Introduction

The string view class builds itself on the string class. It restricts the operations that can be performed on a string.

A string view is a non-owning reference to a string. It represents a view of a sequence of characters. This sequence of characters can be a C++ string or a C-string. A string view needs the header `<string_view>`.

> 🔑 **A string view is a for copying optimized string**
> From a birds-eye perspective the purpose of `std::string_view` is to avoid copying data which is already owned by someone else and to allow immutable access to a `std::string` like object. The string view is a kind of a restricted string that supports only the immutable operations. Additionally, a string view `sv` has two additional mutating operations: `sv.remove_prefix` and `sv.remove_suffix`.

String views are class templates parameterized by their character and their character trait. The character trait has a default. In contrast to a string, a string view is non-owner and, therefore, needs no allocator.

```
template < class CharT, class Traits = std::char_traits<CharT> >
  class basic_string_view;
```

According to strings, there exist for string views four synonyms for the underlying character types `char`, `wchar_t`, `char16_t` and `char32_t`.

```
typedef std::string_view std::basic_string_view<char>
typedef std::wstring_view std::basic_string_view<wchar_t>
typedef std::u16string_view std::basic_string_view<char16_t>
typedef std::u32string_view std::basic_string_view<char32_t>
```

> **i std::string_view is the string view**
> If we speak in C++ about a string view, we refer with 99% probability to

> the specialization `std::basic_string_view` for the character type `char`. This statement is also true for this book.