Glossary

nondefining declaration – one — such as `class Foo;` — that does not provide all of the collateral information, such as function or class body, associated with a complete definition. Note that a `typedef` or `using` declaration (see Section 1.1."`using` Aliases" on page 133) is *non*defining as type aliases are declared, not defined. Also note that an opaque enumeration declaration provides only the underlying type for that enumeration, sufficient to instantiate opaque objects of the enumerated type yet *not* sufficient to interpret its values; hence, it too is not (fully) defining and therefore is *nondefining*. Note that a nondefining declaration may be repeated within a single translation unit (TU); see also defining declaration. *Rvalue* References (729)

nonprimitive functionality – implementable in terms of the publicly accessible functionality of a type or a set of types and, hence, does not require access to any of their encapsulated (private) implementation details. `explicit` Operators (67)

nonreporting contract – a function contract that does *not* specify an out clause — i.e., the contract (e.g., for `std::vector::push_back`) makes no mention of what happens if the operation were to fail. `noexcept` Specifier (1120)

nonreporting function – a function whose contract offers no mechanism to report whether the principal action requested was performed. `noexcept` Specifier (1119)

nonstatic data member – one declared without the `static` keyword, an instance of which appears in every object of the class. `constexpr` Variables (305)

non-trivial constructor – not a trivial constructor, such as a user-provided constructor (which is never trivial, even if the resulting generated code matches exactly what a trivial constructor would do). `union` '11 (1174)

non-trivial destructor – not a trivial destructor — e.g., a user-provided destructor or one whose implementation invokes some non-trivial destructor of a base-class or data-member subobject. `noexcept` Specifier (1118)

non-trivial special member function – one that is not trivial; see also special member function. `union` '11 (1174)

non-type parameter – short for non-type template parameter. Variadic Templates (902)

non-type parameter pack – a template parameter pack made up of non-type template parameters. Variadic Templates (902)

non-type template parameter – one whose argument is a constant expression, rather than a type or template; the parameter must have integral type, enumeration type, pointer type, pointer-to-member type, or lvalue reference type. C++20 broadens slightly this category of types to structural types, which includes floating-point types and class types comprising other structural types. Variadic Templates (902)

normative wording – implies, for wording in the Standard, that it forms part of the definition of ISO C++, as opposed to (non-normative) notes, which exist only to make the Standard easier to read. *Rvalue* References (808)

notionally trivially destructible – implies, for a given class type, that it could (and would) have a trivial destructor if not for a desire to have some entirely optional operations that do not change the semantics of a correct program — e.g., defensive checks to verify that object invariants have been maintained properly throughout the lifetime of the object. Note that such a type might be written using, e.g., conditional compilation to have a trivial destructor in some build modes but not in others. Generalized PODs '11 (468)