Other Operations

A string_view borrows a limited number of methods from the string type. Check them out below.

WE'LL COVER THE FOLLOWING

`string_view` is modelled to be very similar to `std::string`. The view, however, is non-owning, so any operation that modifies the data cannot go into the API. Here's a brief list of methods that you can use with this new type:

Iterators

Method	Description
`cbegin()`, `begin()`	Return an iterator to the first character
`crbegin()`, `rbegin()`	Return a reverse iterator to the first character of the reversed view. It corresponds to the last character of the sequence.
`cend()`, `end()`	Returns an iterator to a place after the last character of a sequence
`crend()`, `rend()`	Returns an iterator to the end of

	reversed sequence. It corresponds to a place before the first character

> *Note:* all of the above methods are `constexpr` and `const`, so you always get a const iterator (even for `begin()` or `end()`).

Accessing Elements

Method	Description
`operator[]`	Returns a const reference to the character at the specified position. Bounds are not checked.
`at()`	Returns a const reference to the character at specified position with bound checking (might throw `std::out_of_range`)
`front()`	Returns a const reference to the first character of the sequence
`back()`	Returns a const reference to the last character of the sequence
`data()`	Returns a pointer to the underlying data

> *Note:* If the view is empty then you'll get undefined behaviour for `operator[]`, `front()`, `back()` and `data()`.

Size & Capacity

Method	Description
`size()` / `length()`	Returns the numbers of characters in a sequence
`max_size()`	The largest possible number of char-like objects that can be referred to by a `basic_string_view`.
`empty()`	Returns `size == 0`

Modifiers

Method	Description
`remove_prefix(size_type n)`	Equivalent to: `data_ += n`; `size_ -= n;`
`remove_suffix(size_type n)`	Equivalent to: `size_ -= n;`
`swap(basic_string_view& s)`	Exchanges the values of `*this` and `s`

Other

Method	Description
`copy(charT* s, size_type n, size_type pos)`	Copies `n` characters into `s` starting from `pos`. not `constexpr`
`substr(size_type pos, size_type n)`	Complexity `O(1)` and not `O(n)` as in `std::string`

`compare(...)` [^metelipsis]	Compares strings, similarly to `std::basic_string::compare`
`find(...)`	Returns position of the first occurence of the input string or `basic_string_view::npos`
`rfind(...)`	Returns position of the last occurence of the input string or `basic_string_view::npos`
`find_first_of(...)`	Returns position of the first character that is equal to any character from the input pattern or `basic_string_view::npos`
`find_last_of(...)`	Returns position of the last character that is equal to any character from the input pattern or `basic_string_view::npos`
`find_first_not_of(...)`	Returns position of the first character that is different to any character from the input pattern or `basic_string_view::npos`
`find_last_not_of(...)`	Returns position of the last character that is different to any character from the input pattern or `basic_string_view::npos`

> [^metelipsis]: ellipsis `(...)` means that a method has several overloads.

Non-member functions:

Function	Description
comparison operators: `==`, `!=`, `<=`, `>=`, `<`, `>`	Lexicographically compares two string views
`operator <<`	For `ostream` output

Key things about the above operations: Key things about the above methods, functions and types:

- All of the above methods (except for `copy`, `operator <<` and `std::hash` specialisation) are also `constexpr`! With this capability, you might now work with contiguous character sequences in constant expressions.
- The above list is almost the same as all non-mutable string operations. However, there are two new methods: `remove_prefix` and `remove_suffix` - they are not `const`, and they modify the `string_view` object. Note that they still cannot modify the referenced data.
- `operator[]`, `at`, `front`, `back`, `data` - are also `const` - thus you cannot change the underlying character sequence (it's only "read access"). In `std::string` there are overloads for those methods that return a reference, so you get "write access". That's not possible with `string_view`.
- `string_view` also has specialisation for `std::hash`
- `string_view` has a string literal `""sv`, and you can define a variable like `auto sv = "hello"sv;`

> *More in C++20!* In C++20 we'll get two new methods:
>
> - `starts_with()`
> - `ends_with()`
>
> They are implemented both for `std::basic_string_view` and `std::basic_string`. As of August 2019 Clang 6.0, GCC 9.0 and VS 2019 16.2 support them.

`string_view` offers a significant amount of functionality, but there are precautions which should be kept in mind while working with this type. The next lesson will help us understand this better.