integral constant expression, and integral type. Deleted Functions (56), **enum class** (334), *Rvalue* References (726), Underlying Type ’11 (832)

integral type – ~~a category of fundamental types, codified by the `std::is_integral` trait, denoting one of **bool**, **char**, **signed char**, **unsigned char**, **char16_t**, **char32_t**, **wchar_t**, and the familiar **signed** and **unsigned** variations on **short**, **int**, **long**, **long long** (see Section 1.1.“**long long**” on page 89), and any implementation defined extended-integer type; C++20 adds **char8_t** to this list.~~ **long long** (89), Underlying Type ’11 (829)

interface inheritance – a form of inheritance in which the interface (only) of one or more pure **virtual** functions declared in a base class is inherited in a derived class; see also implementation inheritance. Inheriting Ctors (541)

interface trait – a (typically standard) trait, such as `std::is_trivially_destructible`, that describes an aspect of the usable interface of a type but does *not* correspond to a property named in the core language specification; see also core trait. Generalized PODs ’11 (482)

internal linkage – linkage that prevents an entity from being referenced by name from another translation unit. Multiple distinct entities having internal linkage may have the same name, provided each resides in a separate translation unit; see also external linkage. Function **static** ’11 (77), **constexpr** Variables (307)

intra-thread dependency – a data dependency that exists between evaluations within a single thread. `carries_dependency` (998)

invocable – implies, for a given entity `f` and zero or more arguments `t1`, `t2`, ..., `tN`, that one of `(t1.*f)(t2, ..., tN)`, `((*t1).*f)(t2, ..., tN)`, `t1.*f`, `(*t1).*f`, or `f(t1, t2, ..., tN)` is well formed at the point of invocation — i.e., `f` must be usable and either a (1) callable entity, (2) pointer-to-member function, or (3) pointer-to-data member. Generalized PODs ’11 (482), Lambda Captures (986)

invocable entity – one that is invocable; see also callable entity.

join (a thread) – the operation by which execution of the current thread is suspended until execution of one or more other threads completes.

lambda body – the statements in a lambda expression that will form the body of a lambda closure’s call operator. Lambdas (581), *Generic* Lambdas (976)

lambda capture – a syntax by which variables from a reaching scope are made available for use within the body of a lambda expression. See also captured by copy and captured by reference. Lambdas (577), Variadic Templates (919), *Generic* Lambdas (969)

lambda closure – the object created by evaluating a lambda expression. Lambdas (584)

lambda declarator – the function parameter list, mutability, exception specification, and return type of a lambda expression, all of which are imbued on the call operator of the lambda closure. Lambdas (591)

lambda expression – an anonymous *callable* type having unnamed data members used to store values that are, by default, captured by copy (`=`) or else captured by reference (`&`); see Section 2.1.“Lambdas” on page 573. Local Types ’11 (83), Lambdas (576), *Generic* Lambdas (968), Lambda Captures (995), **auto** Return (1182), **decltype(auto)** (1206)

lambda introducer – a possibly empty lambda capture list, surrounded by `[]`, used to begin a lambda expression; e.g., `[](){}` is a lambda expression that captures nothing, takes no arguments, does nothing, and returns **void**. Lambdas (582), Lambda Captures (986)