Glossary

in which those instructions will execute — e.g., vectorized operations or operations that will exhibit superior pipelining on the target CPU. `noexcept` Specifier (1136)

insulate – reduce or eliminate compile-time coupling (e.g., an implementation detail); see insulation. `noreturn` (96), `constexpr` Functions (299), `extern template` (369), Opaque `enum`s (665)

insulating – implies, for a given interface, that implementation details are insulated and can change without forcing clients to recompile, only relink.

insulation – a strong form of encapsulation in which the representation of private implementation details can change without forcing clients to recompile their code, but simply to relink it; see **lakos20**, section 3.11.1, "Formal Definitions of *Encapsulation* vs. *Insulation*," pp. 790–791. Opaque `enum`s (663)

integer literal – one specifying an integral value. This value can be expressed (1) in decimal, in which case the literal sequence of (decimal) digits is in the range `[0-9]` and does not begin with `0`; (2) in octal, in which case the (octal) digit sequence is limited to the range `[0-7]` and begins with `0`; (3) in hexadecimal, in which case the literal begins with a `0x` or `0X` and is followed by one or more (hexadecimal) digits in the case-insensitive range `[0-9a-f]`; or, as of C++14, (4) in binary, in which case the literal begins with a `0b` or `0B` and is followed by one or more (binary) digits, `0` or `1` (see Section 1.2."Binary Literals" on page 142). An integer literal may have an optional suffix, which may contain a (case-insensitive) `L` or `LL` and may also contain a (case-insensitive) `U`, e.g., `1L`, `0377uL`, or `0xABCuL`. As of C++14, digit separators (`'`) (see Section 1.2."Digit Separators" on page 152) may be used to visually group digits (e.g., `1'000'000`) and are especially useful in C++14 with binary literals (see Section 1.2. "Binary Literals" on page 142), e.g., `0b1100'1011`. Note that every integer literal has a non-negative value; expressions such as `-1`, `-02`, and `-0x3` apply the unary-minus (`-`) operator to the non-negative **`int`** value of that integer literal. Note that if the value is too large to fit in **`int`**, then **`unsigned int`**, **`long`**, **`unsigned long`**, **`long long`**, and **`unsigned long long`** may be tried (in that order); however, the set of possible types for an integer literal may be constrained by its suffix. In particular, a decimal literal without a suffix always has a signed type and overflows if the value cannot be represented as a **`signed long long`**. More generally, if the value of the integer literal is not representable in any type that is compatible with the prefix and suffix, it is ill formed. User-Defined Literals (837)

integer-to-floating-point conversion – an implicit conversion from an integral type to a floating-point type. User-Defined Literals (843)

integer type – a synonym for integral type.

integral constant – a constant expression of integral type such as an integer literal or **`constexpr`** variable of integral type (see Section 2.1."**`constexpr`** Variables" on page 302). Note that a **`const`**-qualified variable of integral type that is initialized with a constant expression can be an integral constant too. Braced Init (223), `constexpr` Variables (303)

integral constant expression – an expression of integral or unscoped enumeration type that is a constant expression, i.e., one that can be evaluated at compile time. `alignas` (169), `alignof` (184)

integral promotion – the implicit conversion of the type of an integral expression to a larger integral type that preserves value. In expressions, an integral bit field of less than the number of bits in an **`int`** is, by default, promoted to an **`int`**. In *integral* expressions, if a (binary) operation is applied to two integral expressions of different sizes, the smaller one will be promoted to the larger size before the operation is performed; see also integral constant,