Making `std::string` constexpr

Document #:	D0980R1
Date:	2019-07-19
Project:	Programming Language C++
Audience:	LEWG

Reply-to: Louis Dionne <ldionne@apple.com>

1 Revision history

- R0 – Initial draft
- R1 –
  - Editorial: remove stray \ character, add missing spaces and fix placement of `constexpr` with `[[nodiscard]]`
  - Add feature-test macro `__cpp_lib_constexpr_string`
  - Add a note to the editor to clarify the intent of the wording

2 Abstract

`std::string` is not currently `constexpr` friendly. With the loosening of requirements on `constexpr` in [P0784R1] and related papers, we can now make it so, and we should in order to support the `constexpr` reflection effort (and other evident use cases).

3 Encountered issues

We surveyed the implementation of `std::string` in libc++ and noted the following issues:

- We need to make `char_traits` `constexpr` for at least its `char` specialization.
- At least one implementation (libc++) currently explicitly instantiates the `basic_string<char>` in a shared library. We need to investigate whether `constexpr` has any bad interactions with this. We don't suspect this should be the case, as long as the member functions of `basic_string<char>` are defined in the headers.
- We expected the SSO to be a problem, but it is implemented with unions and no `reinterpret_cast`s are required.

4 Proposed wording

This wording is based on the working draft [N4727].

Direction to the editor: please also apply `constexpr` to `std::basic_string`'s functions missing from this paper, at the editor's discretion.

In [**support.limits.general**], add the new feature test macro `__cpp_lib_constexpr_string` with the corresponding value for header `<string>` to Table 36 [**tab:support.ft**].

