Introduction

In this chapter, we'll explore one of the most prominent classes in C++: Strings.
Let's begin!

A string is a sequence of characters. C++ has many methods to analyze or to change a string. C++strings are the safe replacement for C Strings: `const char*`. Strings need the header `<string>`.

> **i A string is very similar to a std::vector**
>
> A string feels like a `std::vector` containing characters. It supports a very similar interface. This means that in addition to the methods of the string class, you have the algorithms of the Standard Template Library to work with the string.
>
> The following code snippet has the `std::string name` with the value `RainerGrimm`. I use the STL algorithm std::find_if to get the upper letter and then extract my first and last name into the variables `firstName` and `lastName`. The expression `name.begin()+1` shows, that strings support random access iterators:

```
//string versus vector
...
#include <algorithm>
#include <string>

std::string name{"RainerGrimm"};
auto strIt= std::find_if(name.begin()+1, name.end(),
                         [](char c){ return std::isupper(c); });
```

```
if (strIt != name.end()){
  firstName= std::string(name.begin(), strIt);

  lastName= std::string(strIt, name.end());
}
```

Strings are class templates parametrized by their character, their character trait and their allocator. The character trait and the allocator have defaults.

```
template <typename charT, typename traits= char_traits<charT>, typename Allocator= allocator<
class basic_string;
```

C++ has synonyms for the character types `char`, `wchar_t`, `char16_t` and `char32_t`

```
typedef basic_string<char> string;
typedef basic_string<wchar_t> wstring;
typedef basic_string<char16_t> u16string;
typedef basic_string<char32_t> u32string;
```

> **i std::string is the string**
> If we speak in C++ about a string, we refer with 99 % probability to the specialisation `std::basic_string` for the character type `char`. This statement is also true for this book.