Glossary

horizontal encoding – a convention whereby the meaning of certain bits that occur throughout an encoding (e.g., the microcode of a computer) are independent of the values of bits that occur elsewhere in that encoding.

hot path – the part of a program (specifically, generated object code) that is executed under normal and frequently encountered conditions; see also cold path. **`noexcept`** Specifier (1103)

Hyrum's law – the observation attributed to Hyrum Wright of Google that, given a sufficient number of users of an API, all observable behavior — notably including those that are undocumented, unintentional, nonessential, or unstable — will be depended upon by the user base. **`final`** (1012), **`friend`** '11 (1036)

id expression – a qualified id or unqualified id that can be used to name an entity or a set of entities, such as variable names, function names, and (after a `.` or `->`) class member names. **`decltype`** (25), *Rvalue* References (780)

identity – a property of an expression that can be identified uniquely, e.g., by name or address, independently of its value; see also has identity.

IFNDR – short for ill formed, no diagnostic required.

ill formed – implies, for a given program, that it is not valid C++. A compiler is required to fail to compile such a program and issue an appropriate diagnostic (error) message unless the ill~~-~~formed nature is explicitly identified as one where no diagnostic is required (a.k.a. IFNDR); see ill formed, no diagnostic required. **`static_assert`** (120), Braced Init (227), **`constexpr`** Variables (303), User-Defined Literals (839), **`inline namespace`** (1067), **`auto`** Return (1203)

ill formed, no diagnostic required (IFNDR) – implies, for a given program, that it is ill formed in a way where the compiler is not required to issue a diagnostic. Typical examples of IFNDR, such as violations of the ODR, do not require a diagnostic because identifying the problem would either drastically impact compile times or be otherwise impracticable (if not impossible) in general. Delegating ~~Ctors~~ (50), **`static_assert`** (117), **`alignas`** (177), **`constexpr`** Functions (262), **`enum class`** (350), Opaque **`enum`**s (666), Underlying Type '11 (832), User-Defined Literals (840), Variadic Templates (900), `carries_dependency` (1000), **`inline namespace`** (1067)

immutable type – a user-defined type for which objects instantiated from that type, once fully constructed, cannot be changed. Ref-Qualifiers (1167)

imperative programming – implies, for a given language or programming paradigm, the use of a sequence of statements describing the evaluations of expressions that progressively *mutate* existing state (e.g., variables, objects) within a program instead of always creating new objects of immutable types as is common in *declarative* or *functional* programming. **`constexpr`** Functions '14 (959)

implementation defined – implies, for a given behavior, that it is not fully specified by the Standard but that an implementation must specify in its documentation. Attribute Syntax (12), **`nullptr`** (100), **`alignas`** (168), **`constexpr`** Functions (295), **`enum class`** (335), Generalized PODs '11 (501), Opaque **`enum`**s (660), *Rvalue* References (747), **`noexcept`** Specifier (1093)

implementation inheritance – a form of inheritance in which the implementation of a non**`virtual`** or nonpure **`virtual`** function defined in a base class is inherited along with its interface in a derived class; note that inheriting the definitions of non**`virtual`** functions is sometimes referred to more specifically as structural inheritance; see also interface inheritance. Inheriting Ctors (541)