1 Type Traits

Classes

Helper Classes

`integral_constant` (C++11) `bool_constant` (C++17)	compile-time constant of specified type with specified value (class template)
`true_type`	`std::integral_constant<bool, true>`
`false_type`	`std::integral_constant<bool, false>`

Primary type categories

`is_void` (C++11)	checks if a type is `void` (class template)
`is_null_pointer` (C++14)	checks if a type is `std::nullptr_t` (class template)
`is_integral` (C++11)	checks if a type is an integral type (class template)
`is_floating_point` (C++11)	checks if a type is a floating-point type (class template)
`is_array` (C++11)	checks if a type is an array type (class template)
`is_enum` (C++11)	checks if a type is an enumeration type (class template)
`is_union` (C++11)	checks if a type is a union type (class template)
`is_class` (C++11)	checks if a type is a non-union class type (class template)
`is_function` (C++11)	checks if a type is a function type (class template)
`is_pointer` (C++11)	checks if a type is a pointer type (class template)
`is_lvalue_reference` (C++11)	checks if a type is an *lvalue reference* (class template)
`is_rvalue_reference` (C++11)	checks if a type is an *rvalue reference* (class template)
`is_member_object_pointer` (C++11)	checks if a type is a pointer to a non-static member object (class template)
`is_member_function_pointer` (C++11)	checks if a type is a pointer to a non-static member function (class template)

Composite type categories

`is_fundamental` (C++11)	checks if a type is a fundamental type (class template)
`is_arithmetic` (C++11)	checks if a type is an arithmetic type (class template)
`is_scalar` (C++11)	checks if a type is a scalar type (class template)
`is_object` (C++11)	checks if a type is an object type (class template)
`is_compound` (C++11)	checks if a type is a compound type (class template)
`is_reference` (C++11)	checks if a type is either an *lvalue reference* or *rvalue reference* (class template)
`is_member_pointer` (C++11)	checks if a type is a pointer to a non-static member function or object (class template)

Type properties

`is_const` (C++11)	checks if a type is const-qualified (class template)
`is_volatile` (C++11)	checks if a type is volatile-qualified (class template)
`is_trivial` (C++11)	checks if a type is trivial (class template)
`is_trivially_copyable` (C++11)	checks if a type is trivially copyable (class template)
`is_standard_layout` (C++11)	checks if a type is a standard-layout type (class template)
`is_pod` (C++11)(deprecated in C++20)	checks if a type is a plain-old data (POD) type (class template)
(C++11) `is_literal_type` (deprecated in C++17) (removed in C++20)	checks if a type is a literal type (class template)
`has_unique_object_representations` (C++17)	checks if every bit in the type's object representation contributes to its value (class template)
`is_empty` (C++11)	checks if a type is a class (but not union) type and has no non-static data members (class template)
`is_polymorphic` (C++11)	checks if a type is a polymorphic class type (class template)
`is_abstract` (C++11)	checks if a type is an abstract class type (class template)
`is_final` (C++14)	checks if a type is a final class type (class template)
`is_aggregate` (C++17)	checks if a type is an aggregate type (class template)
`is_implicit_lifetime` (C++23)	checks if a type is an implicit-lifetime type (class template)
`is_signed` (C++11)	checks if a type is a signed arithmetic type (class template)
`is_unsigned` (C++11)	checks if a type is an unsigned arithmetic type (class template)
`is_bounded_array` (C++20)	checks if a type is an array type of known bound (class template)
`is_unbounded_array` (C++20)	checks if a type is an array type of unknown bound (class template)
`is_scoped_enum` (C++23)	checks if a type is a scoped enumeration type (class template)

Supported operations

`is_constructible` (C++11) `is_trivially_constructible` (C++11) `is_nothrow_constructible` (C++11)	checks if a type has a constructor for specific arguments (class template)
`is_default_constructible` (C++11) `is_trivially_default_constructible` (C++11) `is_nothrow_default_constructible` (C++11)	checks if a type has a default constructor (class template)
`is_copy_constructible` (C++11) `is_trivially_copy_constructible` (C++11) `is_nothrow_copy_constructible` (C++11)	checks if a type has a copy constructor (class template)
`is_move_constructible` (C++11) `is_trivially_move_constructible` (C++11) `is_nothrow_move_constructible` (C++11)	checks if a type can be constructed from an rvalue reference (class template)

