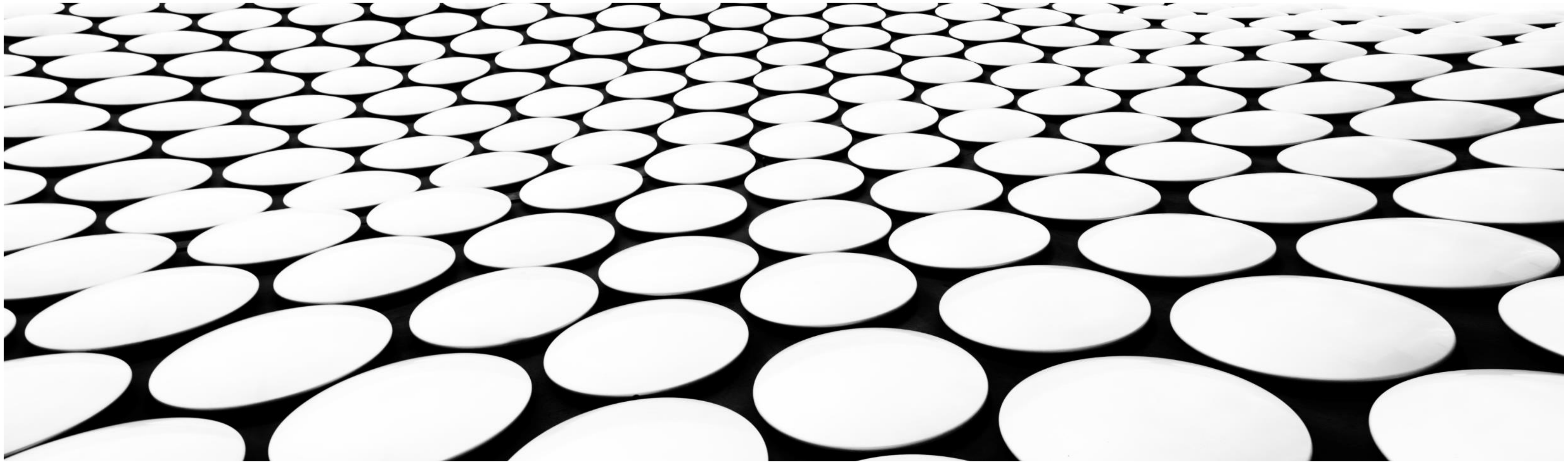
CATCHING ⍐: C++ UNICODE

PART 2 OF 5 | INTO THE DETAILS



JEANHEYD “THEPHD” MENEIDE MEETING C++ 2019 SATURDAY, NOVEMBER 16TH, 2019

RECAP

A LIVING BODY OF WORK

C++ UNICODE SUPPORT INCLUDES:

C++ UNICODE SUPPORT INCLUDES:

[This Space Intentionally Left Blank]

`char` IS NOT YOUR FRIEND

-
  ```
  std::string totally_utf8(u8“柴”);
  std::string also_totally_utf8(“柴”);
  assert(totally_utf8 == also_totally_utf8); // 😂
  ```

`char` IS NOT YOUR FRIEND

- 40,000 Lines into your codebase...
- `void process(std::string definitely_utf8);`

  ```
  process(u8“Jí pes bagetu?”); // okay
  process(“jedna koruna česká”); // uhh ...
  process(argv[1]); // ⚠ !!
  ```

`char8_t` INTRODUCED FOR C++20

- New fundamental type
  - Turns ambiguity into hard compiler errors

-
  ```
  std::u8string totally_utf8(u8“柴”);
  std::string also_totally_utf8(“柴”);
  assert(totally_utf8 == also_totally_utf8); // ✕
  ```

`unsigned char`: **savior before C++20**

- Allows you to get hard compiler errors, early
  - `using u8string = std::basic_string<unsigned char>;`
- 🏵 Blossom 🏵 changes out to program boundaries
  - Explicitly encode/decode or mark when going from/to `std::string`