Change in [**string.syn**] **20.3.1**:

```
#include <initializer_list>

namespace std {
  // 20.2, character traits
  template<class charT> struct char_traits;
  template<> struct char_traits<char>;
  template<> struct char_traits<char16_t>;
  template<> struct char_traits<char32_t>;
  template<> struct char_traits<wchar_t>;

  // 20.3.2, basic_string
  template<class charT, class traits = char_traits<charT>, class Allocator = allocator<charT>>
    class basic_string;

  template<class charT, class traits, class Allocator>
    constexpr basic_string<charT, traits, Allocator>
      operator+(const basic_string<charT, traits, Allocator>& lhs,
                const basic_string<charT, traits, Allocator>& rhs);
  template<class charT, class traits, class Allocator>
    constexpr basic_string<charT, traits, Allocator>
      operator+(basic_string<charT, traits, Allocator>&& lhs,
                const basic_string<charT, traits, Allocator>& rhs);
  template<class charT, class traits, class Allocator>
    constexpr basic_string<charT, traits, Allocator>
      operator+(const basic_string<charT, traits, Allocator>& lhs,
                basic_string<charT, traits, Allocator>&& rhs);
  template<class charT, class traits, class Allocator>
    constexpr basic_string<charT, traits, Allocator>
      operator+(basic_string<charT, traits, Allocator>&& lhs,
                basic_string<charT, traits, Allocator>&& rhs);
  template<class charT, class traits, class Allocator>
    constexpr basic_string<charT, traits, Allocator>
      operator+(const charT* lhs,
                const basic_string<charT, traits, Allocator>& rhs);
  template<class charT, class traits, class Allocator>
    constexpr basic_string<charT, traits, Allocator>
      operator+(const charT* lhs,
                basic_string<charT, traits, Allocator>&& rhs);
  template<class charT, class traits, class Allocator>
    constexpr basic_string<charT, traits, Allocator>
```

```
    operator+(charT lhs,
              const basic_string<charT, traits, Allocator>& rhs);
template<class charT, class traits, class Allocator>
  constexpr basic_string<charT, traits, Allocator>
    operator+(charT lhs,
              basic_string<charT, traits, Allocator>&& rhs);
template<class charT, class traits, class Allocator>
  constexpr basic_string<charT, traits, Allocator>
    operator+(const basic_string<charT, traits, Allocator>& lhs,
              const charT* rhs);
template<class charT, class traits, class Allocator>
  constexpr basic_string<charT, traits, Allocator>
    operator+(basic_string<charT, traits, Allocator>&& lhs,
              const charT* rhs);
template<class charT, class traits, class Allocator>
  constexpr basic_string<charT, traits, Allocator>
    operator+(const basic_string<charT, traits, Allocator>& lhs,
              charT rhs);
template<class charT, class traits, class Allocator>
  constexpr basic_string<charT, traits, Allocator>
    operator+(basic_string<charT, traits, Allocator>&& lhs,
              charT rhs);

template<class charT, class traits, class Allocator>
  constexpr
  bool operator==(const basic_string<charT, traits, Allocator>& lhs,
                  const basic_string<charT, traits, Allocator>& rhs) noexcept;
template<class charT, class traits, class Allocator>
  constexpr
  bool operator==(const charT* lhs,
                  const basic_string<charT, traits, Allocator>& rhs);
template<class charT, class traits, class Allocator>
  constexpr
  bool operator==(const basic_string<charT, traits, Allocator>& lhs,
                  const charT* rhs);
template<class charT, class traits, class Allocator>
  constexpr
  bool operator!=(const basic_string<charT, traits, Allocator>& lhs,
                  const basic_string<charT, traits, Allocator>& rhs) noexcept;
template<class charT, class traits, class Allocator>
  constexpr
  bool operator!=(const charT* lhs,
                  const basic_string<charT, traits, Allocator>& rhs);
template<class charT, class traits, class Allocator>
  constexpr
  bool operator!=(const basic_string<charT, traits, Allocator>& lhs,
                  const charT* rhs);

template<class charT, class traits, class Allocator>
  constexpr
  bool operator< (const basic_string<charT, traits, Allocator>& lhs,
```

```
                const basic_string<charT, traits, Allocator>& rhs) noexcept;
template<class charT, class traits, class Allocator>
  constexpr
  bool operator< (const basic_string<charT, traits, Allocator>& lhs,
                const charT* rhs);
template<class charT, class traits, class Allocator>
  constexpr
  bool operator< (const charT* lhs,
                const basic_string<charT, traits, Allocator>& rhs);
template<class charT, class traits, class Allocator>
  constexpr
  bool operator> (const basic_string<charT, traits, Allocator>& lhs,
                const basic_string<charT, traits, Allocator>& rhs) noexcept;
template<class charT, class traits, class Allocator>
  constexpr
  bool operator> (const basic_string<charT, traits, Allocator>& lhs,
                const charT* rhs);
template<class charT, class traits, class Allocator>
  constexpr
  bool operator> (const charT* lhs,
                const basic_string<charT, traits, Allocator>& rhs);

template<class charT, class traits, class Allocator>
  constexpr
  bool operator<=(const basic_string<charT, traits, Allocator>& lhs,
                const basic_string<charT, traits, Allocator>& rhs) noexcept;
template<class charT, class traits, class Allocator>
  constexpr
  bool operator<=(const basic_string<charT, traits, Allocator>& lhs,
                const charT* rhs);
template<class charT, class traits, class Allocator>
  constexpr
  bool operator<=(const charT* lhs,
                const basic_string<charT, traits, Allocator>& rhs);
template<class charT, class traits, class Allocator>
  constexpr
  bool operator>=(const basic_string<charT, traits, Allocator>& lhs,
                const basic_string<charT, traits, Allocator>& rhs) noexcept;
template<class charT, class traits, class Allocator>
  constexpr
  bool operator>=(const basic_string<charT, traits, Allocator>& lhs,
                const charT* rhs);
template<class charT, class traits, class Allocator>
  constexpr
  bool operator>=(const charT* lhs,
                const basic_string<charT, traits, Allocator>& rhs);

// 20.3.3.8, swap
template<class charT, class traits, class Allocator>
  constexpr
  void swap(basic_string<charT, traits, Allocator>& lhs,
```

```
            basic_string<charT, traits, Allocator>& rhs)
    noexcept(noexcept(lhs.swap(rhs)));

// 20.3.3.9, inserters and extractors
template<class charT, class traits, class Allocator>
  basic_istream<charT, traits>&
    operator>>(basic_istream<charT, traits>& is,
               basic_string<charT, traits, Allocator>& str);
template<class charT, class traits, class Allocator>
  basic_ostream<charT, traits>&
    operator<<(basic_ostream<charT, traits>& os,
               const basic_string<charT, traits, Allocator>& str);
template<class charT, class traits, class Allocator>
  basic_istream<charT, traits>&
    getline(basic_istream<charT, traits>& is,
            basic_string<charT, traits, Allocator>& str,
            charT delim);
template<class charT, class traits, class Allocator>
  basic_istream<charT, traits>&
    getline(basic_istream<charT, traits>&& is,
            basic_string<charT, traits, Allocator>& str,
            charT delim);
template<class charT, class traits, class Allocator>
  basic_istream<charT, traits>&
    getline(basic_istream<charT, traits>& is,
            basic_string<charT, traits, Allocator>& str);
template<class charT, class traits, class Allocator>
  basic_istream<charT, traits>&
    getline(basic_istream<charT, traits>&& is,
            basic_string<charT, traits, Allocator>& str);

// basic_string typedef names
using string    = basic_string<char>;
using u16string = basic_string<char16_t>;
using u32string = basic_string<char32_t>;
using wstring   = basic_string<wchar_t>;

// 20.3.4, numeric conversions
int stoi(const string& str, size_t* idx = nullptr, int base = 10);
long stol(const string& str, size_t* idx = nullptr, int base = 10);
unsigned long stoul(const string& str, size_t* idx = nullptr, int base = 10);
long long stoll(const string& str, size_t* idx = nullptr, int base = 10);
unsigned long long stoull(const string& str, size_t* idx = nullptr, int base = 10);
float stof(const string& str, size_t* idx = nullptr);
double stod(const string& str, size_t* idx = nullptr);
long double stold(const string& str, size_t* idx = nullptr);
string to_string(int val);
string to_string(unsigned val);
string to_string(long val);
string to_string(unsigned long val);
string to_string(long long val);
```

```
string to_string(unsigned long long val);
string to_string(float val);
string to_string(double val);
string to_string(long double val);

int stoi(const wstring& str, size_t* idx = nullptr, int base = 10);
long stol(const wstring& str, size_t* idx = nullptr, int base = 10);
unsigned long stoul(const wstring& str, size_t* idx = nullptr, int base = 10);
long long stoll(const wstring& str, size_t* idx = nullptr, int base = 10);
unsigned long long stoull(const wstring& str, size_t* idx = nullptr, int base = 10);
float stof(const wstring& str, size_t* idx = nullptr);
double stod(const wstring& str, size_t* idx = nullptr);
long double stold(const wstring& str, size_t* idx = nullptr);
wstring to_wstring(int val);
wstring to_wstring(unsigned val);
wstring to_wstring(long val);
wstring to_wstring(unsigned long val);
wstring to_wstring(long long val);
wstring to_wstring(unsigned long long val);
wstring to_wstring(float val);
wstring to_wstring(double val);
wstring to_wstring(long double val);

namespace pmr {
  template<class charT, class traits = char_traits<charT>>
    using basic_string = std::basic_string<charT, traits, polymorphic_allocator<charT>>;

  using string    = basic_string<char>;
  using u16string = basic_string<char16_t>;
  using u32string = basic_string<char32_t>;
  using wstring   = basic_string<wchar_t>;
}

// 20.3.5, hash support
template<class T> struct hash;
template<> struct hash<string>;
template<> struct hash<u16string>;
template<> struct hash<u32string>;
template<> struct hash<wstring>;
template<> struct hash<pmr::string>;
template<> struct hash<pmr::u16string>;
template<> struct hash<pmr::u32string>;
template<> struct hash<pmr::wstring>;

inline namespace literals {
inline namespace string_literals {
  // 20.3.6, suffix for basic_string literals
  constexpr string    operator""s(const char* str, size_t len);
  constexpr u16string operator""s(const char16_t* str, size_t len);
  constexpr u32string operator""s(const char32_t* str, size_t len);
  constexpr wstring   operator""s(const wchar_t* str, size_t len);
```

```
    }
    }
  }
```

