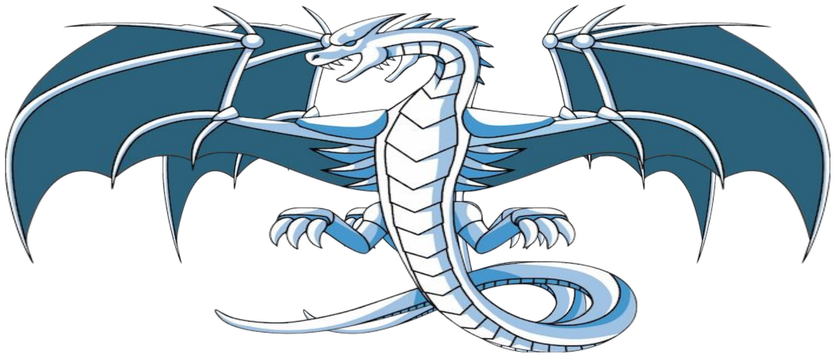
Widen Your Char-izons

Adding wide character conversion to LLVM-libc

By: Uzair Nawaz and Sriya Pratipati



01

Introduction

How Do We Represent Characters/Strings?

- Strings are just arrays of bytes (`char` is 1 byte)
- Super convenient for 1-byte characters, but not straightforward when representing more complex characters (emojis, other languages, etc)

L	L	V	M
0x4C	0x4C	0x56	0x4D

Multibyte vs Wide Characters

- Multibyte-Character Strings
  - Characters vary in size between 1 to 4 bytes
  - Length of a string in bytes != # of characters in the string
  - Referenced by a char * so possible to stop in the middle of a character
  - Typically represented by UTF-8 encoding
- Wide-Character Strings
  - Every character takes up the same number of bytes (usually 4 on most systems)
  - Easy to calculate length of string
  - Can’t stop in the middle of a wide character
  - Represented by UTF-32 encoding on most systems

“🤡Σ”

Multibyte representation (UTF-8) - 6 bytes

🤡				Σ	
0xF0	0x9F	0xA4	0xAF	0xCE	0xA3

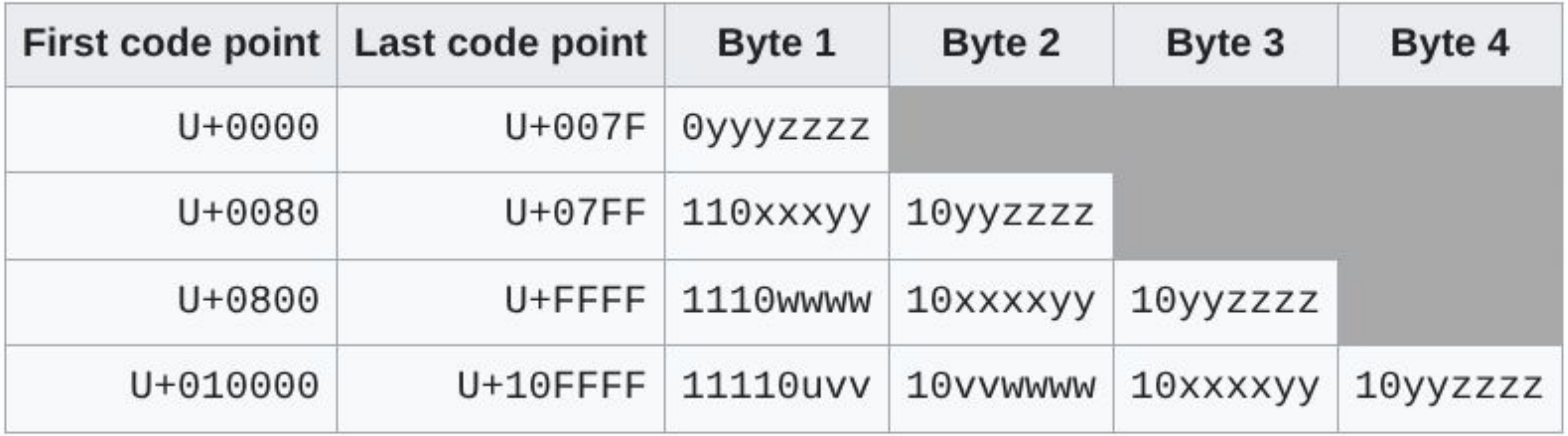
Wide character representation (UTF-32) - 8 bytes

