Document number: P1956R0
Revises: D1956R1
Date: 2020-01-13
Project: ISO JTC1/SC22/WG21: Programming Language C++
Audience: LEWG, LWG
Reply to: Vincent Reverdy
École Normale Supérieure, rue d'Ulm, Paris, France
vince.rev@gmail.com

On the names of low-level bit manipulation functions

Abstract

We provide wording to rename the bit functions following National Body comments PL326, US327, GB332, US328, GB331 and following LEWG votes in Belfast 2019.

Contents

Introduction

In Belfast, and following several national body comments and SG6 suggestions, LEWG voted to rename some of the bit manipulation functions in order to avoid references to powers of two since manipulating bits should work in the future on bytes and machine words for which numeric operations are not defined.

0.1 Bit manipulation [bit]

0.1.1 General [bit.general]

1 The header bit provides components to access, manipulate and process both individual bits and bit sequences. Binary representation is assumed as in ISO/IEC 80000-13 item 13-9.

0.1.2 Header <bit> synopsis [bit.syn]

```
namespace std {
  // 0.1.3, bit_cast
  template<class To, class From>
    constexpr To bit_cast(const From& from) noexcept;

  // 0.1.4, single bit manipulation
  template<class T>
    constexpr bool has_single_bit(T x) noexcept;
  template<class T>
    constexpr T bit_ceil(T x);
  template<class T>
    constexpr T bit_floor(T x) noexcept;
  template<class T>
    constexpr T bit_width(T x) noexcept;

  // 0.1.5, rotating
  template<class T>
    [[nodiscard]] constexpr T rotl(T x, int s) noexcept;
  template<class T>
    [[nodiscard]] constexpr T rotr(T x, int s) noexcept;

  // 0.1.6, counting
  template<class T>
    constexpr int countl_zero(T x) noexcept;
  template<class T>
    constexpr int countl_one(T x) noexcept;
  template<class T>
    constexpr int countr_zero(T x) noexcept;
  template<class T>
    constexpr int countr_one(T x) noexcept;
  template<class T>
    constexpr int popcount(T x) noexcept;

  // 0.1.7, endian
  enum class endian {
    little = see below,
    big    = see below,
    native = see below
  };
}
```

0.1.3 Function template bit_cast [bit.cast]

Nothing to modify.

0.1.4 Single bit manipulation [bit.pow.two]

has_single_bit

```
template<class T>
```

```
constexpr bool has_single_bit(T x) noexcept;
```

1 `T` is an unsigned integer typebasic.fundamental.

2 *Returns:* `true` if `x` ~~is an integral power of two~~has only one bit set; `false` otherwise.

bit_ceil

```
template<class T>
  constexpr T bit_ceil(T x);
```

3 Let N be the smallest power of 2 greater than or equal to `x`.

4 `T` is an unsigned integer typebasic.fundamental.

5 N is representable as a value of type `T`.

6 *Returns:* N.

7 *Throws:* Nothing.

8 *Remarks:* A function call expression that violates the precondition in the element is not a core constant expressionexpr.const.

bit_floor

```
template<class T>
  constexpr T bit_floor(T x) noexcept;
```

9 `T` is an unsigned integer typebasic.fundamental.

10 *Returns:* ~~If `x == 0`, 0; otherwise the maximal value `y` such that `has_single_bit(y)` is `true` and `y <= x`.~~ If `x == 0`, 0; otherwise `x` where all the bits except the most significant one have been cleared.

bit_width

```
template<class T>
  constexpr T bit_width(T x) noexcept;
```

11 `T` is an unsigned integer typebasic.fundamental.

12 *Returns:* If `x == 0`, 0; otherwise one plus the base-2 logarithm of `x`, with any fractional part discarded.

0.1.5 Rotating [bit.rotate]

Nothing to modify.

0.1.6 Counting [bit.count]

Nothing to modify.

0.1.7 Endian [bit.endian]

Nothing to modify.

Acknowledgements

This work has been made possible thanks to the National Science Foundation through the awards CCF-1647432 and SI2-SSE-1642411, as well as French institutions École Normale Supérieure, INRIA, and PSL.