In [**basic.string**]/**20.3.2**:

```
namespace std {
  template<class charT, class traits = char_traits<charT>,
           class Allocator = allocator<charT>>
  class basic_string {
  public:
    // types
    using traits_type            = traits;
    using value_type             = charT;
    using allocator_type         = Allocator;
    using size_type              = typename allocator_traits<Allocator>::size_type;
    using difference_type        = typename allocator_traits<Allocator>::difference_type;
    using pointer                = typename allocator_traits<Allocator>::pointer;
    using const_pointer          = typename allocator_traits<Allocator>::const_pointer;
    using reference              = value_type&;
    using const_reference        = const value_type&;

    using iterator               = implementation-defined // see 21.2
    using const_iterator         = implementation-defined // see 21.2
    using reverse_iterator       = std::reverse_iterator<iterator>;
    using const_reverse_iterator = std::reverse_iterator<const_iterator>;
    static const size_type npos  = -1;

    // 20.3.2.2, construct/copy/destroy
    constexpr
    basic_string() noexcept(noexcept(Allocator())) : basic_string(Allocator()) { }
    constexpr
    explicit basic_string(const Allocator& a) noexcept;
    constexpr
    basic_string(const basic_string& str);
    constexpr
    basic_string(basic_string&& str) noexcept;
    constexpr
    basic_string(const basic_string& str, size_type pos, const Allocator& a = Allocator());
    constexpr
    basic_string(const basic_string& str, size_type pos, size_type n,
                 const Allocator& a = Allocator());
    template<class T>
      constexpr
      basic_string(const T& t, size_type pos, size_type n, const Allocator& a = Allocator());
    template<class T>
      constexpr explicit basic_string(const T& t, const Allocator& a = Allocator());
    constexpr basic_string(const charT* s, size_type n, const Allocator& a = Allocator());
    constexpr basic_string(const charT* s, const Allocator& a = Allocator());
    constexpr basic_string(size_type n, charT c, const Allocator& a = Allocator());
    template<class InputIterator>
      constexpr
```

```
  basic_string(InputIterator begin, InputIterator end, const Allocator& a = Allocator());
constexpr basic_string(initializer_list<charT>, const Allocator& = Allocator());
constexpr basic_string(const basic_string&, const Allocator&);
constexpr basic_string(basic_string&&, const Allocator&);

constexpr ~basic_string();
constexpr basic_string& operator=(const basic_string& str);
constexpr basic_string& operator=(basic_string&& str)
  noexcept(allocator_traits<Allocator>::propagate_on_container_move_assignment::value ||
           allocator_traits<Allocator>::is_always_equal::value);
template<class T>
  constexpr basic_string& operator=(const T& t);
constexpr basic_string& operator=(const charT* s);
constexpr basic_string& operator=(charT c);
constexpr basic_string& operator=(initializer_list<charT>);

// 20.3.2.3, iterators
constexpr iterator       begin() noexcept;
constexpr const_iterator begin() const noexcept;
constexpr iterator       end() noexcept;
constexpr const_iterator end() const noexcept;

constexpr reverse_iterator       rbegin() noexcept;
constexpr const_reverse_iterator rbegin() const noexcept;
constexpr reverse_iterator       rend() noexcept;
constexpr const_reverse_iterator rend() const noexcept;

constexpr const_iterator         cbegin() const noexcept;
constexpr const_iterator         cend() const noexcept;
constexpr const_reverse_iterator crbegin() const noexcept;
constexpr const_reverse_iterator crend() const noexcept;

// 20.3.2.4, capacity
constexpr size_type size() const noexcept;
constexpr size_type length() const noexcept;
constexpr size_type max_size() const noexcept;
constexpr void resize(size_type n, charT c);
constexpr void resize(size_type n);
constexpr size_type capacity() const noexcept;
constexpr void reserve(size_type res_arg);
constexpr void shrink_to_fit();
constexpr void clear() noexcept;
[[nodiscard]] constexpr bool empty() const noexcept;

// 20.3.2.5, element access
constexpr const_reference operator[](size_type pos) const;
constexpr reference       operator[](size_type pos);
constexpr const_reference at(size_type n) const;
constexpr reference       at(size_type n);

constexpr const charT& front() const;
```

```
constexpr charT&       front();
constexpr const charT& back() const;
constexpr charT&       back();

// 20.3.2.6, modifiers
constexpr basic_string& operator+=(const basic_string& str);
template<class T>
  constexpr basic_string& operator+=(const T& t);
constexpr basic_string& operator+=(const charT* s);
constexpr basic_string& operator+=(charT c);
constexpr basic_string& operator+=(initializer_list<charT>);
constexpr basic_string& append(const basic_string& str);
constexpr basic_string& append(const basic_string& str, size_type pos, size_type n = npos);
template<class T>
  constexpr basic_string& append(const T& t);
template<class T>
  constexpr basic_string& append(const T& t, size_type pos, size_type n = npos);
constexpr basic_string& append(const charT* s, size_type n);
constexpr basic_string& append(const charT* s);
constexpr basic_string& append(size_type n, charT c);
template<class InputIterator>
  constexpr basic_string& append(InputIterator first, InputIterator last);
constexpr basic_string& append(initializer_list<charT>);

constexpr void push_back(charT c);

constexpr basic_string& assign(const basic_string& str);
constexpr basic_string& assign(basic_string&& str)
  noexcept(allocator_traits<Allocator>::propagate_on_container_move_assignment::value ||
           allocator_traits<Allocator>::is_always_equal::value);
constexpr basic_string& assign(const basic_string& str, size_type pos, size_type n = npos);
template<class T>
  constexpr basic_string& assign(const T& t);
template<class T>
  constexpr basic_string& assign(const T& t, size_type pos, size_type n = npos);
constexpr basic_string& assign(const charT* s, size_type n);
constexpr basic_string& assign(const charT* s);
constexpr basic_string& assign(size_type n, charT c);
template<class InputIterator>
  constexpr basic_string& assign(InputIterator first, InputIterator last);
constexpr basic_string& assign(initializer_list<charT>);

constexpr basic_string& insert(size_type pos, const basic_string& str);
constexpr basic_string& insert(size_type pos1, const basic_string& str,
                                          size_type pos2, size_type n = npos);
template<class T>
  constexpr basic_string& insert(size_type pos, const T& t);
template<class T>
  constexpr basic_string& insert(size_type pos1, const T& t, size_type pos2, size_type n = npos);
constexpr basic_string& insert(size_type pos, const charT* s, size_type n);
constexpr basic_string& insert(size_type pos, const charT* s);
```

```
constexpr basic_string& insert(size_type pos, size_type n, charT c);
constexpr iterator insert(const_iterator p, charT c);
constexpr iterator insert(const_iterator p, size_type n, charT c);
template<class InputIterator>
  constexpr iterator insert(const_iterator p, InputIterator first, InputIterator last);