C STANDARD: BUSTED

	mb	wc	mbs	wcs	c8	c16	c32	c8s	c16s	c32s
mb	—	✓			✗	R	R			
wc	✓	—			✗	✗	✗			
mbs			—	✓				✗	✗	✗
wcs			✓	—				✗	✗	✗
c8	✗	✗			—	✗	✗			
c16	R	✗			✗	—	✗			
c32	R	✗			✗	✗	—			
c8s			✗	✗				—	✗	✗
c16s			✗	✗				✗	—	✗
c32s			✗	✗				✗	✗	—

NEARLY UNANIMOUS CONSENSUS: FIX IT!

	mb	wc	mbs	wcs	c8	c16	c32	c8s	c16s	c32s
mb	—	✓			P R	R	R			
wc	✓	—			P R	P R	P R			
mbs			—	✓				P ✓	P ✓	P ✓
wcs			✓	—				P ✓	P ✓	P ✓
c8	P R	P R			—	✗	✗			
c16	R	P R			✗	—	✗			
c32	R	P R			✗	✗	—			
c8s			P ✓	P ✓				—	✗	✗
c16s			P ✓	P ✓				✗	—	✗
c32s			P ✓	P ✓				✗	✗	—

N2440 - https://thephd.github.io/vendor/future_cxx/papers/source/n2440.html

SIZED FUNCTIONS, TOO!

- n-style of C functions

  -
    ```
    size_t mbsnrtocXs(charX_t* restrict dest,
    const char** restrict src, size_t dest_len, size_t src_len,
    mbstate_t* restrict state);
    ```
- Allows:
  - usage of embedded nulls in data
  - SIMD optimizations, even on exotic architectures

[[SIDEBAR]] `rsize`

- “There were sized conversion functions in the C Standard Already!”
  - Referring to: Annex K
  - `RSIZE_MAX` – implementation defined, “should be (`SIZE_MAX / 2`)”!
  - “Implementations will do the right thing!”

🤣

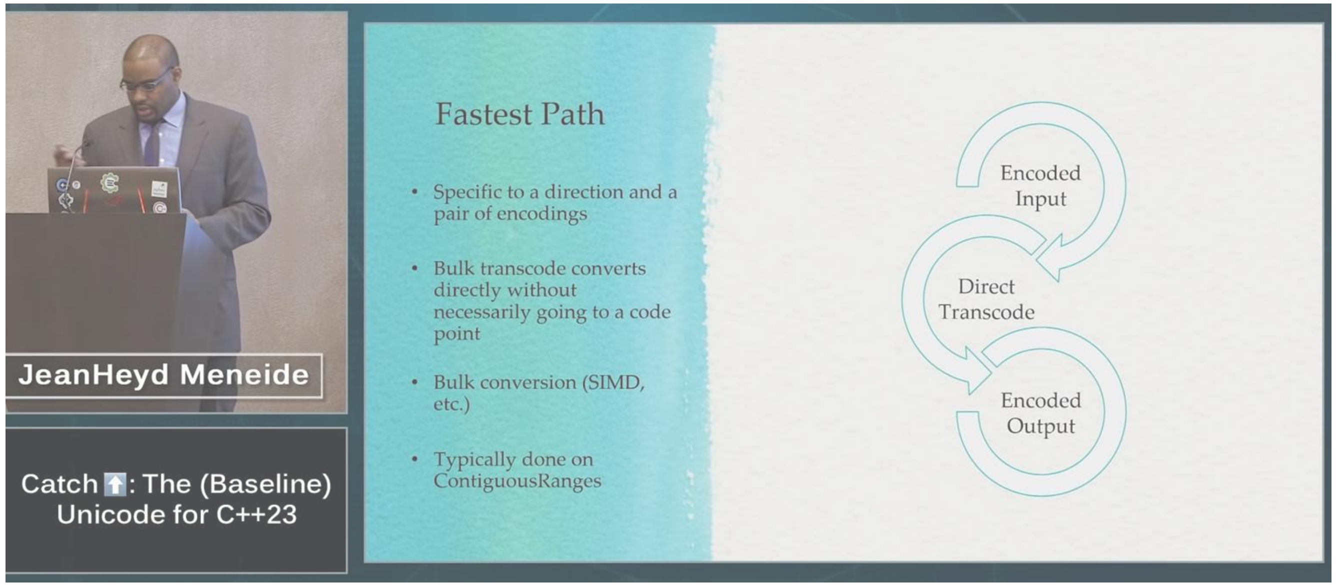
[[SIDEBAR]] THEOREM: THERE IS ALWAYS A DEATH STATION 9000

- Corollary: there exists a Hell++
- *Always*

[[END SIDEBAR]]

C STANDARD: MAKING PROGRESS

- WG14 C Standard General Rules:
  - There should be at least 2 implementations
  - It should build on existing practice
- Implementation in the *Small Device C Compiler* (*SDCC*)
  - Plus, freestanding in-progress library with *cuneicode*
  - musl and glibc next, soon?



JeanHeyd Meneide

Catch ⬆️: The (Baseline) Unicode for C++23

Fastest Path

- Specific to a direction and a pair of encodings
- Bulk transcode converts directly without necessarily going to a code point
- Bulk conversion (SIMD, etc.)
- Typically done on ContiguousRanges

CURRENT PROGRESS

STATUS REPORT ON THE ADVANCEMENTS

ENCODING CONCEPT

- Core Abstraction: represents an encode/decode pairing
  - Typedefs describe behavior

- Requires:
  - 3 Typedefs (`state`, `code_point`, `code_unit`)
  - 2 Static Member Variables (`max_code_points`, `max_code_units`)
  - 2 Functions (`encode_one`, `decode_one`)

HELPER TYPES

