Document Number:	P2551R1
Date:	2022-05-09
Reply-to:	Matthias Kretz <m.kretz@gsi.de> Jonathan Wakely <cxx@kayari.org>
Audience:	LEWG
Target:	C++23

Clarify intent of P1841 numeric traits

ABSTRACT

A list of design-related questions after implementation of [P1841R2] "Wording for Individually Specializable Numeric Traits".

CONTENTS

0 CHANGELOG

0.1 CHANGES FROM REVISION 0

Previous revision: P2551R0

- Removed questions that were answered in the last telecon.
- Present options for `reciprocal_overflow_threshold`.

1 INTRODUCTION

[P1841R2] provides wording for numeric traits. The last design paper was [P0437R1] with additions from [P1370R1]. Most of the open questions were answered in LEWG already. The question on `reciprocal_overflow_threshold` was deferred to let the authors of [P1370R1] and this paper determine the original intent and its consequences.

2 REMAINING DESIGN QUESTIONS

1. Decision already taken in LEWG.
2. Decision already taken in LEWG.
3. (no action requested) In the prevision discussion (and poll) of this point we noticed that the traits P2551R0 listed was incomplete/incorrect. The poll taken in LEWG therefore said: "The numeric traits that are not meaningful for `numeric_limits` (`denorm_min`, `epsilon`, etc) should be disabled for integral types." The resulting list then is:
   - `denorm_min`
   - `epsilon`
   - `max_exponent`
   - `max_exponent10`
   - `min_exponent`
   - `min_exponent10`
   - `infinity`
   - `quiet_NaN`
   - `signaling_NaN`

The (bad) list in P2551R0 was:

- `denorm_min`
- `epsilon`
- `norm_min`
- `reciprocal_overflow_threshold`
- `round_error`
- `max_exponent`
- `max_exponent10`
- `min_exponent`
- `min_exponent10`

4. `reciprocal_overflow_threshold` is currently defined as:

P1841R2 [num.traits.val]

```
template <class T> struct reciprocal_overflow_threshold<T> { see below };
```

9 The smallest positive value x of type `T` such that `T(1)`$/x$ does not overflow.

This yields a subnormal number for IEC559 types. How should this value change wrt. treat-denormals-as-zero? I.e. in a situation where the hardware treats subnormal operands as zero you get 1/0 -> inf, which does overflow. In which case it doesn't match the specification anymore. This trait is specified by a behavior and as such may depend on processor state. As a compile-time constant this value must be independent from runtime behavior. But what is the correct value? See `https://godbolt.org/z/eWxdnTYf8` for a demonstration of the problem.

Update after consultation with Mark and Damien (the P1370R1 authors):

- It would be possible to decouple the specification from runtime behavior by specifying behavior of constant expressions only; i.e. that `T(1)`$/x$ does not overflow *in a constant expression*.
- P1370R1 presented an algorithm to determine the value and it does not yield the "*smallest* positive value x of type `T` such that `T(1)`$/x$ does not overflow".
- The P1370R1 algorithm seems to ensure that the value is never subnormal. Thus, the specification should have been "The smallest positive *normal* value x of type `T` such that `T(1)`$/x$ does not overflow"

- Since the actual reciprocal overflow threshold depends on runtime state, we're not sure who would/should use a compile-time constant. It seems simpler and safer to remove `reciprocal_overflow_threshold` from P1841. Mark wrote:

  > I would prefer to remove `reciprocal_overflow_threshold` entirely. The intent of the feature was to describe actual computer behavior at run time, so that library authors could write generic code. However, we can't do that with traits. For example, traits can't change value based on compiler flags. I wish I had realized that better when proposing the feature.

5. Decision already taken in LEWG.

3 SUGGESTED STRAW POLLS

Poll: Remove `reciprocal_overflow_threshold` from P1841.

SF	F	N	A	SA

If the above poll doesn't reach consensus:

Poll: Specify the behavior of `1 / reciprocal_overflow_threshold` only for constant expressions.

SF	F	N	A	SA

Poll: Require `reciprocal_overflow_threshold` to be a normal number.

SF	F	N	A	SA

4 STRAW POLLS

4.1 LEWG telecon 2022-03-29

Poll: Numeric traits can deviate from `numeric_limits`.

SF	F	N	A	SA
13	8	0	0	0

Poll: Numeric traits should be based on representation rather than behavior (ignoring `reciprocal_overflow_threshold`).

SF	F	N	A	SA
7	5	2	0	0

Poll: All numeric traits for bool should be disabled.

SF	F	N	A	SA
12	6	1	0	0

Poll: The numeric traits that are not meaningful for `numeric_limits` (`denorm_min`, `epsilon`, etc) should be disabled for integral types.

SF	F	N	A	SA
14	3	0	0	0

Poll: `max_digits10` should deviate from `numeric_limits` and yields `digits10_v + 1`.

SF	F	N	A	SA
6	5	2	0	0

A BIBLIOGRAPHY

[P0437R1] Walter E. Brown. *P0437R1: Numeric Traits for the Standard Library*. ISO/IEC C++ Standards Committee Paper. 2018. URL: `https://wg21.link/p0437r1`.

[P1841R2] Walter E. Brown. *P1841R2: Wording for Individually Specializable Numeric Traits*. ISO/IEC C++ Standards Committee Paper. 2021. URL: `https://wg21.link/p1841r2`.

[P1370R1] Mark Hoemmen and Damien Lebrun-Grandie. *P1370R1: Generic numerical algorithm development with(out) numeric_limits*. ISO/IEC C++ Standards Committee Paper. 2019. URL: `https://wg21.link/p1370r1`.