Document number: P2291R0
Project: Programming Language C++
Audience: LEWGI, LEWG, LWG

Daniil Goncharov neargye@gmail.com
Karaev Alexander akaraevz@mail.ru

Date: 2021-01-26

Add Constexpr Modifiers to Functions to_chars and from_chars for Integral Types in <charconv> Header

I. Introduction and Motivation

There is currently no standard way to make conversion between numbers and strings *at compile time*.

`std::to_chars` and `std::from_chars` are fundamental blocks for parsing and formatting being locale-independent and non-throwing without memory allocation, so they look like natural candidates for constexpr string conversions. The paper proposes to make `std::to_chars` and `std::from_chars` functions for **integral types** usable in constexpr context.

Consider the simple example:

```
constexpr std::optional<int> to_int(std::string_view s) {
    int value;

    if (auto [p, err] = std::from_chars(s.begin(), s.end(), value); err ==
std::errc{}) {
        return value;
    } else {
        return std::nullopt;
    }
}

static_assert(to_int("42") == 42);
static_assert(to_int("foo") == std::nullopt);
```

⚠We do **not** propose `constexpr` for floating-point overloads, see design choices below.

`constexpr std::format` and reflection

In C++20 `constexpr std::string` was adopted, so we can already build strings at compile-time:

```
static_assert(std::string("Hello, ") + "world" + "!" == "Hello, world");
```

In addition, `std::format` was also adopted in C++20 and now its original author actively proposes various improvements like P2216 for compile-time format string checking. The current proposal is another step towards fully `constexpr std::format` which implies not only format string checking but also compile-time formatting (the only non-`constexpr` dependency of `std::format` is `<charconv>`):

```
static_assert(std::format("Hello, C++{}!", 23) == "Hello, C++23!");
```

This can be very useful in context of reflection, i.e. to generate unique member names:

```
// consteval function
for (std::size_t i = 0; i < sizeof...(Ts); i++) {
    std::string member_name = std::format("member_{}", i);
}
```

No standard way to parse integer from string at compile-time

There are too many ways to convert string-like object to number - `atol`, `sscanf`, `stoi`, `strto*l`, `istream` and the best C++17 alternative - `from_chars`. However, none of them are `constexpr`. This leads to numerous hand-made `constexpr int detail::parse_int(const char* str)` or `template <char...> constexpr int operator"" _foo()` in various libraries:

- `boost::multiprecision` and similar examples with `constexpr` user-defined literals for *my-big-integer-type* construction at compile-time.
- `boost::metaparse` — *yet another* `template <> struct digit_to_int_c<'0'> : boost::mpl::int_<0> {};`
- `lexy` — parser combinator library with manually written `constexpr std::from_chars` equivalent for integers (any radix, overflow checks).
- `ctre` (compile time regular expressions) — number parsing is an important part of regex pattern processing (`ctre::pcre_actions::hexdec`).

II. Design Decisions

The discussion is based on the implementation of `to_chars` and `from_chars` from Microsoft/STL, because it has full support of `<charconv>`.

During testing, the following changes were made to the original algorithm to make the implementation possible:

- Add constexpr modifiers to all functions
- Replace internal assert-like macro with simple assert (`_Adl_verify_range`, `_STL_ASSERT`, `_STL_INTERNAL_CHECK`)
- Replace `static constexpr` variables inside function scope with `constexpr`
- Replace `std::memcpy`, `std::memmove`, `std::memset` with constexpr equivalents: `third_party::trivial_copy`,`third_party::trivial_move`, `third_party::trivial_fill`. To keep performance in a real implementation, one should use `std::is_constant_evaluated`

Testing

All the corresponding tests were *constexprified* and checked at compile-time and run-time. The modified version passes full set tests from Microsoft/STL test.

Floating-point

`std::from_chars`/`std::to_chars` are probably the most difficult to implement parts of a standard library. As of January 2021, only one of the three major implementations has full support of P0067R5:

`Vendor`	**`<charconv>` support (according to cppreference.com)**
`libstdc++`	✗ no floating-point `std::to_chars`
`libc++`	✗ no floating-point `std::from_chars`/`std::to_chars`
`MS STL`	✓ full support

So at least for now we don't propose `constexpr` for floating-point overloads.

III. Conclusions

`to_chars` and `from_chars` are basic building blocks for string conversions, so marking them `constexpr` provides a standard way for compile-time parsing and formatting.

IV. Proposed Changes relative to N4861

All the additions to the Standard are marked with green.

A. Modifications to "20.19.1 Header <charconv> synopsis" [charconv.syn]

constexpr to_chars_result to_chars(char* first, char* last, see below value, int base = 10);

to_chars_result to_chars(char* first, char* last, bool value, int base = 10) = delete;

to_chars_result to_chars(char* first, char* last, float value);

to_chars_result to_chars(char* first, char* last, double value);

to_chars_result to_chars(char* first, char* last, long double value);

to_chars_result to_chars(char* first, char* last, float value, chars_format fmt);

to_chars_result to_chars(char* first, char* last, double value, chars_format fmt);

to_chars_result to_chars(char* first, char* last, long double value, chars_format fmt);

to_chars_result to_chars(char* first, char* last, float value, chars_format fmt, int precision);

to_chars_result to_chars(char* first, char* last, double value, chars_format fmt, int precision);

to_chars_result to_chars(char* first, char* last, long double value, chars_format fmt, int precision);

constexpr from_chars_result from_chars(const char* first, const char* last, see below & value, int base = 10);

from_chars_result from_chars(const char* first, const char* last, float& value, chars_format fmt = chars_format::general);

from_chars_result from_chars(const char* first, const char* last, double& value, chars_format fmt = chars_format::general);

from_chars_result from_chars(const char* first, const char* last, long double& value, chars_format fmt = chars_format::general);

D. Modify to "17.3.2 Header <version> synopsis" [version.syn]

#define __cpp_lib_to_chars *DATE OF ADOPTION*

V. Revision History

Revision 0:

- Initial proposal

VI. Acknowledgements

Thanks to Antony Polukhin for reviewing the paper and providing valuable feedback.

VII. References

- [N4861] Working Draft, Standard for Programming Language C++. Available online at https://github.com/cplusplus/draft/releases/download/n4861/n4861.pdf
- Microsoft's C++ Standard Library https://github.com/microsoft/STL, commit 2b4cf99c044176637497518294281046439a1bcc
- Proof of concept for `to_chars` and `from_chars` functions for integral types https://github.com/Neargye/charconv-constexpr-proposal/tree/integral
- [P0067R5] Elementary string conversions http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2016/p0067r5.html
- [P2216R2] `std::format` improvements http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2021/p2216r2.html