH

The UTF-32 Encoding

- UTF-32 encoding sets aside 32 bits for each character and represents each character using its code point number (aka "Unicode scalar value.")
- Has the advantage that we can find the n-th character in a sequence using indexing a[i] - allows us to think of unicode "strings" as arrays of unicode characters
-  is 0x0001F385
- Disadvantages:
  - Wasteful: most files will use only a fraction of the possible characters (say U+0000 - U+FFFF), and many files will use even fewer (say U+0000 - U+007F). For common English texts, encounter overhead of 4× compared to ASCII.
  - Endianness must now be defined (e.g., infamous BOM mark)
- Linux's `wchar_t` is 32 bits and uses this encoding
- C11's string literals support it as well (U"...") with `char32_t`

VIRGINIA TECH

The UTF-16 encoding

- If UTF-32 is too wasteful, then let's use 16-bits (2 bytes) to represent each character
- Now some characters will take 2 bytes, others will take 4 bytes. If 4 bytes, the first 2 bytes are called a "surrogate"
-  is 0xD83C 0xDF85
- Advantages: more compact than UTF-32, and we can index as an array ... well, most of the time
- Disadvantages:
  - still wasteful for many unicode sequences
  - error prone - i-th code unit is not i-th unicode character
- Unfortunately, this is the model chosen in both Java and JavaScript :-(

VIRGINIA TECH

The UTF-8 encoding

- Variable-length encoding that uses 1, 2, 3, or 4 bytes to encode a Unicode character/code point.
-  is 0xF0 0x9F 0x8E 0x85
- Advantages:
  - Space efficient, optimizing for common case
  - 7-bit ASCII strings are valid UTF-8 - thus no storage overhead for many English texts (and program code, etc.)
  - Can synchronize with input stream in at most 3 characters
- Disadvantage:
  - Indexing is impossible
- UTF-8 is by far the most common encoding when Unicode content is transmitted and/or stored
- UTF-8 is the format of built-in strings in languages like Go and Rust - these languages do not abstract the encoding away

VIRGINIA TECH

Q. Given a bunch of bytes, can we tell if it represents an encoding of a character set?

- In general, no.
- Legacy encodings (e.g., ISO-8859-1 are single-byte encodings that encode their 256 possible values using 0..255 — any random set of bytes will be a "valid" encoding of a string of ISO-8859 characters
- We can tell if it's not valid UTF-8 since not all byte sequences are valid UTF-8, but there's no guarantee that something that looks like valid UTF-8 is in fact an encoding of Unicode characters - could be coincidence
- Consequence: we cannot reliably interpret a file or transmitted object unless we receive or assume out-of-band information about the character set encoding of this file or object
  - HTTP responses include: `Content-Type: text/html; charset=UTF-8`
  - Files that are stored in most file systems (Linux, Windows) rely on the user or opening program to be interpreted correctly. They generally do not keep track of information such as "this file's content is meant to be interpreted as UTF-8 encoded Unicode." Suffixes are just conventions.

Practical Questions

- What Unicode encoding does a language use or provide to represent Unicode strings?
  - How can common string operations (indexing, searching for characters, etc.) be implemented?
- Do the I/O facilities perform conversion from an encoding to an internal representation ("decoding") and conversion form an internal representation to an encoding?
  - If so, which facilities do not do that, but operate on bytes instead?

VIRGINIA TECH

Example: Java (1.1 or later)

- String representation is UTF-16
  - To extract actual Unicode scalar values, need to use CharSequence.codePoints
- I/O facilities that refer to 'characters' or 'character-stream' decode on input and encode on output
  - But, not always to/from UTF-8 by default. On Windows, windows-1252 is used.
- By default, InputStreamReader ignores decoding errors, provide your own Decoder to control this
- Can use `java.lang.Character` methods to determine UTF-16 encoding - scalar values that require 2 UTF-16 code units are called "surrogate pairs," see `isHighSurrogate()` etc.

VIRGINIA TECH

Example: Python 3

- Python 3 strings are Unicode
- Can be thought of as 'UTF-32' - fully indexable, searchable, etc.
- Language optimizes under the hood
- Any attempts to mix Unicode strings and variables representing encoded strings (i.e., bytes) will result in a TypeError
- I/O in text mode converts from Unicode representation/encoding to internal Unicode strings
- Unicode finally done right?
- Only Swift (Apple's language for iPhones) goes one step further and abstracts away grapheme clusters, e.g., U+0041 U+0308 (A with combining diaeresis) is turned into U+00C4 automatically. (in Python 3, requires unicodedata.normalize)

VIRGINIA TECH

Example: Rust + Go

- Language's string type represents UTF-8 encoded strings
  - e.g., can't treat as array of Unicode scalar values
- Decoding/Encoding is not abstracted away from programmer
- Need to use crates/packages if Unicode processing is desired

VIRGINIA TECH

Example: C11

- Limited facilities:
- `wchar_t` performs encoding/decoding in "locale" defined encoding - on Linux, if set to UTF-8, corresponding to single Unicode scalar values
- Limited support for processing Unicode in internal representation
- However, many C library functions that were designed for traditional, byte-based, 00-terminated strings apply as well if these strings represent UTF-8 encoded Unicode strings. (But, a string of just zeros is valid UTF-8 but cannot be expressed here.)

VIRGINIA TECH

References

VIRGINIA TECH