Type Traits

Enables compile-time type checks, comparisons, and transformations.

➡ Type traits have no influence on the runtime of the program.

- Requires the header `<type_traits>`.

- Application of template metaprogramming
  - Programming at compile time
  - Programming based on types and not on values
  - The compiler generates different code depending on the used types

Type Traits

- Type checks
  - Primary type categories(`::value`)

    `std::is_pointer<T>, std::is_integral<T>, std::is_floating_point<T>`
  - Composite type categories (`::value`)

    `std::is_arithmetic<T>, std::is_object<T>`

- Type comparisons (`::value`)

  `std::is_same<T, U>, std::is_base_of<Base, Derived>,`
  `std::is_convertible<From, To>`

- Type transformations (`::type`)

  `std::add_const<T>, std::remove_reference<T>, std::make_signed<T>,`
  `std::add_pointer<T>`

`typeTraitsCategories.cpp`

Type Traits: Objectives

- Optimization
  - Code that improves itself during compilation

    ➡ Depending on the type a special algorithm will be chosen.
  - Optimized version of `std::copy`, `std::fill`, or `std::equal`

    ➡ Algorithm can be directly applied on raw memory.

- Correctness
  - Type information will be evaluated at compile time.
  - The evaluated type informations combined with `static_assert`, provides the necessary conditions for correctness.