🤡	Σ
0x0001F921	0x000003A3

02

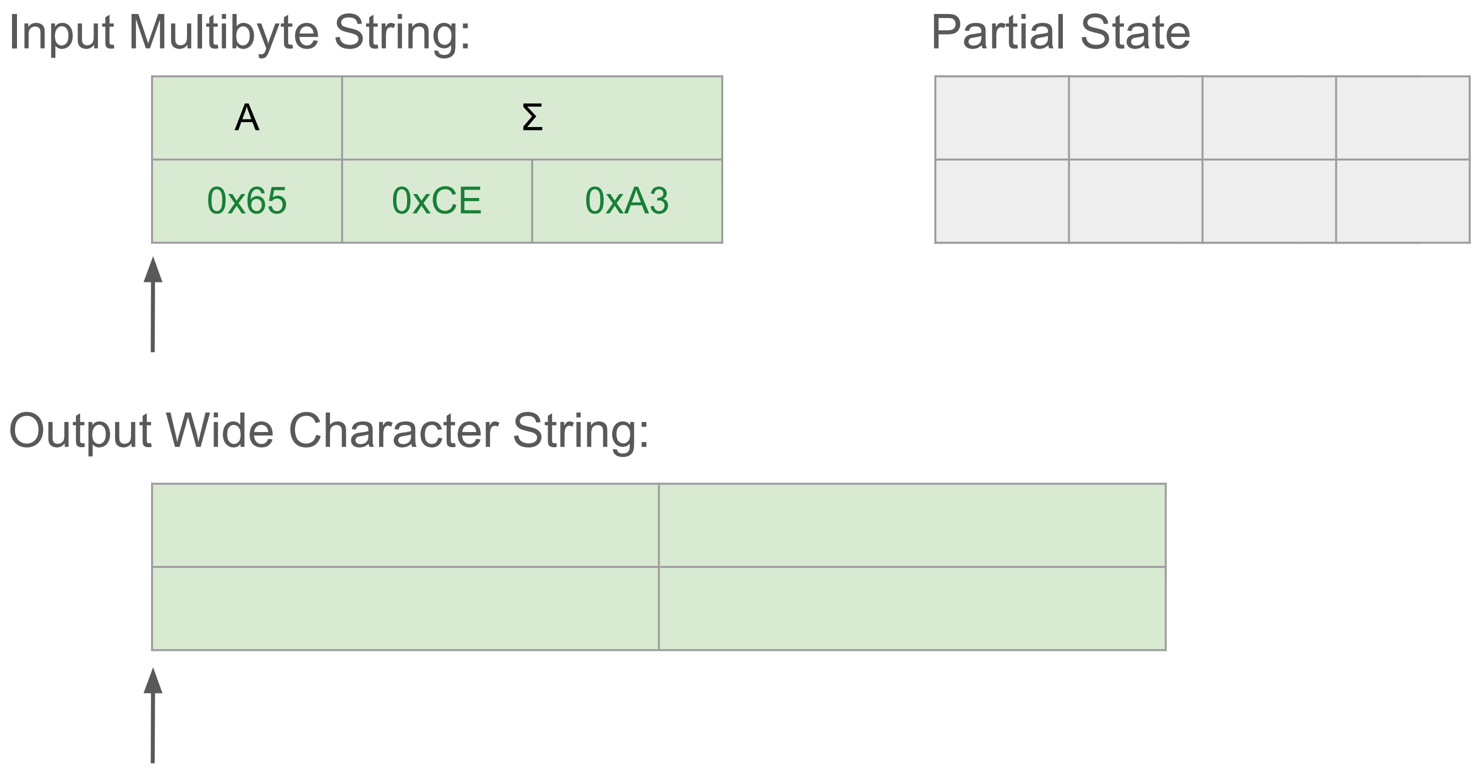
Conversion Process 🦖

UTF-8 Encoding Details

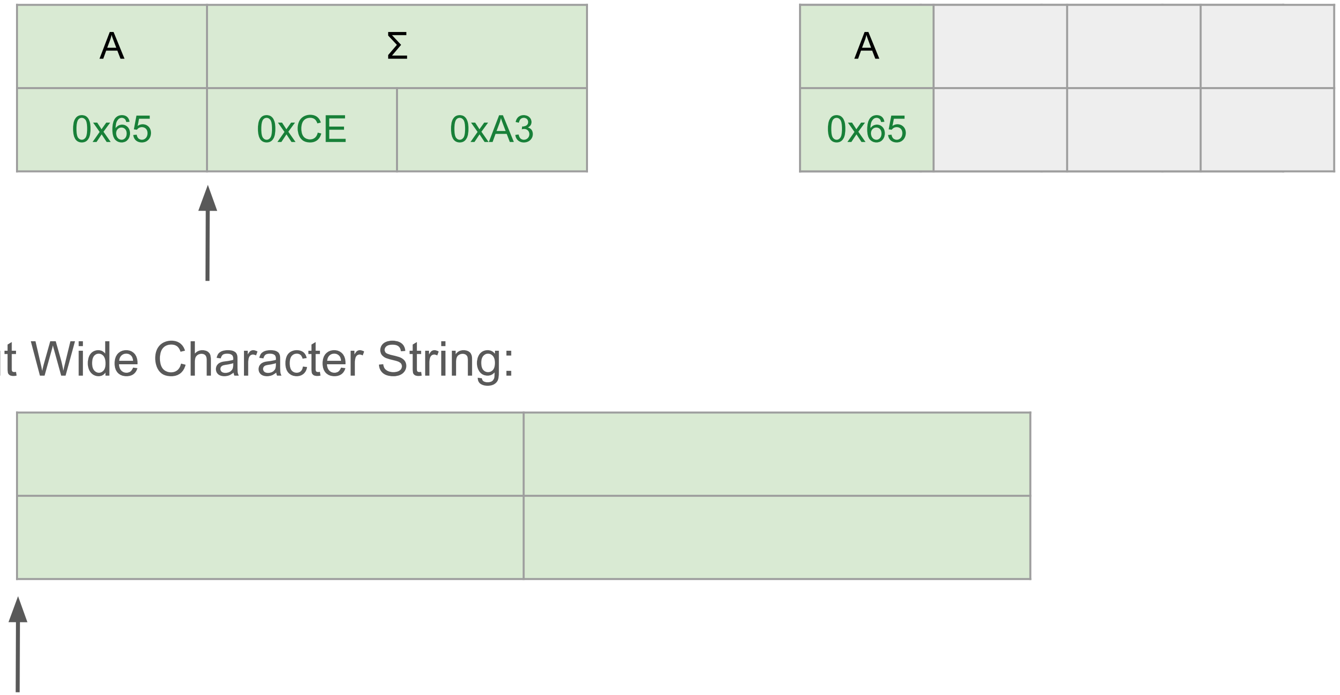

Code point ↔ UTF-8 conversion

First code point	Last code point	Byte 1	Byte 2	Byte 3	Byte 4
U+0000	U+007F	0yyyzzzz			
U+0080	U+07FF	110xxxyy	10yyzzzz		
U+0800	U+FFFF	1110wwww	10xxxxyy	10yyzzzz	
U+010000	U+10FFFF	11110uvv	10vvwwww	10xxxxyy	10yyzzzz

Multibyte -> Wide Character Example



Multibyte -> Wide Character Example



Input Multibyte String:

