Glossary

EBO – see empty-base optimization.

embedded development – writing, documenting, testing, and deploying software for embedded systems. Binary Literals (145)

embedded system – one that runs either on resource-limited hardware or in restricted environments, ranging from pacemakers to set-top entertainment devices. **`long long`** (93), **`noexcept`** Specifier (1101)

emplacement – an often more efficient alternative to copy construction in which the arguments to some value constructor of an object, rather than a reference to a constructed object itself, are used to construct a new object directly in its final destination — e.g., `template<typename T> push_back(const T&);` versus `template<typename... Args> void emplace_back(Args&&... args);` for the `std::vector` container; see, e.g., **hu20**. Forwarding References (390)

empty-base optimization (EBO) – a compiler optimization in which a base-class subobject that introduces no non**`static`** data members is assigned the same address as another subobject of the derived-class object, provided they do not have the same type, to avoid any size overhead that would otherwise be required. Since C++11, compilers are required to perform this optimization if the derived class is a standard-layout class; otherwise, this optimization is allowed but not required. Had the same empty base type been used instead to create a data member, at least one additional byte would have been required within the footprint of the outer class; hence, the preference for making empty types base classes rather than data members. Note that C++20 introduces an attribute to address the inefficiency of empty data members. **`alignof`** (185), Generalized PODs '11 (499), Lambdas (607), Variadic Templates (933), **`final`** (1028)

encapsulation – the colocation of (typically private) data along with manipulator and accessory functions used to act upon and retrieve that data; ideally the representation of the data can change, perhaps necessitating client code be recompiled, but without forcing any clients to rework their code; see also insulation. Opaque **`enum`**s (663)

encoding prefix – one placed before a string or character literal used to indicate a literal having a character type other than **`char`**. C++03 supported `L` for **`wchar_t`**; C++11 added `u` for **`char16_t`**, `U` for **`char32_t`**, and `u8` for **`char`** (with UTF-8 encoding). User-Defined Literals (844)

entity – one of the primary logical building blocks of a C++ program: value, *object*, reference, *function*, *enumerator*, *type*, class member, bit field, *template*, *template* specialization, *namespace*, parameter pack, or **`this`**. **`decltype`** (25), Local Types '11 (84), `deprecated` (147)

equality comparable – implies, for a given type, that the homogeneous equality-comparison operators, **`operator`**`==` and **`operator`**`!=`, are defined and publicly accessible for the purpose of determining whether two objects of that type have (represent) the same value; see value semantics. Note that equality comparable is independent of homogeneous relational operators (<, <=, >, >=).

escalation – a form of refactoring (a.k.a. *escalation technique*) whereby parts of a pair of components that are mutually dependent are moved to a separate, higher-level component, enabling the removal of a potential cyclic physical dependency; see **lakos20**, section 3.5.2, "Escalation," pp. 604–614. **`extern template`** (374)

essential behavior – a superset of postconditions that includes aspects of the computation beyond the final result, such as runtime complexity, thread safety, exception safety, etc.