constexpr iterator insert(const_iterator p, initializer_list<charT>);

constexpr basic_string& erase(size_type pos = 0, size_type n = npos);
constexpr iterator erase(const_iterator p);
constexpr iterator erase(const_iterator first, const_iterator last);

constexpr void pop_back();

constexpr basic_string& replace(size_type pos1, size_type n1, const basic_string& str);
constexpr basic_string& replace(size_type pos1, size_type n1, const basic_string& str,
                                size_type pos2, size_type n2 = npos);
template<class T>
  constexpr basic_string& replace(size_type pos1, size_type n1, const T& t);
template<class T>
  constexpr basic_string& replace(size_type pos1, size_type n1, const T& t,
                                  size_type pos2, size_type n2 = npos);
constexpr basic_string& replace(size_type pos, size_type n1, const charT* s, size_type n2);
constexpr basic_string& replace(size_type pos, size_type n1, const charT* s);
constexpr basic_string& replace(size_type pos, size_type n1, size_type n2, charT c);

constexpr
basic_string& replace(const_iterator i1, const_iterator i2, const basic_string& str);
template<class T>
  constexpr
  basic_string& replace(const_iterator i1, const_iterator i2, const T& t);
constexpr
basic_string& replace(const_iterator i1, const_iterator i2, const charT* s, size_type n);
constexpr
basic_string& replace(const_iterator i1, const_iterator i2, const charT* s);
constexpr
basic_string& replace(const_iterator i1, const_iterator i2, size_type n, charT c);
template<class InputIterator>
  constexpr basic_string& replace(const_iterator i1, const_iterator i2,
                                  InputIterator j1, InputIterator j2);
constexpr basic_string& replace(const_iterator, const_iterator, initializer_list<charT>);

constexpr size_type copy(charT* s, size_type n, size_type pos = 0) const;

constexpr void swap(basic_string& str)
  noexcept(allocator_traits<Allocator>::propagate_on_container_swap::value ||
           allocator_traits<Allocator>::is_always_equal::value);

// 20.3.2.7, string operations
constexpr const charT* c_str() const noexcept;
constexpr const charT* data() const noexcept;
constexpr charT* data() noexcept;
```

```
constexpr operator basic_string_view<charT, traits>() const noexcept;
constexpr allocator_type get_allocator() const noexcept;

template<class T>
  constexpr size_type find (const T& t, size_type pos = 0) const;
constexpr size_type find (const basic_string& str, size_type pos = 0) const noexcept;
constexpr size_type find (const charT* s, size_type pos, size_type n) const;
constexpr size_type find (const charT* s, size_type pos = 0) const;
constexpr size_type find (charT c, size_type pos = 0) const;
template<class T>
  constexpr size_type rfind(const T& t, size_type pos = npos) const;
constexpr size_type rfind(const basic_string& str, size_type pos = npos) const noexcept;
constexpr size_type rfind(const charT* s, size_type pos, size_type n) const;
constexpr size_type rfind(const charT* s, size_type pos = npos) const;
constexpr size_type rfind(charT c, size_type pos = npos) const;

template<class T>
  constexpr size_type find_first_of(const T& t, size_type pos = 0) const;
constexpr size_type find_first_of(const basic_string& str, size_type pos = 0) const noexcept;
constexpr size_type find_first_of(const charT* s, size_type pos, size_type n) const;
constexpr size_type find_first_of(const charT* s, size_type pos = 0) const;
constexpr size_type find_first_of(charT c, size_type pos = 0) const;
template<class T>
  constexpr size_type find_last_of (const T& t, size_type pos = npos) const;
constexpr size_type find_last_of (const basic_string& str, size_type pos = npos) const noexcept;
constexpr size_type find_last_of (const charT* s, size_type pos, size_type n) const;
constexpr size_type find_last_of (const charT* s, size_type pos = npos) const;
constexpr size_type find_last_of (charT c, size_type pos = npos) const;

template<class T>
  constexpr
  size_type find_first_not_of(const T& t, size_type pos = 0) const;
constexpr size_type
find_first_not_of(const basic_string& str, size_type pos = 0) const noexcept;
constexpr size_type
find_first_not_of(const charT* s, size_type pos, size_type n) const;
constexpr size_type
find_first_not_of(const charT* s, size_type pos = 0) const;
constexpr size_type
find_first_not_of(charT c, size_type pos = 0) const;
template<class T>
  constexpr
  size_type find_last_not_of (const T& t, size_type pos = npos) const;
constexpr
size_type find_last_not_of (const basic_string& str, size_type pos = npos) const noexcept;
constexpr
size_type find_last_not_of (const charT* s, size_type pos, size_type n) const;
constexpr
size_type find_last_not_of (const charT* s, size_type pos = npos) const;
constexpr
size_type find_last_not_of (charT c, size_type pos = npos) const;
```

```
    constexpr basic_string substr(size_type pos = 0, size_type n = npos) const;
    template<class T>
      constexpr int compare(const T& t) const;
    template<class T>
      constexpr int compare(size_type pos1, size_type n1, const T& t) const;
    template<class T>
      constexpr int compare(size_type pos1, size_type n1, const T& t,
                                    size_type pos2, size_type n2 = npos) const;
    constexpr int compare(const basic_string& str) const noexcept;
    constexpr int compare(size_type pos1, size_type n1, const basic_string& str) const;
    constexpr int compare(size_type pos1, size_type n1, const basic_string& str,
                                   size_type pos2, size_type n2 = npos) const;
    constexpr int compare(const charT* s) const;
    constexpr int compare(size_type pos1, size_type n1, const charT* s) const;
    constexpr int compare(size_type pos1, size_type n1, const charT* s, size_type n2) const;

    constexpr bool starts_with(basic_string_view<charT, traits> x) const noexcept;
    constexpr bool starts_with(charT x) const noexcept;
    constexpr bool starts_with(const charT* x) const;
    constexpr bool ends_with(basic_string_view<charT, traits> x) const noexcept;
    constexpr bool ends_with(charT x) const noexcept;
    constexpr bool ends_with(const charT* x) const;
  };

  template<class InputIterator,
           class Allocator = allocator<typename iterator_traits<InputIterator>::value_type>>
    basic_string(InputIterator, InputIterator, Allocator = Allocator())
      -> basic_string<typename iterator_traits<InputIterator>::value_type,
                      char_traits<typename iterator_traits<InputIterator>::value_type>,
                      Allocator>;

  template<class charT,
           class traits,
           class Allocator = allocator<charT>>
    explicit basic_string(basic_string_view<charT, traits>, const Allocator& = Allocator())
      -> basic_string<charT, traits, Allocator>;

  template<class charT,
           class traits,
           class Allocator = allocator<charT>>
    basic_string(basic_string_view<charT, traits>,
                 typename see below::size_type, typename see below::size_type,
                 const Allocator& = Allocator())
      -> basic_string<charT, traits, Allocator>;
}
```