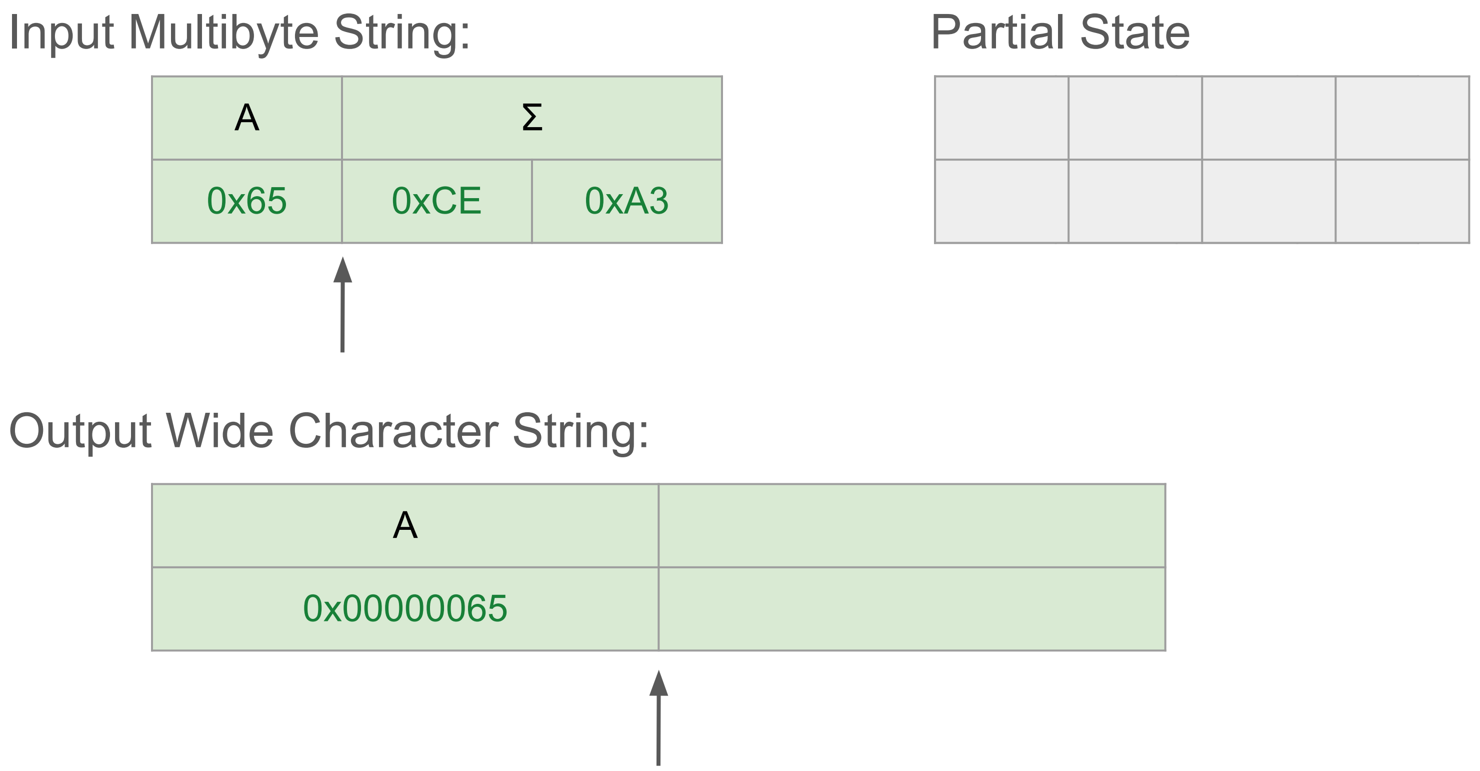
A	Σ	
0x65	0xCE	0xA3

Partial State

A			
0x65			

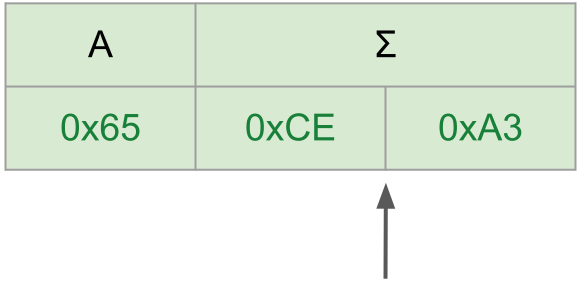
Output Wide Character String:

Multibyte -> Wide Character Example

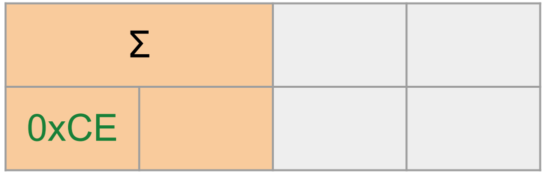


Multibyte -> Wide Character Example

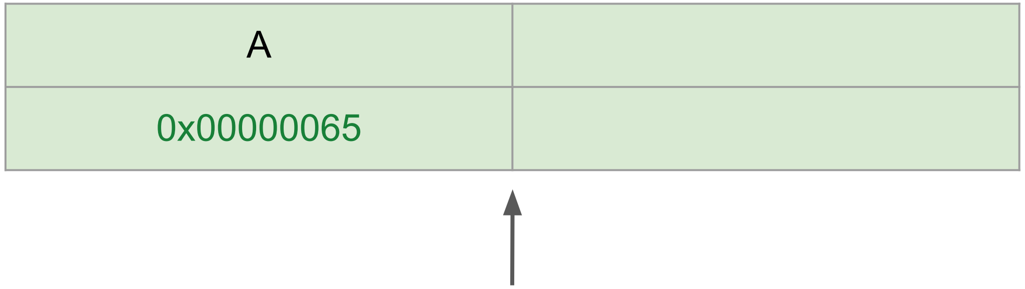
Input Multibyte String:



