Doc No: N2144=07-0004
Date: 2007-1-11
Reply to: M.J. Kronenburg
M.Kronenburg@inter.nl.net

Proposal for exact specification of is_modulo

Contents

Chapter 1

Introduction

1.1 Motivation and Scope

With the introduction of the `numeric_limits` template in C++, the programmer can determine if an integer base type overflow is wrapped around using modulo arithemetic (`is_modulo` flag) or is trapped (`traps` flag). The exact meaning of modulo arithmetic however should also be defined, so that programmers know how the base type (whether signed or unsigned) exactly behaves at overflow when the `numeric_limits` template specialization `is_modulo` flag is `true`.
An important application of a modulo wrapped around signed integer base type is when not only wrap around is required, but also a signed subtraction result for determining whether one value is relatively smaller or larger than another. An example is using a system clock where the time difference between the current time and some preset time goes from negative to positive. Then the integer base type representing the time must be both signed and modulo wrapped around.

1.2 Impact on the Standard

In section 3.9.1 [basic.fundamental] item 4 it is stated that unsigned integer base types obey modulo arithmetic. This should be extended to signed integer base types, and refer to the `is_modulo` and `traps` flags of the `numeric_limits` template (section 18.2.1).
In section 18.2.1.2 [lib.numeric.limits.members] item 57 the description of the `is_modulo` flag of the `numeric_limits` template is extended with an exact definition of modulo arithmetic.

Chapter 2

Proposed Text

2.1 3.9.1 [basic.fundamental] item 4

In section 3.9.1 [basic.fundamental] item 4, the following is added:

Signed integers obey the laws of arithmetic modulo 2^n when its `numeric_limits` template specialization `is_modulo` flag is true, or traps overflow when its `traps` flag is true, see 18.2.1. The laws of arithmetic modulo 2^n are defined there.

2.2 18.2.1.2 [lib.numeric.limits.members] item 57

In section 18.2.1.2 [lib.numeric.limits.members] item 57 the line "A type is modulo if..." is deleted, and the following is added:

A type is modulo if an arithmetic result x that is not within its range between $x_{\min}$ and $x_{\max}$ inclusive (see `min()` and `max()` above), is wrapped around this range with
$x = x_{\min} + ((x - x_{\min}) \bmod (x_{\max} - x_{\min} + 1))$,
where $x \bmod y = x - y * \text{floor}(x/y)$, and where floor truncates downward (that is toward minus infinity). This means that the resulting x is within its range and differs from the original x by a multiple of $x_{\max} - x_{\min} + 1$. For unsigned integer base types, and for signed integer base types on twos complement machines, $x_{\max} - x_{\min} + 1 = 2^n$, where n is the number of bits.