```
struct empty_struct {};

using byte_span = std::span<std::byte>;
using u8_span = std::span<char8_t>;
using u16_span = std::span<char16_t>;
using u32_span = std::span<char32_t>;

enum class encoding_errc : int {
   ok = 0x00,
   invalid_sequence = 0x01,
   insufficient_output_space = 0x02,
};
```

HELPER TYPES: RESULTS

```
struct decode_result {
   u8_span input;
   u32_span output;
   empty_struct& state;
   encoding_errc error_code;
   bool handled_error;
};
```

```
struct encode_result {
      u32_span input;
      u8_span output;
      empty_struct& state;
      encoding_errc error_code;
      bool handled_error;
};
```

HELPER TYPES: ERROR CALLBACKS

```
using decode_error_handler = std::function_ref<
   decode_result(utf8&, decode_result, u8_span)
>;

using encode_error_handler = std::function_ref<
   encode_result(utf8&, encode_result, u32_span)
>;
```

ENCODING OBJECTS

```
struct utf8 {
    using code_unit                   = char8_t;
    using code_point                  = char32_t;
    using state                       = empty_struct;
    static constexpr inline std::size_t max_code_points = 1;
    static constexpr inline std::size_t max_code_units = 4;

    encode_result encode_one(u8_span input, u32_span output,
        state& current, encode_error_handler error_handler);

    decode_result decode_one(u32_span input, u8_span output,
        state& current, decode_error_handler error_handler);
};
```

ENCODING OBJECTS

```
struct utf16 {
   using code_unit          = char16_t;
   using code_point         = char32_t;
   using state              = empty_struct;
   static constexpr inline std::size_t max_code_points = 1;
   static constexpr inline std::size_t max_code_units = 2;

   encode_result encode_one(u16_span input, u32_span output,
      state& current, encode_error_handler16 error_handler);

   decode_result decode_one(u32_span input, u16_span output,
      state& current, decode_error_handler16 error_handler);
};
```

ENCODING OBJECTS