Partial State



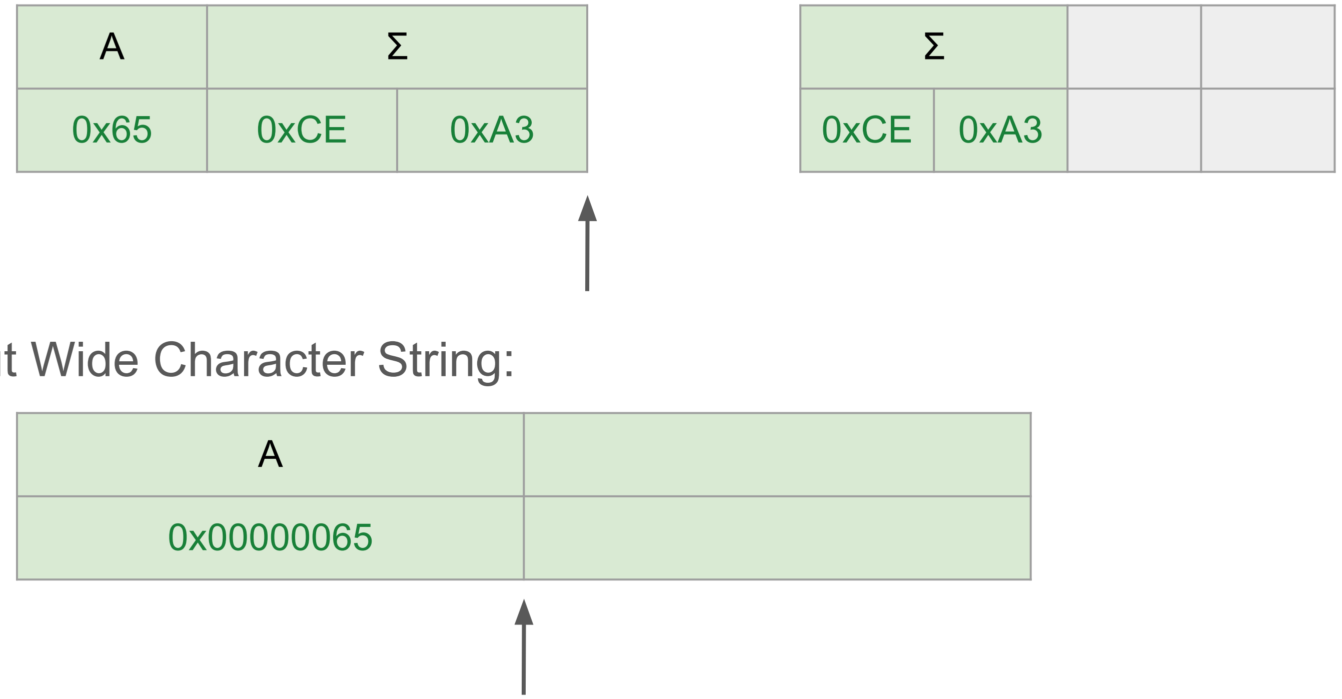
Output Wide Character String:



Multibyte -> Wide Character Example

Input Multibyte String:

