Type Traits

In this lesson, we will learn about type traits, their advantages, and their application.

Template metaprogramming

Template metaprogramming is programming at compile-time. However, what does template metaprogramming have in common with the type-traits library? As it turns out, quite a lot! The type-traits library is pure template metaprogramming enclosed in a library.

Applications of template metaprogramming

- Programming at compile-time.
- Programming with types such as `double` or `int` and not with values such as 5.5 or 5.
- The compiler instantiates the templates and generates C++ code.

Type traits: goals

The type traits library requires the following header:

```
#include <type_traits>
```

Looking carefully, we can see that type traits have a big optimization potential. In the first step, the type traits analyze the code at compile-time, and in the second step, they optimize the code based on that analysis. How is that possible? Depending on the type of the variable, a faster variant of an algorithm will be chosen.

Optimization

- An optimized version of `std::copy`, `std::fill`, or `std::equal` is used so the algorithms can work on memory blocks.

Correctness

- Type checks will be performed at compile-time.
- Together with `static_assert`, the type information defines the requirements for the code.

Check types

The type trait library supports primary and composite type categories. We get the answer with the attribute `::value`.

In the next lesson, we'll discuss the categories and transformations of type traits.