Glossary

unique object address – implies, for a given object of a given type, that no other object of that type resides at that same address at the same time. In general, two non-bit-field objects having overlapping lifetimes must have distinct addresses unless one is nested within the other (e.g., a base class subobject and the enclosing derived class object, or an object and its first non**static** data member) or they are of different types and at least one is an empty base class (e.g., a base class and a non**static** data member with a different type at offset 0). As of C++20, the requirement for an object to have a distinct address may be relaxed under certain circumstances (e.g., for an empty member object of class type) through use of the `[[no_unique_address]]` attribute. Generalized PODs '11 (418)

unique ownership – a resource-management model in which at most one object can claim ownership of a resource at any given time. The move operations for a type implementing this model (a.k.a. a move-only type) will typically transfer ownership of any allocated resource to the moved-to object, leaving the moved-from object resourceless. Destroying the current owner releases the resource entirely — e.g., `std::unique_ptr`. *Rvalue* References (768)

unit test – a (sometimes standalone) test intended to verify the correctness of the implementation of a single software component along with any of its inherent physical dependencies.

universal reference – a synonym for forwarding reference proposed by Scott Meyers, favored by some, and discouraged by the C++ Standards Committee. Forwarding References (400)

unnamed namespace – one introduced without a name (a.k.a. an *anonymous namespace*). Any entity that is declared within an unnamed namespace is unique to the translation unit in which it is defined, has internal linkage (which, for an object, is comparable to declaring it **static** at file scope), and can be used as if it were declared in the enclosing namespace without additional qualification (see Section 3.1."**inline namespace**" on page 1055). Function **static** '11 (77)

unqualified id – an identifier (e.g., `x`), operator name (e.g., **operator**=), or template id (e.g., `T<A,C::B>`) that is not preceded by a scope-resolution operator (`::`) or class member access operator (`.` or `->`).

unqualified name lookup – the process by which an unqualified ID is matched to an entity by searching through enclosing class and namespace scopes, as well as associated namespaces nominated by argument-dependent lookup (ADL). User-Defined Literals (841)

unrelated types – types that are either (1) entirely unrelated by inheritance or (2) do not share a common polymorphic class as a base class (note that pointers and references to unrelated types are not interconvertible using **dynamic_cast**). Generalized PODs '11 (507)

unsigned ordinary character type – either **unsigned char** or, on platforms where **char** is *unsigned*, **char**. Generalized PODs '11 (515)

usable – implies, for a given member function, that it is accessible, defined, and, in the context in which it is called, *unambiguous*, i.e., overload resolution will identify it as the *best viable function*.

usable literal type – one that provides a nonempty set of operations beyond merely those required of it to be a literal type, enabling meaningful use in a constant expression. **constexpr** Functions (282)

user declared – implies, for a given function, that its declaration appears in the source code irrespective of whether it is deleted or defaulted; see Section 1.1."Deleted Functions" on page 53 and Section 1.1."Defaulted Functions" on page 33, respectively. **constexpr** Functions (274), Generalized PODs '11 (413), **noexcept** Specifier (1086)