Partial State



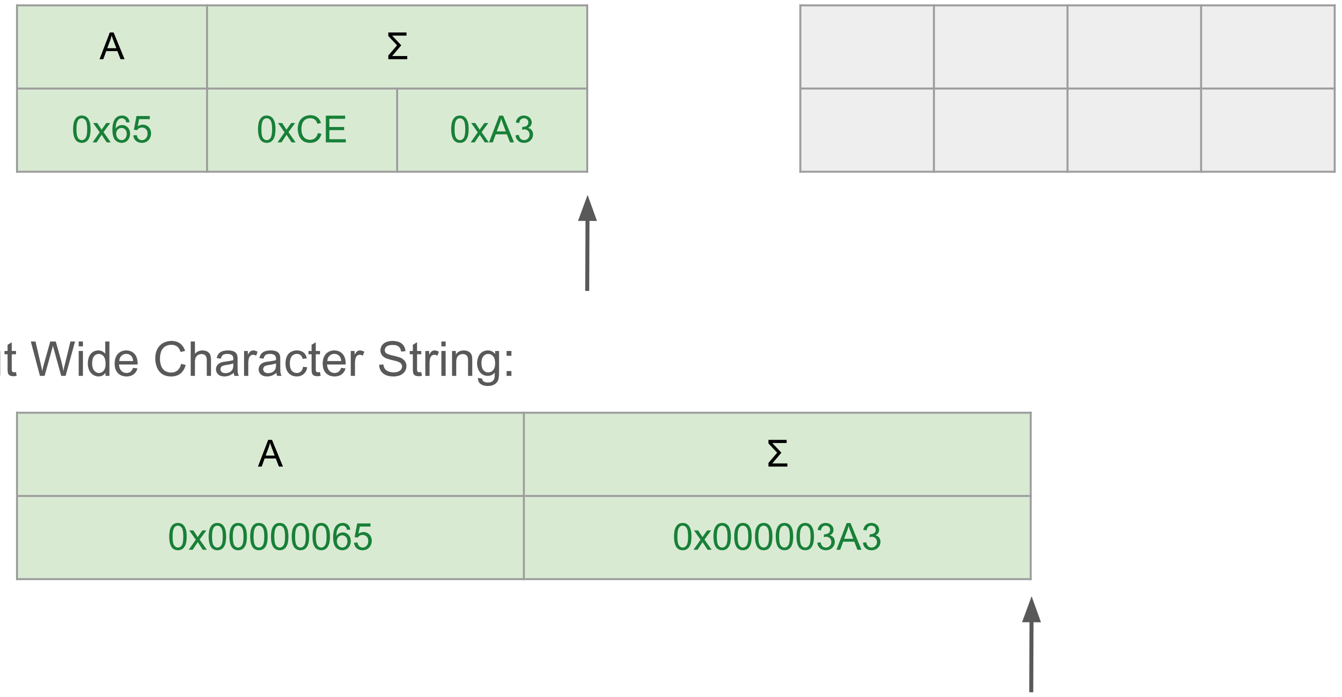
A	Σ	
0x65	0xCE	0xA3

Σ			
0xCE	0xA3		

Output Wide Character String:

A	
0x00000065	

Multibyte -> Wide Character Example



Input Multibyte String:

A	Σ	
0x65	0xCE	0xA3

Partial State

Output Wide Character String:

A	Σ
0x00000065	0x000003A3

03

Libc Interface 🪸

Example of mbrtowc use

```
const char* mb_str = "🤡";
wchar_t wc_string[1];
mbstate_t mbs;
size_t ret = mbrtowc(wc_string, &mb_str, /* max # of bytes to read */ 1,
&mbs);

ASSERT(ret == -2);
```

Libc Interface

```
const char* mb_str = "🤡";
wchar_t wc_string[1];
mbstate_t mbs;
size_t ret = mbrtowc(wc_string, &mb_str, /* max # of bytes to read */ 1,
&mbs);

ASSERT(ret == -2);

ret = mbrtowc(wc_string, &mb_str + 1, /* max # of bytes to read */ 3, &mbs);
ASSERT(ret == 3);

ASSERT(wc_string[0] == 0x0001F921);
```

Restartable vs Non-Restartable

- Restartable
  - Takes in an input of an mbstate, can stop conversion mid-character and pick up where it left off
- Non-Restartable
  - Has its own internal state that is maintained globally on each call to the function

04

Architecture 🏗️

mbstate_t

- Represents a partial conversion state
- Layout:

Field	**Size**
Partial State as UTF-32	32 bits
# bytes stored in partial state	8 bits
# of total bytes in mb-character	8 bits