```
struct gb18030 {
    using code_unit             = std::byte;
    using code_point            = gb_code_point; // !!
    using state                 = empty_struct;
    static constexpr inline std::size_t max_code_points = 1;
    static constexpr inline std::size_t max_code_units  = 4;

    encode_result encode_one(byte_span input, std::span<gb_code_point> output,
        state& current, encode_error_handlergb error_handler);

    decode_result decode_one(std::span<gb_code_point> input, byte_span output,
        state& current, decode_error_handler error_handler);
};
```

CHANGE IN CODE POINT?!

- GB18030 is a Unicode Transformation Format (UTF)
  - Mandated by the PRC
- But it stores information differently:
  - Uses Private Use Area (PUA) characters as well

STRONG CODE POINT TYPES

- Explored in Tom Honermann’s text_view
  - Helps emphasize each encoding might have its own “character set”
  - Industry players have written dissents against such a design
- phd::text allows it
  - Cost: layers above encoding expecting char32_t fail without conversion

PREVENTING CONSEQUENCES

- Failure to roundtrip data if it is not convertible to `char32_t`

```
struct gb_code_point {
    /* … */
    operator char32_t () const; // whew, okay…
};
```

DO WE NEED STRONG CODE POINTS?

- Strong Code Points?
  - Is `char32_t` enough?
  - Handling it in higher levels of code?
- Better Character Sets?
  - Strong encoding ⇔character collection association

STANDARD ENCODINGS

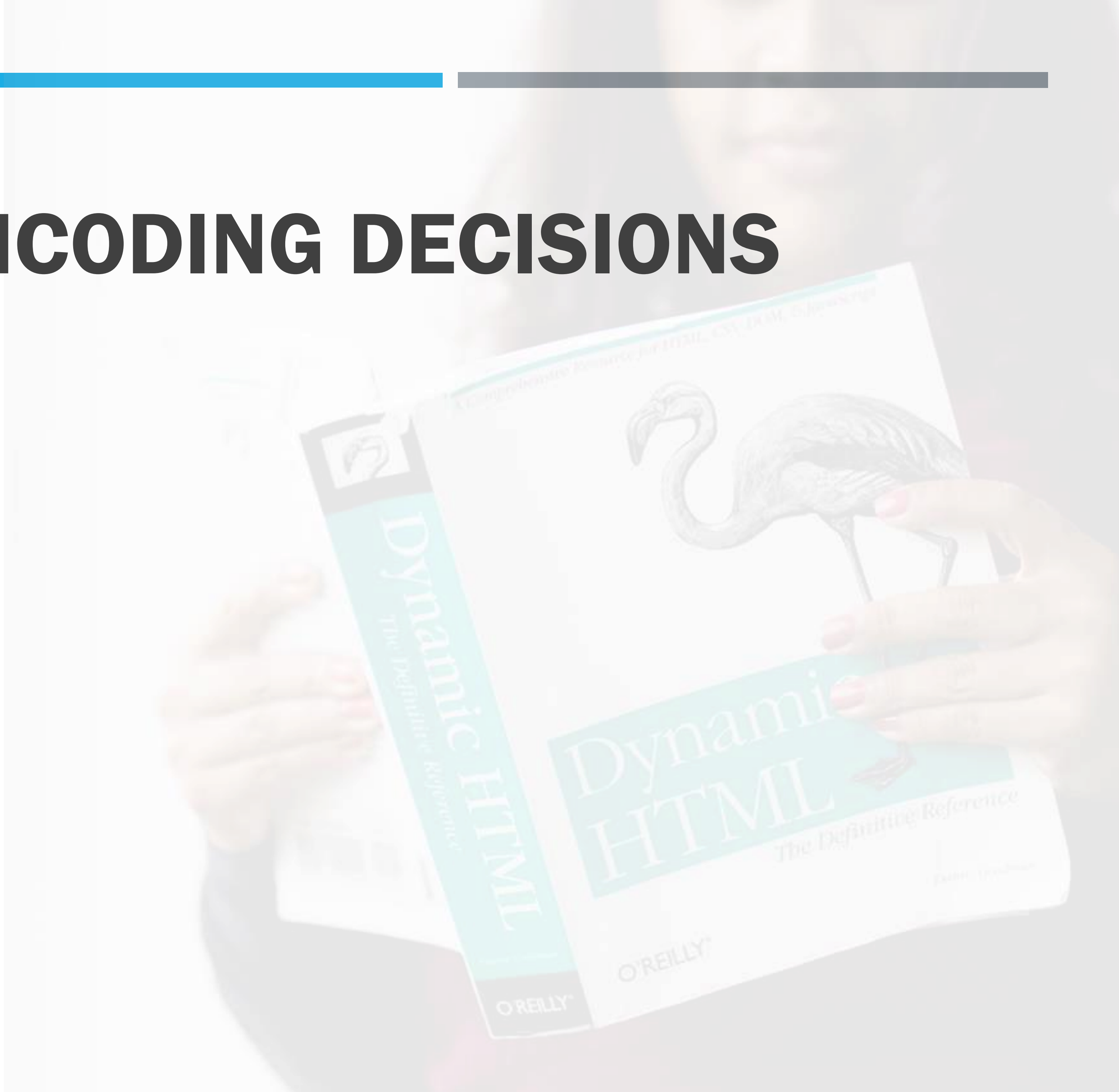
```
template <typename Char>
class basic_utf8;
template <typename Char>
class basic_utf16;
template <typename Char>
class basic_utf32;

