Reinterpreting pointers of character types

Document #:	P2292R0
Date:	2021-01-28
Project:	Programming Language C++
Audience:	LEWG, EWG, SG-16

Reply-to: Corentin Jabot <corentin.jabot@gmail.com>

The best way to get the right answer is to write bad wording

Abstract

We propose a function `reinterpret_string_cast` to cast between `char*` and c`char8_t*` without UB

Tony tables

Before	After
`const char* str = "UB";` `auto res = reinterpret_cast<char8_t*>(str);`	`const char* str = "Well defined";` `auto res = std::reinterpret_string_cast<const char8_t*>(str);`

Motivation

`char8_t` introduced in C++20 by P0482R6 [1] is a character type designed to handle UTF-8 code units. An important motivation for `char8_t` is that it doesn't alias with `char` and `char8_t*` are not implicitly convertible to `char_t`. This allows users to make sure they never lose the utf8-ness of their strings and that they can handle both utf-8 and non-utf8 text and data within the same overload set.

There however exists a large number of libraries (iconv, ICU, gtk), and C functions that may use `char` to represent UTF-8 code units. As such, many people have expressed an interest in a way to interpret a range of `char8_t` to `char` or inversely for 2 way compatibilities with existing libraries, and system functions.

Design and constraints

- We don't want to change the design of `char8_t` (no aliasing or implicit conversion), as this would defeat the purpose of `char8_t`
- Because such a cast would only be safe if the preconditions (valid UTF code units sequence) are met, we think the syntax should be explicit.

Aliasing

As to not break the memory model and compilers, the proposed function assumes that after a call to `reinterpret_string_cast` the range passed as parameter will only be accessed through the return pointer, thereby sidesteping aliasing issues.

Open Questions

- Can it be constexpr?
- Does it needs to be?
- How much do we need to modify the object model and core wording?

Alternative design

It might be possible to add more rules to `reinterpret_cast`, to achieve similar results, however

- We want this conversion to be as explicit as possible
- `reinterpret_cast` is not `constexpr`,
- We think the magic function should work for span and string_view

Wording

The following wording is provided in an attempt to nerd-snipe a willing CWG expert into providing better wording.

```
// header <new> ?
namespace std {
    template <typename To, typename From>
    constexpr To reinterpret_string_cast(From from, std::size_t n) noexcept;
}

// header <span>
namespace std {
    template <typename To, typename From>
    constexpr std::span<To> reinterpret_string_cast(std::span<From> from);
```

```
}

// header string_view
namespace std {
    template <typename To, typename From>
    constexpr reinterpret_string_cast(std::basic_string_view<From> from) noexcept;
}


template <typename To, typename From>
constexpr To string_cast(From from, std::size_t n) noexcept;
```

Constraints:

- `From` is a pointer,
- `To` is a pointer,
- `std::remove_const_t<std::remove_pointer_t<From>>` is a character type,
- `std::remove_const_t<std::remove_pointer_t<To>>` is a character type,
- `sizeof(To) == sizeof(From)` is `true`,
- `From` and `To` have the same alignement.

Expects:

- `[from, from + n)` denotes a valid range
- `[from, from + n)` denotes a valid sequence of code units *seq* such that
  - if `same_as<From, char8_t> || same_as<To, char8_t>` is `true`, *seq* is a valid sequence of UTF-8 code units,
  - otherwise if `same_as<From, char16_t> || same_as<To, char16_t>` is `true`, *seq* is a valid sequence of UTF-16 code units,
  - otherwise if `same_as<From, char32_t> || same_as<To, char32_t>` is `true`, *seq* is a valid sequence of UTF-32 code units.
  - otherwise the behavior is undefined.

Effects: This function behaves as if:

1. The range `[from, from + n)` is copied to a temporary buffer `b` of `To` elements as if using `memcpy`
2. The destructor `~From()` is called on each element of the range `[from, from + n)`, ending their lifetime. Any pointer or reference to any address in the range `[from, from + n)` is invalidated.
3. A new array `a` of type `To` and size `n` is constructed at address `p` as if `new (p) To[n]`
4. Each element of `b` is copied in `a` as if per `memcpy`

Returns: A pointer to the first element in `a`.

Complexity: Constant time

[*Note:* After a call to this function accessing any object in the range `[from, from + n)` by any pointer or reference other than the one returned by the function is undefined. — *end note*]

Acknowledgments

References

[1] Tom Honermann. P0482R6: char8_t: A type for utf-8 characters and strings (revision 6). https://wg21.link/p0482r6, 11 2018.

[N4861] Richard Smith *Working Draft, Standard for Programming Language C++* https://wg21.link/N4861