In [**string.cons**]/**20.3.2.2**:

```
  constexpr explicit basic_string(const Allocator& a) noexcept;

  constexpr basic_string(const basic_string& str);
```

```
constexpr basic_string(basic_string&& str) noexcept;

constexpr basic_string(const basic_string& str, size_type pos,
                                const Allocator& a = Allocator());
constexpr basic_string(const basic_string& str, size_type pos, size_type n,
                                const Allocator& a = Allocator());

template<class T>
  constexpr
  basic_string(const T& t, size_type pos, size_type n, const Allocator& a = Allocator());
  constexpr basic_string(sv.substr(pos, n), a);


template<class T>
  constexpr explicit basic_string(const T& t, const Allocator& a = Allocator());

constexpr basic_string(const charT* s, size_type n, const Allocator& a = Allocator());

constexpr basic_string(const charT* s, const Allocator& a = Allocator());

constexpr basic_string(size_type n, charT c, const Allocator& a = Allocator());

template<class InputIterator>
  constexpr
  basic_string(InputIterator begin, InputIterator end, const Allocator& a = Allocator());

constexpr basic_string(initializer_list<charT> il, const Allocator& a = Allocator());

constexpr basic_string(const basic_string& str, const Allocator& alloc);
constexpr basic_string(basic_string&& str, const Allocator& alloc);

constexpr basic_string& operator=(const basic_string& str);

constexpr basic_string& operator=(basic_string&& str)
  noexcept(allocator_traits<Allocator>::propagate_on_container_move_assignment::value ||
           allocator_traits<Allocator>::is_always_equal::value);

template<class T>
  constexpr basic_string& operator=(const T& t);

constexpr basic_string& operator=(const charT* s);

constexpr basic_string& operator=(charT c);

constexpr basic_string& operator=(initializer_list<charT> il);
```

