Risks Using string_view

Let's look at some cases where string_view may cause problems.

WE'LL COVER THE FOLLOWING

`std::string_view` was added into the Standard mostly to allow performance optimizations. Nevertheless, it's not a replacement for strings! That's why when you use views you have to remember about a few potentially risky things:

Taking Care of Not Null-Terminated Strings

`string_view` may not contain `\0` at the end of the string. So you have to be prepared for that.

- `string_view` is problematic with all functions that accept traditional C-strings because `string_view` breaks with C-string termination assumptions. If a function accepts only a `const char*` parameter, it's probably a bad idea to pass `string_view` into it. On the other hand, it might be safe when such a function accepts `const char*` and **length** parameters.
- Conversion into strings - you need to specify not only the pointer to the contiguous character sequence but also the length.

References and Temporary Objects

`string_view` doesn't own the memory, so you have to be very careful when working with temporary objects.

In general, the lifetime of a `string_view` must never exceed the lifetime of the string-owning object.

That might be important when:

- Returning `string_view` from a function - the view has to point to that data that is still alive after the function has completed.
- Storing `string_view` in objects or containers - this is similar to storing pointers in a container. The referenced data must be still present when you access elements of this container.

To explore all these issues, let's start with the initial example from this chapter.