Document number: P2010R0
Project: Programming Language C++
Audience: SG6 Numerics

Daniil Goncharov neargye@gmail.com

Date: 2019-12-25

Remove iostream operators from P1889

I. Introduction and Motivation

[P1889R0] includes `iostream` specializations and `std::to_string` overloads for numerics types. As was noted by LEWGI in Belfast iostreams may not be the best IO interface in C++20. `std::format` provides a more general formatting and parse functionality, `std::to_chars` and `std::from_chars` provide a more basic formatting and parsing functionality that could be used in freestanding environments.

We propose revising the output and input APIs from [P1889R0].

II. Design Decisions

A. `iostream` specializations

In [P1889R0] `operator<<` is usually defined as `os << to_string(numeric)`. It is simple to implement, but such API is difficult to expand. For example it is impossible to localize the output or provide different representations via io manipulators.

[N4842] has a detailed description of working with locale and formatting for `operator<<(int val)` and `operator>>(int val)`, [P1889R0] does not do that.

B. `to_string`

`to_string` an easy-to-use function but definitions from [P1889R0] have problems:

1. [N4842] defines `string to_string(int val)` without additional parameters for formatting, but some parts of [P1889R0] `to_string` as `string to_string(const integer& val, int radix = 10)`. This is inconsistent.
2. Freestanding implementations may not have `std::string`, which makes `to_string` unimplementable on such platforms.
3. `to_string` functions do not have common counterparts that provide a conversion from strings to numbers

C. `std::format`

Implementation of `std::format` for integer types usually requires `to_chars` and `from_chars`. Therefore `to_chars` and `from_chars` should be added to [P1889R0] first.

[P1889R0] primarily includes integer types, therefore the description of `to_chars` and `from_chars` functions can be based on descriptions of already implemented overloads.

III. Conclusions

`to_chars` and `from_chars` are the basic building block for formatting. For integral types they are simple implement and provide a slodid base for future extensions.

Other ways of formatting from [P1889R0] are inconsistent or broken. We propose to remove them for now.

IV. Proposed Changes

We propose the following changes to [P1889R0].

All additions are marked with green, all removes are marked with red.

A. Remove `iostream` specializations from [P1889R0].

Modifications to "Header `<wide_integer>` synopsis" [numeric.wide_integer.syn]

```
 template<class Char, class Traits, size_t Bits, typename S>
basic_ostream<Char, Traits>& operator<<(basic_ostream<Char, Traits>& os,
                                        const wide_integer<Bits, S>& val);

template<class Char, class Traits, size_t Bits, typename S>
basic_istream<Char, Traits>& operator>>(basic_istream<Char, Traits>& is,
                                        wide_integer<Bits, S>& val);
```

Modifications to "Bits" [numeric.bits.syn]

```
 template<class Char, class Traits, size_t Bits, typename S>
basic_ostream<Char, Traits>& operator<<(basic_ostream<Char, Traits>& os,
                                        const bits& val);

template<class Char, class Traits, size_t Bits, typename S>
basic_istream<Char, Traits>& operator>>(basic_istream<Char, Traits>& is,
                                        bits& val);
```

Modifications to "Integer" [numeric.integer.syn]

```
template<class Char, class Traits, size_t Bits, typename S>
basic_ostream<Char, Traits>& operator<<(basic_ostream<Char, Traits>& os,
                                        const integer& val);

template<class Char, class Traits, size_t Bits, typename S>
basic_istream<Char, Traits>& operator>>(basic_istream<Char, Traits>& is,
                                        integer& val);
```

B. Remove `std::to_string` overloads from [P1889R0].

Modifications to "Header `<wide_integer>` synopsis" [numeric.wide_integer.syn]

```
template<size_t Bits, typename S>
std::string to_string(const wide_integer<Bits, S>& val);

template<size_t Bits, typename S>
std::wstring to_wstring(const wide_integer<Bits, S>& val);
```

Modifications to "Bits" [numeric.bits.syn]

```
template <class CharT = char,
          class Traits = char_traits<CharT>,
          class Alloc = allocator<CharT>>
basic_string<CharT, Traits, Alloc> to_string(CharT zero = CharT('0'),
                                             CharT one = CharT('1')) const;
```

Modifications to "Integer" [numeric.integer.syn]

```
string to_string(const integer& val, int radix = 10);
```

C. Add `to_chars` and `from_chars` overloads to [P1889R0].

Modifications to "Header `<wide_integer>` synopsis" [numeric.wide_integer.syn]

```
to_chars_result to_chars(const char* first, const char* last,
                         const wide_integer<Bits, S>& value, int base = 10);

from_chars_result from_chars(const char* first, const char* last,
                             wide_integer<Bits, S>& value, int base = 10);
```

Modifications to "Integer" [numeric.integer.syn]

```
 to_chars_result to_chars(const char* first, const char* last,
                          const integer& value, int base = 10);

from_chars_result from_chars(const char* first, const char* last,
                             integer& value, int base = 10);
```

VI. Revision History

Revision 0:

- Initial proposal

VII. Acknowledgements

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

VIII. References:

- [P1889R0] C++ Numerics Work In Progresshttp://www.open-std.org/jtc1/sc22/wg21/docs/papers/2019/p1889r0.pdf.
- [N4842] Working Draft, Standard for Programming Language C++. Available online at https://github.com/cplusplus/draft/releases/download/n4842/n4842.pdf