`is_assignable` (C++11) `is_trivially_assignable` (C++11) `is_nothrow_assignable` (C++11)	checks if a type has an assignment operator for a specific argument (class template)
`is_copy_assignable` (C++11) `is_trivially_copy_assignable` (C++11) `is_nothrow_copy_assignable` (C++11)	checks if a type has a copy assignment operator (class template)
`is_move_assignable` (C++11) `is_trivially_move_assignable` (C++11) `is_nothrow_move_assignable` (C++11)	checks if a type has a move assignment operator (class template)
`is_destructible` (C++11) `is_trivially_destructible` (C++11) `is_nothrow_destructible` (C++11)	checks if a type has a non-deleted destructor (class template)
`has_virtual_destructor` (C++11)	checks if a type has a virtual destructor (class template)
`is_swappable_with` (C++17) `is_swappable` (C++17) `is_nothrow_swappable_with` (C++17) `is_nothrow_swappable` (C++17)	checks if objects of a type can be swapped with objects of same or different type (class template)

Property queries

`alignment_of` (C++11)	obtains the type's alignment requirements (class template)
`rank` (C++11)	obtains the number of dimensions of an array type (class template)
`extent` (C++11)	obtains the size of an array type along a specified dimension (class template)

Type relationships

`is_same` (C++11)	checks if two types are the same (class template)
`is_base_of` (C++11)	checks if a type is derived from the other type (class template)
`is_convertible` (C++11) `is_nothrow_convertible` (C++20)	checks if a type can be converted to the other type (class template)
`is_layout_compatible` (C++20)	checks if two types are *layout-compatible* (class template)
`is_pointer_interconvertible_base_of` (C++20)	checks if a type is a *pointer-interconvertible* (initial) base of another type (class template)

`is_invocable`
`is_invocable_r`
`is_nothrow_invocable` (C++17)
`is_nothrow_invocable_r`
	checks if a type can be invoked (as if by `std::invoke`) with the given argument types
	(class template)

Const-volatility specifiers

`remove_cv` (C++11) `remove_const` (C++11) `remove_volatile` (C++11)	removes `const` and/or `volatile` specifiers from the given type (class template)
`add_cv` (C++11) `add_const` (C++11) `add_volatile` (C++11)	adds `const` and/or `volatile` specifiers to the given type (class template)

References

`remove_reference` (C++11)	removes a reference from the given type (class template)
`add_lvalue_reference` (C++11) `add_rvalue_reference` (C++11)	adds an *lvalue* or *rvalue* reference to the given type (class template)

Pointers

`remove_pointer` (C++11)	removes a pointer from the given type (class template)
`add_pointer` (C++11)	adds a pointer to the given type (class template)

Sign modifiers

`make_signed` (C++11)	makes the given integral type signed (class template)
`make_unsigned` (C++11)	makes the given integral type unsigned (class template)

Arrays

`remove_extent` (C++11)	removes one extent from the given array type (class template)
`remove_all_extents` (C++11)	removes all extents from the given array type (class template)

Miscellaneous transformations

`aligned_storage` (C++11)(deprecated in C++23)	defines the type suitable for use as uninitialized storage for types of given size (class template)
`aligned_union` (C++11)(deprecated in C++23)	defines the type suitable for use as uninitialized storage for all given types (class template)
`decay` (C++11)	applies type transformations as when passing a function argument by value (class template)
`remove_cvref` (C++20)	combines `std::remove_cv` and `std::remove_reference` (class template)
`enable_if` (C++11)	conditionally removes a function overload or template specialization from overload resolution (class template)
`conditional` (C++11)	chooses one type or another based on compile-time boolean (class template)
`common_type` (C++11)	determines the common type of a group of types (class template)
`common_reference` `basic_common_reference` (C++20)	determines the common reference type of a group of types (class template)
`underlying_type` (C++11)	obtains the underlying integer type for a given enumeration type (class template)
`result_of` (C++11)(removed in C++20) `invoke_result` (C++17)	deduces the result type of invoking a callable object with a set of arguments (class template)
`void_t` (C++17)	void variadic alias template (alias template)
`type_identity` (C++20)	returns the type argument unchanged (class template)

Operations on traits

`conjunction` (C++17)	variadic logical AND metafunction (class template)
`disjunction` (C++17)	variadic logical OR metafunction (class template)
`negation` (C++17)	logical NOT metafunction (class template)

Functions

Member relationships

`is_pointer_interconvertible_with_class` (C++20)	checks if objects of a type are pointer-interconvertible with the specified subobject of that type (function template)
`is_corresponding_member` (C++20)	checks if two specified members correspond to each other in the common initial subsequence of two specified types (function template)

Constant evaluation context

`is_constant_evaluated` (C++20)	detects whether the call occurs within a constant-evaluated context (function)