In [**string.iterators**]/**20.3.2.3**:

```
constexpr iterator       begin() noexcept;
constexpr const_iterator begin() const noexcept;
constexpr const_iterator cbegin() const noexcept;

constexpr iterator       end() noexcept;
constexpr const_iterator end() const noexcept;
constexpr const_iterator cend() const noexcept;
```

```
constexpr reverse_iterator       rbegin() noexcept;
constexpr const_reverse_iterator rbegin() const noexcept;
constexpr const_reverse_iterator crbegin() const noexcept;

constexpr reverse_iterator       rend() noexcept;
constexpr const_reverse_iterator rend() const noexcept;
constexpr const_reverse_iterator crend() const noexcept;
```

In [**string.capacity**]/**20.3.2.4**:

```
constexpr size_type size() const noexcept;

constexpr size_type length() const noexcept;

constexpr size_type max_size() const noexcept;

constexpr void resize(size_type n, charT c);

constexpr void resize(size_type n);

constexpr size_type capacity() const noexcept;

constexpr void reserve(size_type res_arg);

constexpr void shrink_to_fit();

constexpr void clear() noexcept;

[[nodiscard]] constexpr bool empty() const noexcept;
```

In [**string.access**]/**20.3.2.5**:

```
constexpr const_reference operator[](size_type pos) const;
constexpr reference       operator[](size_type pos);

constexpr const_reference at(size_type pos) const;
constexpr reference       at(size_type pos);

constexpr const charT& front() const;
constexpr charT& front();

constexpr const charT& back() const;
constexpr charT& back();
```

In [**string.modifiers**]/**20.3.2.6**:

```
constexpr basic_string& operator+=(const basic_string& str);

template<class T>
  constexpr basic_string& operator+=(const T& t);

constexpr basic_string& operator+=(const charT* s);

constexpr basic_string& operator+=(charT c);

constexpr basic_string& operator+=(initializer_list<charT> il);
```

In [**string.append**]/**20.3.2.6.2**:

```
constexpr basic_string& append(const basic_string& str);

constexpr basic_string& append(const basic_string& str, size_type pos, size_type n = npos);

template<class T>
  constexpr basic_string& append(const T& t);

template<class T>
  constexpr basic_string& append(const T& t, size_type pos, size_type n = npos);

constexpr basic_string& append(const charT* s, size_type n);

constexpr basic_string& append(const charT* s);

constexpr basic_string& append(size_type n, charT c);

template<class InputIterator>
  constexpr basic_string& append(InputIterator first, InputIterator last);

constexpr basic_string& append(initializer_list<charT> il);

constexpr void push_back(charT c);
```

In [**string.assign**]/**20.3.2.6.3**:

```
constexpr basic_string& assign(const basic_string& str);

constexpr basic_string& assign(basic_string&& str)
  noexcept(allocator_traits<Allocator>::propagate_on_container_move_assignment::value ||
           allocator_traits<Allocator>::is_always_equal::value);

constexpr basic_string& assign(const basic_string& str, size_type pos, size_type n = npos);

template<class T>
  constexpr basic_string& assign(const T& t);

template<class T>
  constexpr basic_string& assign(const T& t, size_type pos, size_type n = npos);

constexpr basic_string& assign(const charT* s, size_type n);

constexpr basic_string& assign(const charT* s);

constexpr basic_string& assign(initializer_list<charT> il);

constexpr basic_string& assign(size_type n, charT c);

template<class InputIterator>
  constexpr basic_string& assign(InputIterator first, InputIterator last);
```

In [**string.insert**]/**20.3.2.6.4**:

```
constexpr basic_string& insert(size_type pos, const basic_string& str);

constexpr
```

```
basic_string& insert(size_type pos1, const basic_string& str, size_type pos2, size_type n = npos);

template<class T>
  constexpr basic_string& insert(size_type pos, const T& t);

template<class T>
  constexpr
  basic_string& insert(size_type pos1, const T& t, size_type pos2, size_type n = npos);

constexpr basic_string& insert(size_type pos, const charT* s, size_type n);

constexpr basic_string& insert(size_type pos, const charT* s);

constexpr basic_string& insert(size_type pos, size_type n, charT c);

constexpr iterator insert(const_iterator p, charT c);

constexpr iterator insert(const_iterator p, size_type n, charT c);

template<class InputIterator>
  constexpr iterator insert(const_iterator p, InputIterator first, InputIterator last);

constexpr iterator insert(const_iterator p, initializer_list<charT> il);
```

In [**string.erase**]/**20.3.2.6.5**:

```
constexpr basic_string& erase(size_type pos = 0, size_type n = npos);

constexpr iterator erase(const_iterator p);

constexpr iterator erase(const_iterator first, const_iterator last);

constexpr void pop_back();
```

In [**string.replace**]/**20.3.2.6.6**:

```
constexpr basic_string& replace(size_type pos1, size_type n1, const basic_string& str);

constexpr basic_string& replace(size_type pos1, size_type n1, const basic_string& str,
                                size_type pos2, size_type n2 = npos);

template<class T>
  constexpr basic_string& replace(size_type pos1, size_type n1, const T& t);

template<class T>
  constexpr basic_string& replace(size_type pos1, size_type n1, const T& t,
                                  size_type pos2, size_type n2 = npos);

constexpr basic_string& replace(size_type pos1, size_type n1, const charT* s, size_type n2);

constexpr basic_string& replace(size_type pos, size_type n, const charT* s);

constexpr basic_string& replace(size_type pos1, size_type n1, size_type n2, charT c);

constexpr
basic_string& replace(const_iterator i1, const_iterator i2, const basic_string& str);
```

```
template<class T>
  constexpr
  basic_string& replace(const_iterator i1, const_iterator i2, const T& t);

constexpr
basic_string& replace(const_iterator i1, const_iterator i2, const charT* s, size_type n);

constexpr
basic_string& replace(const_iterator i1, const_iterator i2, const charT* s);

constexpr
basic_string& replace(const_iterator i1, const_iterator i2, size_type n, charT c);

template<class InputIterator>
  constexpr
  basic_string& replace(const_iterator i1, const_iterator i2, InputIterator j1, InputIterator j2);

constexpr
basic_string& replace(const_iterator i1, const_iterator i2, initializer_list<charT> il);
```