class ascii;
class narrow_execution;
class wide_execution;
using utf8 = basic_utf8<char8_t>;
using utf16 = basic_utf16<char16_t>;
using utf32 = basic_utf32<char32_t>;
```

ANY_ENCODING

RUNTIME ENCODING STORAGE

RUNTIME-ONLY ENCODING DECISIONS

- Extremely Common
  - <meta charset=“UTF-8”>
  - Byte Order Marks
  - Robust document processing
  - Data exchange formats



```
struct any_encoding {
   using code_unit          = std::byte;
   using code_point         = char32_t;
   using state              = …;
   static constexpr inline std::size_t max_code_points = 4;
   static constexpr inline std::size_t max_code_units = 16;

   template <typename Encoding>
   any_encoding(Encoding&& some_encoding);

   encode_result encode_one(byte_span input, u32_span output,
      …, encode_error_handler error_handler);
   decode_result decode_one(u32_span input, byte_span output,
      …, decode_error_handler error_handler);
};
```

`any_encoding`

- Value-type wrapper around polymorphic storage
  - Typical abstract internal base class with typed implementation;
  - Internally, strengthen with Small Size optimization
  - Most encodings are stateless or empty, but…

`execution`, `wide_execution`

- Contains data:
  - `std::mbstate_t`
  - Shared multibyte state type for `char`/`wchar_t` encodings

```
struct wide_state {
    std::mbstate_t extra_space;
};
```

STATIC TYPES ☢ TYPE-ERASURE

- Implicit assumption in some parts of the library:

```
using some_state = encoding_state_t<SomeEncoding>;

some_state fresh_state{}; // create a fresh state to work on

old_state = some_state{}; // overwrite old state to “start new”
```

OH GEEZ.

