Glossary

defaulted special member function – a (user-declared) special member function specified (see Section 1.1."Defaulted Functions" on page 33) to be implemented using its default (i.e., compiler-generated) definition. Note that, in some situations (e.g., see Section 3.1."**union** '11" on page 1174), the compiler may choose to delete an implicitly or explicitly declared and defaulted function.

defect report (DR) – an acknowledgment by the C++ Standards Committee of a defect in the current C++ Standard that is considered by ISO to apply to the currently active C++ Standard and is generally taken by implementers to apply to all previous versions of the C++ Standard where the change would be both applicable and practicable. Braced Init (218), **constexpr** Functions (280), Generalized PODs '11 (432), Inheriting Ctors (551), `initializer_list` (561), Lambdas (594), **noexcept** Operator (615), Range **for** (681), *Rvalue* References (722), **noexcept** Specifier (1086)

defensive check – one typically performed at runtime (e.g., using a C-style `assert` macro) to verify some condition that is impossible to occur in a correct program. A common use case is to verify, for a given function, that there has not been a contract violation — i.e., a precondition or postcondition violation — yet is entirely superfluous in a correctly implemented program. Generalized PODs '11 (468), *Rvalue* References (744)

defensive programming – a term, sometimes (mis)used, to suggest generally good programming practice implies the use of defensive checks to, for example, detect client misuse of a given function, by violating its preconditions when invoking it. **final** (1024)

define – to provide, for a given entity, any additional details, e.g., size, layout, address, etc., beyond just its declaration, needed to use that entity in a running process. Deleted Functions (58), Forwarding References (390), *Rvalue* References (762), Variadic Templates (880)

defined behavior – (1) behavior that is unambiguously codified in terms of C++'s abstract machine or (2) the full set of behaviors defined for a given component or library. Note that invoking a component or library out of contract is library undefined behavior (a.k.a. soft UB), which might lead to language undefined behavior (a.k.a. hard UB). **noexcept** Specifier (1112)

defining declaration – one — such as **class** `Foo { };` — that provides a complete definition of the entity being declared. Note that a **typedef** or **using** declaration (see Section 1.1."**using** Aliases" on page 133) would not be considered *defining* because, according to the C++ Standard, neither is a definition. Also note that an opaque enumeration declaration does not provide the enumerators corresponding to the complete definition and, although sufficient to instantiate opaque objects of the enumerated type, does not allow for interpretation of their values; hence, it too would not be considered *defining*; see also nondefining declaration. *Rvalue* References (729)

definition – a statement that fully characterizes an entity (e.g., type, object, or function); note that all definitions are subject to the one-definition rule. Function **static** '11 (68), **constexpr** Variables (315), Variadic Templates (879), **noexcept** Specifier (1105)

delegating constructor – one that, rather than fully initializing data members and base-class objects itself, invokes another constructor after which it might perform additional work in its own body (see Section 1.1."Delegating Ctors" on page 46). Delegating Ctors (46)

deleted – implies (1) for a given function, that it has been rendered inaccessible from *any* access level — either explicitly, by being annotated using `=` **delete** (see Section 1.1."Deleted Functions" on page 53) or implicitly (e.g., see Section 3.1."**union** '11" on page 1174); or (2) for a given pointer to a dynamically allocated object, that the (typically global) **delete** operator