In [**string.copy**]/**20.3.2.6.7**:

```
constexpr size_type copy(charT* s, size_type n, size_type pos = 0) const;
```

In [**string.swap**]/**20.3.2.6.8**:

```
constexpr
void swap(basic_string& s)
  noexcept(allocator_traits<Allocator>::propagate_on_container_swap::value ||
           allocator_traits<Allocator>::is_always_equal::value);
```

In [**string.accessors**]/**20.3.2.7.1**:

```
constexpr const charT* c_str() const noexcept;
constexpr const charT* data() const noexcept;

constexpr charT* data() noexcept;

constexpr operator basic_string_view<charT, traits>() const noexcept;

constexpr allocator_type get_allocator() const noexcept;
```

In [**string.find**]/**20.3.2.7.2**:

```
template<class T>
  constexpr size_type find(const T& t, size_type pos = 0) const;

constexpr size_type find(const basic_string& str, size_type pos = 0) const noexcept;

constexpr size_type find(const charT* s, size_type pos, size_type n) const;

constexpr size_type find(const charT* s, size_type pos = 0) const;

constexpr size_type find(charT c, size_type pos = 0) const;
```

In [**string.rfind**] **20.3.2.7.3**:

```
template<class T>
  constexpr size_type rfind(const T& t, size_type pos = npos) const;

constexpr size_type rfind(const basic_string& str, size_type pos = npos) const noexcept;

constexpr size_type rfind(const charT* s, size_type pos, size_type n) const;

constexpr size_type rfind(const charT* s, size_type pos = npos) const;

constexpr size_type rfind(charT c, size_type pos = npos) const;
```

In [**string.find.first.of**]/**20.3.2.7.4**:

```
template<class T>
  constexpr size_type find_first_of(const T& t, size_type pos = 0) const;

constexpr size_type find_first_of(const basic_string& str, size_type pos = 0) const noexcept;

constexpr size_type find_first_of(const charT* s, size_type pos, size_type n) const;

constexpr size_type find_first_of(const charT* s, size_type pos = 0) const;

constexpr size_type find_first_of(charT c, size_type pos = 0) const;
```

In [**string.find.last.of**]/**20.3.2.7.5**:

```
template<class T>
  constexpr size_type find_last_of(const T& t, size_type pos = npos) const;

constexpr size_type find_last_of(const basic_string& str, size_type pos = npos) const noexcept;

constexpr size_type find_last_of(const charT* s, size_type pos, size_type n) const;

constexpr size_type find_last_of(const charT* s, size_type pos = npos) const;

constexpr size_type find_last_of(charT c, size_type pos = npos) const;
```

In [**string.find.first.not.of**]/**20.3.2.7.6**:

```
template<class T>
  constexpr size_type find_first_not_of(const T& t, size_type pos = 0) const;

constexpr size_type find_first_not_of(const basic_string& str, size_type pos = 0) const noexcept;

constexpr size_type find_first_not_of(const charT* s, size_type pos, size_type n) const;

constexpr size_type find_first_not_of(const charT* s, size_type pos = 0) const;

constexpr size_type find_first_not_of(charT c, size_type pos = 0) const;
```

In [**string.find.last.not.of**]/**20.3.2.7.7**:

```
template<class T>
  constexpr
  size_type find_last_not_of(const T& t, size_type pos = npos) const;

constexpr
```

```
size_type find_last_not_of(const basic_string& str, size_type pos = npos) const noexcept;

constexpr
size_type find_last_not_of(const charT* s, size_type pos, size_type n) const;

constexpr
size_type find_last_not_of(const charT* s, size_type pos = npos) const;

constexpr
size_type find_last_not_of(charT c, size_type pos = npos) const;
```

In [**string.substr**]/**20.3.2.7.8**:

```
constexpr basic_string substr(size_type pos = 0, size_type n = npos) const;
```

In [**string.compare**]/**20.3.2.7.9**:

```
template<class T>
  constexpr
  int compare(const T& t) const;

template<class T>
  constexpr
  int compare(size_type pos1, size_type n1, const T& t) const;

template<class T>
  constexpr
  int compare(size_type pos1, size_type n1, const T& t, size_type pos2, size_type n2 = npos) const

constexpr
int compare(const basic_string& str) const noexcept;

constexpr
int compare(size_type pos1, size_type n1, const basic_string& str) const;

constexpr
int compare(size_type pos1, size_type n1, const basic_string& str,
            size_type pos2, size_type n2 = npos) const;

constexpr
int compare(const charT* s) const;

constexpr
int compare(size_type pos, size_type n1, const charT* s) const;

constexpr
int compare(size_type pos, size_type n1, const charT* s, size_type n2) const;
```

In [**string.starts.with**]/**20.3.2.7.10**:

```
constexpr bool starts_with(basic_string_view<charT, traits> x) const noexcept;
constexpr bool starts_with(charT x) const noexcept;
constexpr bool starts_with(const charT* x) const;
```

In [**string.ends.with**]/**20.3.2.7.11**:

```
constexpr bool ends_with(basic_string_view<charT, traits> x) const noexcept;
constexpr bool ends_with(charT x) const noexcept;
constexpr bool ends_with(const charT* x) const;
```