CharacterConverter Class

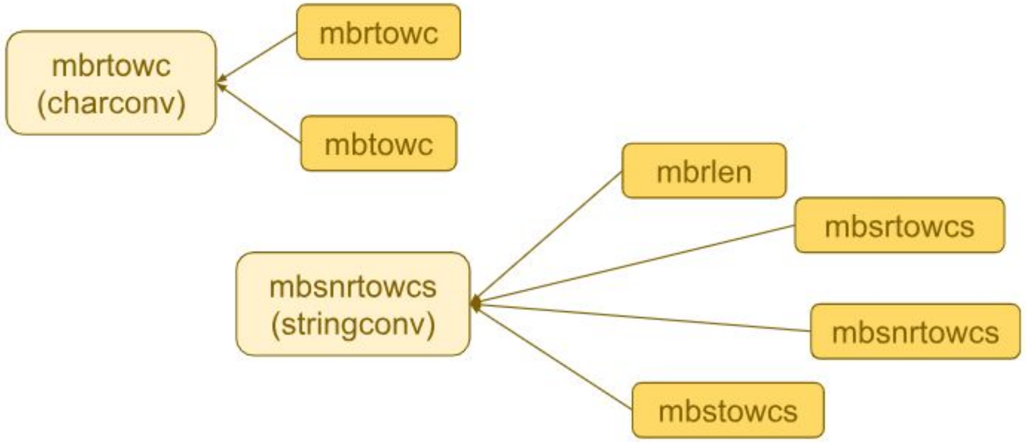
- Main internal interface to interact with mbstate
- Multibyte → Wide Character
  - `push(char8_t)`: Push a single byte from a multibyte sequence
  - `char32_t pop_utf32()`: Pop a wide character
- Wide Character → Multibyte
  - `push(char32_t)`: Push a wide character
  - `char8_t pop_utf8()`: Pop a single byte from a multibyte sequence
- Other utilities
  - `clear()`
  - `isEmpty()/isFull()`
  - `isValidState()`

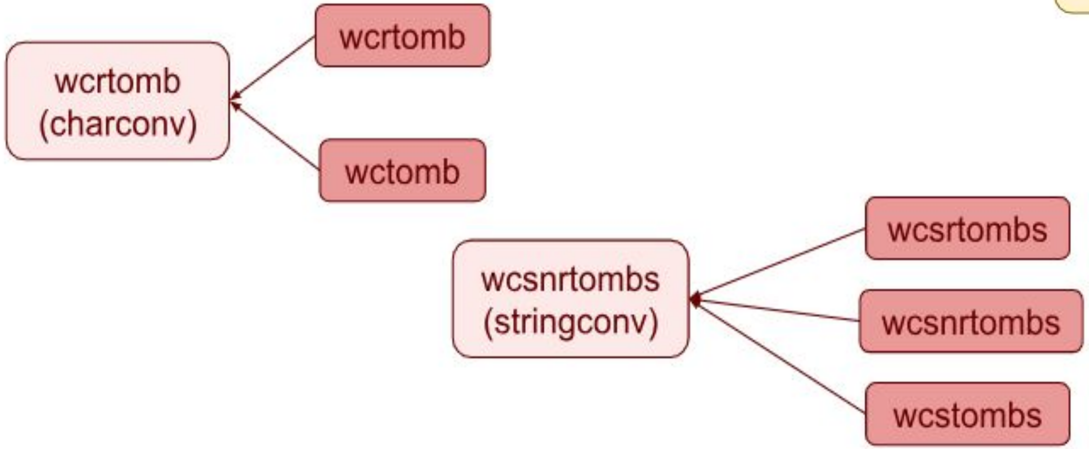
StringConverter Class

- Layer of abstraction above `CharacterConverter`
- Construct with an input string and then `pop` converted characters

Internal Restartable Functions

- Internal functions are equivalent to restartable public functions





05

Design Decisions 🍋

Size of mbstate/what to store

- Final decision: 6 bytes
  - 4 bytes to hold partial conversion
  - 8 bits each for number of total bytes and bytes stored
- Alternative 1: 4 bytes to hold partial conversion
  - Have to deduce total bytes and conversion status each time
- Alternative 2: 4 bytes
  - `state[20:0]` : partial conversion (utf-32)
  - `state[22:21]`: total bytes
  - `state[28:23]`: num bits processed
  - `state[31:29]`: unused

StringConverter Class

- The toughest design decision of the entire project
- Do we need a class to handle string conversion, or is the character converter sufficient?
- Class allows for scalability to UTF-16 conversions
- Simplifies code for internal functions

06

Reflections 🔮

Future Expansion

- Wide character support in FILE
- `wprintf`
- 16-bit wide characters using UTF-16 (for windows)
- `wctypes.h`: `iswalpha`, `iswupper`/`iswlower`, etc
- Widechar to floating point conversion (`wcstod`)
- Add Bazel rules for conversion functions

Thank you for listening!