Doc No: SC22/WG21/N3210
PL22.16/10-0200
Date: 2010-11-12
Project: JTC1.22.32
Reply to: Stefanus Du Toit
Intel Corporation
stefanus.du.toit@intel.com

Background

This paper presents new wording for section 20.6.2 [ratio.arithmetic]. It is a response to NB Comment GB 89 (see N3102) and based on the same basic principles as N3131, but replaces the wording in question completely to make it clearer and better achieve the objectives of the response.

The intention of this wording is to allow two types of implementations of ratio arithmetic:

- a simple implementation where the program will be ill-formed if the intermediate computations overflow, and,
- a more complex implementation that guarantees the correct result if it is representable, even if the intermediate computations may not be representable in their simplest forms.

We furthermore improve the wording of the section overall, for example, by avoiding referring to `ratio`s as types when they are in fact template aliases.

Proposed Wording

Replace 20.6.2 [ratio.arithmetic] "Arithmetic on `ratio` types" completely with the following section:

20.6.2 Arithmetic on `ratio`s [ratio.arithmetic]

1. Each of the template aliases `ratio_add`, `ratio_subtract`, `ratio_multiply`, and `ratio_divide` denotes the result of an arithmetic computation on two `ratio`s `R1` and `R2`. With `X` and `Y` computed (in the absence of arithmetic overflow) as specified by [table below], each alias denotes a `ratio<U, V>` such that `U` is the same as `ratio<X, Y>::num` and `V` is the same as `ratio<X, Y>::den`.

2. If it is not possible to represent `U` or `V` with `intmax_t`, the program is ill-formed. Otherwise, an implementation should yield correct values of `U` and `V`. If it is not possible to represent `X` or `Y` with `intmax_t`, the program is ill-formed unless the implementation yields correct values of `U` and `V`.

Table NN - Expressions used to perform ratio arithmetic

Type	Value of `X`	Value of `Y`
`ratio_add<R1, R2>`	`R1::num * R2::den + R2::num * R1::den`	`R1::den * R2::den`
`ratio_subtract<R1, R2>`	`R1::num * R2::den - R2::num * R1::den`	`R1::den * R2::den`

`ratio_multiply<R1, R2>`	`R1::num * R2::num`	`R1::den * R2::den`
`ratio_divide<R1, R2>`	`R1::num * R2::den`	`R1::den * R2::num`

3. [Example:

```
static_assert(ratio_add<ratio<1,3>, ratio<1,6>>::num == 1, "1/3+1/6 == 1/2");
static_assert(ratio_add<ratio<1,3>, ratio<1,6>>::den == 2, "1/3+1/6 == 1/2");

static_assert(ratio_multiply<ratio<1,3>, ratio<3,2>>::num == 1, "1/3*3/2 == 1/2");
static_assert(ratio_multiply<ratio<1,3>, ratio<3,2>>::den == 2, "1/3*3/2 == 1/2");

// The following cases may cause the program to be ill-formed under
// some implementations

static_assert(ratio_add<ratio<1,INTMAX_MAX>, ratio<1,INTMAX_MAX>>::num == 2,
              "1/MAX+1/MAX == 2/MAX");
static_assert(ratio_add<ratio<1,INTMAX_MAX>, ratio<1,INTMAX_MAX>>::den == INTMAX_MAX,
              "1/MAX+1/MAX == 2/MAX");

static_assert(ratio_multiply<ratio<1,INTMAX_MAX>, ratio<INTMAX_MAX,2>>::num == 1,
              "1/MAX * MAX/2 == 1/2");
static_assert(ratio_multiply<ratio<1,INTMAX_MAX>, ratio<INTMAX_MAX,2>>::den == 2,
              "1/MAX * MAX/2 == 1/2");
```

-- end example]

Replace the title of 20.6.3 [ratio.comparison] as follows:

Comparison of `ratio` ~~type~~s

Acknowledgements

A great deal of thanks goes out to Anthony Williams for his initial proposed wording in N3131.

I would also like to thank Walter Brown, Pablo Halpern, and Howard Hinnant for their careful review of and very helpful comments on initial versions of this paper.