A more constexpr bitset

Document Number:	P1251r0
Date:	2018-10-04
Audience:	LEWG
Reply to:	Morris Hafner
	hafnermorris@gmail.com

1 Motivation

As of N4762 only the default constructor, the constructor accepting an `unsigned long long` and `operator[]` of `bitset` are marked as constexpr. With the exception of the functions accepting or returning a `basic_string`, there is no reason the rest of the class cannot be made `constexpr`.

The lack of `constexpr` for most member functions was probably due to the non-trivial destructor of `bitset::reference`. However, in libc++ it is trivially destructible already. In libstdc++ it is an empty destructor. This behavior should be standardized.

2 Proposed Changes

Mark every member function except for the `basic_string` constructor and `to_string` as `constexpr`. Make `bitset::reference` trivially destructible and `constexpr`. Add a constructor accepting a `basic_string_view`.

3 Impact on the Standard

This proposal is a pure library addition and does not require new language features.

4 Proposed Wording

Change *19.9.1* of N4762 to the following:

```
#include <string>
#include <iosfwd>   // for istream (27.7.1), ostream (27.7.2), see 27.3.1

namespace std {
  template<size_t N> class bitset;

  // 19.9.4, bitset operators
  template<size_t N>
    constexpr bitset<N> operator&(const bitset<N>&, const bitset<N>&) noexcept;
  template<size_t N>
    constexpr bitset<N> operator|(const bitset<N>&, const bitset<N>&) noexcept;
  template<size_t N>
    constexpr bitset<N> operator^(const bitset<N>&, const bitset<N>&) noexcept;
  template<class charT, class traits, size_t N>
    basic_istream<charT, traits>&
      operator>>(basic_istream<charT, traits>& is, bitset<N>& x);
  template<class charT, class traits, size_t N>
    basic_ostream<charT, traits>&
      operator<<(basic_ostream<charT, traits>& os, const bitset<N>& x);
}
```

Change *19.9.2* of N4762 to the following:

```
namespace std {
  template<size_t N> class bitset {
  public:
    // bit reference
    class reference {
      friend class bitset;
      constexpr reference() noexcept;

    public:
      constexpr reference(const reference&) = default;
      ~reference();
      constexpr reference& operator=(bool x) noexcept;            // for b[i] = x;
      constexpr reference& operator=(const reference&) noexcept;  // for b[i] = b[j];
      constexpr bool operator~() const noexcept;                  // flips the bit
      constexpr operator bool() const noexcept;                   // for x = b[i];
      constexpr reference& flip() noexcept;                       // for b[i].flip();
    };

    // 19.9.2.1, constructors
    constexpr bitset() noexcept;
    constexpr bitset(unsigned long long val) noexcept;
    template<class charT, class traits, class Allocator>
      explicit bitset(
        const basic_string<charT, traits, Allocator>& str,
        typename basic_string<charT, traits, Allocator>::size_type pos = 0,
        typename basic_string<charT, traits, Allocator>::size_type n
          = basic_string<charT, traits, Allocator>::npos,
        charT zero = charT('0'),
        charT one = charT('1'));
    template<class charT, class traits>
      constexpr explicit bitset(
        const basic_string_view<charT, traits>& str,
        typename basic_string_view<charT, traits>::size_type pos = 0,
        typename basic_string_view<charT, traits>::size_type n
          = basic_string_view<charT, traits>::npos,
        charT zero = charT('0'),
        charT one = charT('1'));
    template<class charT>
      constexpr explicit bitset(
        const charT* str,
        typename basic_string<charT>::size_type n = basic_string<charT>::npos,
        charT zero = charT('0'),
        charT one = charT('1'));

    // 19.9.2.2, bitset operations
    constexpr bitset<N>& operator&=(const bitset<N>& rhs) noexcept;
    constexpr bitset<N>& operator|=(const bitset<N>& rhs) noexcept;
    constexpr bitset<N>& operator^=(const bitset<N>& rhs) noexcept;
    constexpr bitset<N>& operator<<=(size_t pos) noexcept;
    constexpr bitset<N>& operator>>=(size_t pos) noexcept;
    constexpr bitset<N>& set() noexcept;
    constexpr bitset<N>& set(size_t pos, bool val = true);
    constexpr bitset<N>& reset() noexcept;
    constexpr bitset<N>& reset(size_t pos);
    constexpr bitset<N>  operator~() const noexcept;
    constexpr bitset<N>& flip() noexcept;
    constexpr bitset<N>& flip(size_t pos);

    // element access
    constexpr bool operator[](size_t pos) const;        // for b[i];
    constexpr reference operator[](size_t pos);       // for b[i];

    constexpr unsigned long to_ulong() const;
    constexpr unsigned long long to_ullong() const;
    template<class charT = char,
             class traits = char_traits<charT>,
             class Allocator = allocator<charT>>
      basic_string<charT, traits, Allocator>
        to_string(charT zero = charT('0'), charT one = charT('1')) const;

    constexpr size_t count() const noexcept;
    constexpr constexpr size_t size() const noexcept;
    constexpr bool operator==(const bitset<N>& rhs) const noexcept;
    constexpr bool operator!=(const bitset<N>& rhs) const noexcept;
    constexpr bool test(size_t pos) const;
    constexpr bool all() const noexcept;
    constexpr bool any() const noexcept;
    constexpr bool none() const noexcept;
    constexpr bitset<N> operator<<(size_t pos) const noexcept;
    constexpr bitset<N> operator>>(size_t pos) const noexcept;
  };

  // 19.9.3, hash support
  template<class T> struct hash;
  template<size_t N> struct hash<bitset<N>>;
}
```