In [**string.nonmembers**]/**20.3.3**:

```
template<class charT, class traits, class Allocator>
  constexpr
  basic_string<charT, traits, Allocator>
    operator+(const basic_string<charT, traits, Allocator>& lhs,
              const basic_string<charT, traits, Allocator>& rhs);

template<class charT, class traits, class Allocator>
  constexpr
  basic_string<charT, traits, Allocator>
    operator+(basic_string<charT, traits, Allocator>&& lhs,
              const basic_string<charT, traits, Allocator>& rhs);

template<class charT, class traits, class Allocator>
  constexpr
  basic_string<charT, traits, Allocator>
    operator+(const basic_string<charT, traits, Allocator>& lhs,
              basic_string<charT, traits, Allocator>&& rhs);

template<class charT, class traits, class Allocator>
  constexpr
  basic_string<charT, traits, Allocator>
    operator+(basic_string<charT, traits, Allocator>&& lhs,
              basic_string<charT, traits, Allocator>&& rhs);

template<class charT, class traits, class Allocator>
  constexpr
  basic_string<charT, traits, Allocator>
    operator+(const charT* lhs, const basic_string<charT, traits, Allocator>& rhs);

template<class charT, class traits, class Allocator>
  constexpr
  basic_string<charT, traits, Allocator>
    operator+(const charT* lhs, basic_string<charT, traits, Allocator>&& rhs);

template<class charT, class traits, class Allocator>
  constexpr
  basic_string<charT, traits, Allocator>
    operator+(charT lhs, const basic_string<charT, traits, Allocator>& rhs);

template<class charT, class traits, class Allocator>
  constexpr
  basic_string<charT, traits, Allocator>
    operator+(charT lhs, basic_string<charT, traits, Allocator>&& rhs);

template<class charT, class traits, class Allocator>
  constexpr
  basic_string<charT, traits, Allocator>
    operator+(const basic_string<charT, traits, Allocator>& lhs, const charT* rhs);
```

```
template<class charT, class traits, class Allocator>
  constexpr
  basic_string<charT, traits, Allocator>
    operator+(basic_string<charT, traits, Allocator>&& lhs, const charT* rhs);

template<class charT, class traits, class Allocator>
  constexpr
  basic_string<charT, traits, Allocator>
    operator+(const basic_string<charT, traits, Allocator>& lhs, charT rhs);

template<class charT, class traits, class Allocator>
  constexpr
  basic_string<charT, traits, Allocator>
    operator+(basic_string<charT, traits, Allocator>&& lhs, charT rhs);

template<class charT, class traits, class Allocator>
  constexpr
  bool operator==(const basic_string<charT, traits, Allocator>& lhs,
                  const basic_string<charT, traits, Allocator>& rhs) noexcept;

template<class charT, class traits, class Allocator>
  constexpr
  bool operator==(const charT* lhs, const basic_string<charT, traits, Allocator>& rhs);

template<class charT, class traits, class Allocator>
  constexpr
  bool operator==(const basic_string<charT, traits, Allocator>& lhs, const charT* rhs);

template<class charT, class traits, class Allocator>
  constexpr
  bool operator!=(const basic_string<charT, traits, Allocator>& lhs,
                  const basic_string<charT, traits, Allocator>& rhs) noexcept;

template<class charT, class traits, class Allocator>
  constexpr
  bool operator!=(const charT* lhs, const basic_string<charT, traits, Allocator>& rhs);

template<class charT, class traits, class Allocator>
  constexpr
  bool operator!=(const basic_string<charT, traits, Allocator>& lhs, const charT* rhs);

template<class charT, class traits, class Allocator>
  constexpr
  bool operator<(const basic_string<charT, traits, Allocator>& lhs,
                 const basic_string<charT, traits, Allocator>& rhs) noexcept;

template<class charT, class traits, class Allocator>
  constexpr
  bool operator<(const charT* lhs, const basic_string<charT, traits, Allocator>& rhs);

template<class charT, class traits, class Allocator>
  constexpr
  bool operator<(const basic_string<charT, traits, Allocator>& lhs, const charT* rhs);
```

```
template<class charT, class traits, class Allocator>
  constexpr
  bool operator>(const basic_string<charT, traits, Allocator>& lhs,
                 const basic_string<charT, traits, Allocator>& rhs) noexcept;

template<class charT, class traits, class Allocator>
  constexpr
  bool operator>(const charT* lhs, const basic_string<charT, traits, Allocator>& rhs);

template<class charT, class traits, class Allocator>
  constexpr
  bool operator>(const basic_string<charT, traits, Allocator>& lhs, const charT* rhs);

template<class charT, class traits, class Allocator>
  constexpr
  bool operator<=(const basic_string<charT, traits, Allocator>& lhs,
                  const basic_string<charT, traits, Allocator>& rhs) noexcept;

template<class charT, class traits, class Allocator>
  constexpr
  bool operator<=(const charT* lhs, const basic_string<charT, traits, Allocator>& rhs);

template<class charT, class traits, class Allocator>
  constexpr
  bool operator<=(const basic_string<charT, traits, Allocator>& lhs, const charT* rhs);

template<class charT, class traits, class Allocator>
  constexpr
  bool operator>=(const basic_string<charT, traits, Allocator>& lhs,
                  const basic_string<charT, traits, Allocator>& rhs) noexcept;

template<class charT, class traits, class Allocator>
  constexpr
  bool operator>=(const charT* lhs, const basic_string<charT, traits, Allocator>& rhs);

template<class charT, class traits, class Allocator>
  constexpr
  bool operator>=(const basic_string<charT, traits, Allocator>& lhs, const charT* rhs);

template<class charT, class traits, class Allocator>
  constexpr
  void swap(basic_string<charT, traits, Allocator>& lhs,
            basic_string<charT, traits, Allocator>& rhs)
    noexcept(noexcept(lhs.swap(rhs)));
```

5 References

[N4727] Richard Smith, *Working Draft, Standard for Programming Language C++* http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2018/n4727.pdf

[P0784R1] Multiple authors, *Standard containers and constexpr*
http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2018/p0784r1.html