```
struct any_encoding {
    using code_unit                     = std::byte;
    using code_point                    = char32_t;
    using state                         = …; // … ??? Uhhh… !
};
```

`any_state` ...?

- But its constructor needs more information than just "nothing"!

CONUNDRUM

- `any_state` needs information from type-erased `any_encoding`
  - But also keep default constructor without providing extra information?
  - Also copy too? Like `std::function<…>` …?

SOLUTIONS?

- 1. Fuse Encoding Object and State together?
  - Bad: default constructor for any_encoding → wipe out stored information
  - Bad: force `nullptr`-like semantics (like `std::function`) 😰

SOLUTIONS?

- 2. Use alternative paths specifically for any_encoding ?
  - Bad: no longer generic, patchy like std::reference_wrapper and stdlib
  - Bad: “secret backdoor” for “implementers only”

SOLUTIONS?

- 3. Store state on the encoding object?
  - Find a way to get data at runtime?
  - Detect this self-storage at compile-time?
  - Self-referential encoding objects?

SELF-REFERENTIAL...

SELF-REFERENTIAL!

```
struct any_encoding {
    using code_unit                  = std::byte;
    using code_point                 = char32_t;
    using state                      = any_encoding;

    /* … */
};
```

SELF-REFERENTIAL.

- Easy to check for!

  -
    ```
    if constexpr (std::is_same_v<Encoding, encoding_state_t<Encoding>>)
    {
      …
    }
    ```

- No special, secret provision: users can make their own

SELF-REFERENTIAL?

- How does someone ask to clear the state?
  - Likely requires a utility free function, `reset_state`
- Good: no special, secret provision!
  - Bad: complexity still being introduced.

ERROR HANDLERS

MAXIMALLY FLEXIBLE FOR DISPARATE WORKLOADS

REMEMBER THESE?

```
using decode_error_handler = std::function_ref<
  decode_result(utf8&, decode_result, u8_span)
>;

using encode_error_handler = std::function_ref<
  encode_result(utf8&, encode_result, u32_span)
>;
```

ANATOMY OF AN ERROR HANDLER

```
decode_result my_error_handler(
  utf8& the_encoding_being_used,
  decode_result current_state_of_reading,
  u8_span consumed_so_far
);
```

PRESERVING ALGORITHMIC INFORMATION

- Hands you the current encoding: useful for e.g. any_encoding
- Hands you the intended result type: manipulate input/output in response to errors
- No need to write caching iterators: last parameter contains all read values

RESULT BY VALUE

```
struct decode_result {
   u8_span input;
   u32_span output;
   empty_struct& state;
   encoding_errc error_code;
   bool handled_error;
};
```

```
struct encode_result {
      u32_span input;
      u8_span output;
      empty_struct& state;
      encoding_errc error_code;
      bool handled_error;
};
```

“FIND FIRST NORMAL SEQUENCE”

```
decode_result my_error_handler(utf8&, decode_result result, u8_span) {
   u8_span& in_ref = result.input;
   auto first_valid
      = std::ranges::find_if(in_ref, &u8valid_start_byte);
   in_ref = u8_span(first_valid, result.input.end());
   /* … insert replacement character … */
   return result;
}
```

MOST PEOPLE WILL USE:

```
class throw_handler;
class replacement_handler;
class ignore_incomplete_handler;
using default_error_handler = replacement_error_handler;
```

FINAL THOUGHTS

- Type erasure is hard with two separate pieces
  - Finding an elegant solution for any_encoding would improve the API
- Flexibility at lower levels is good
  - Higher level APIs take the sting out of the verbosity
  - Keep performance for different use cases

ACKNOWLEDGEMENTS

- Aaron Ballman, for giving me the gentle nudge to join WG14 and fix the C Standard
- sbi, Robot, melak47, and Lounge<C++>
  for showing me a great time here
- #include<C++>, for being an amazing community

SUPPORT THE EFFORT

MAKE UNICODE IN C++ A REALITY FOR YOU, YOUR CODEBASE AND YOUR COMPANY

The Pasture - Text Proposal & Goals
https://thephd.github.io/portfolio/text

Support
https://thephd.github.io/support/

LinkedIn - thephd
https://www.linkedin.com/in/thephd

Twitter - @thephantomderp
https://twitter.com/thephantomderp