P3103

More bitset operations

Tokyo 2024

東京

Author: Jan Schultke
Presenter: Jan Schultke
Audience: SG18
Project: ISO/IEC 14882 Programming Languages — C++, ISO/IEC JTC1/SC22/WG21

Contents

1. Introduction

History

- `std::bitset` standardized in C++98
- minor changes over time
  - most recently, **P2417R2:** A more constexpr bitset (C++23)
- meanwhile, **P0553R4:** Bit operations (C++20)
  - `std::rotl`, `std::popcount`, `std::countl_zero`, …
  - `std::bitset` unchanged

Goals

1. Add most `<bit>` functionality to `std::bitset`.
2. Additional utility (`bitset::reverse`, …).

2. Motivation

- `std::bitset` is useful and worth maintaining.
  - GitHub code search for `/std::bitset language:c++/` ⟶ 73.2K files
- Common complaints:
  - (Mandatory range checks and exceptions.)
  - Difficult to find first/last set bit.
  - Difficult to iterate over all set bits.
  - Essentially, zero-overhead principle violations.
- Hardware support for bit-counting, bit-reversal, …
  - For example:
    - `tzcnt`/`ctz` for counting trailing zeros
    - `bswap`/`rbit` for reversing bytes/bits
  - Difficult to utilize by the user.

3. Impact on the standard

Add the following member functions to `std::bitset`:

`<bit>` function template	Proposed `bitset` member
`std::has_single_bit(T)`	`one() const noexcept`

`std::countl_zero(T)`	`countl_zero() const noexcept`
	`countl_zero(size_t) const`

`std::count{l,r}_{zero,one}(T)`	`count{l,r}_{…}() const noexcept`
	`count{l,r}_{…}(size_t) const`

`std::rotl(T, int)`	`rotl(size_t)`
`std::rotr(T, int)`	`rotr(size_t)`
`std::bit_reverse(T) (P3104)`	`reverse() noexcept`

4. Implementation experience

- GitHub: `ClaasBontus/bitset2` *basically* implements all proposed functions.
  - (for iteration, it has `find_next_one(size_t)` (exclusive index))
- Many other feature-rich `bitset` implementations exist.
- This isn‘t rocket science.

5. Design

Design principles

1. Match the existing design of `std::bitset`.
2. Prefer in-place operations.

Questions

- *“Why `one()` instead of `has_single_bit()`?“*
  - To match conventions (`any()`, `all()`, `none()`).
- *“Why take `size_t` in counting overloads and `rotl(size_t)`?“*
  - To match conventions (`get(size_t)`, …).
- *“Are there other options for supporting iteration?“*
  - Yes, see next slide.

5. Design

P3103

```
bitset<N> bits;
for (size_t i = 0; i != N; ++i) {
    i += bits.countr_zero(i);
    if (i == N) break;
    // ...
}
```

ClaasBontus/bitset2

```
bitset<N> bits;
size_t i = 0;
while ((i = bits.find_next_one(i))
           != bitset<N>::npos) {
    // ...
}
```

Infallible `countr_zero`

```
bitset<N> bits;
for (size_t i = 0; (i += bits.countr_zero(i)) != N; ++i) {
    // ...
}
```

References

Jens Maurer; **P0553R4** Bit operations
https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2019/p0553r4.html

Daniil Goncharov; **P2417R0** A more constexpr bitset
https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2021/p2417r0.pdf

Jan Schultke; **P3103** More bitset operations (latest revision)
https://eisenwave.github.io/cpp-proposals/more-bitset-operations.html

Jan Schultke; **P3104** Bit permutations (latest revision)
https://eisenwave.github.io/cpp-proposals/bit-permutations.html