Glossary

inheriting constructors (see Section 2.1."Inheriting ~~Ctors~~" on page 535) to avoid having to rewrite the derived-class constructors explicitly. For integer types, a classical **enum** having the original type as its underlying type (see Section 2.1."Underlying Type '11" on page 829) can serve the same purpose. Function **static** '11 (74)

structural inheritance – a form of inheritance in which non-**virtual** functions are inherited by a derived class; see also implementation inheritance. Deleted Functions (57), **alignas** (180), Inheriting Ctors (545), **final** (1025)

structural type – a category of type, introduced in C++20, used to characterize the extended set of allowable types for values supplied as non-type template parameters, which include integral type, enumeration type, pointer type, pointer-to-member type, lvalue reference type, as well as floating-point type and class types made up of other structural types.

structured binding – a C++17 feature that introduces new names bound to the subobjects of the object produced by the expression that initializes them, i.e., `auto [a,b] = std::make_tuple(17,42);` where `a` will be initialized to `17` and `b` will hold the value `42`. Range **for** (685)

substitution failure is not an error (SFINAE) – a property of C++ template instantiation that enables the technique of supplying template arguments to a function template to determine if the resulting function will participate in overload resolution or supplying template arguments to a template partial specialization to determine whether it is eligible to be instantiated; errors that result from substituting certain (e.g., syntactically incompatible) arguments for the template parameters result merely in that particular template instantiation being dropped from the overload set, and is not (in and of itself) ill formed.

sum type – an *abstract data type* allowing the representation of one of multiple possible alternative types. Each alternative has its own type (and state), and only one alternative can be active at any given point in time. Sum types keep track of which choice is active and properly implement (value-semantic) special member functions (even for non-trivial types). They can be implemented efficiently as a C++ **class** using a **union** and a separate (integral) discriminator. This sort of implementation is commonly referred to as a discriminating (or tagged) union and is available in C++17 as `std::variant`. **union** '11 (1177)

synchronization paradigm – the general approach used to coordinate memory reads and writes among two or more concurrent threads of execution. As used in this book, one of two approaches within the *release-acquire/consume* memory consistency model — release-acquire or release-consume. `carries_dependency` (998)

syntactic sugar – language and library features that ease the use of the language by providing more ergonomic interfaces that obviate more verbose or difficult-to-use syntax but do not themselves provide new functionality.

template argument – the type, template, or value mapped to a template parameter in the instantiation of a template. Variadic Templates (899)

template argument deduction – the process by which template arguments are determined from the types of the *function* arguments when calling a function template. Variadic Templates (894)

template argument list – the sequence of arguments — types, templates, or values, depending on the corresponding parameters — that are used to specify explicit, nondeduced arguments to a template instantiation. Variadic Templates (882)

template head – the keyword **template** and associated template parameter list used to introduce the declaration or definition of a template. Variable Templates (157)