Glossary

exception agnostic – a particular style of implementation in which no exception-related syntax (**`try`**, **`catch`**, or **`throw`**) is used and hence no exceptions are thrown directly; nonetheless the code remains entirely exception safe with respect to injected exceptions (e.g., from derived classes or template arguments) through judicious use of RAII. Importantly, a fully exception agnostic library will be buildable and behave correctly, irrespective of whether support for exceptions in the compiler is enabled; note that exception and nonexception builds might not necessarily be ABI compatible. **`noexcept`** Operator (644), **`noexcept`** Specifier (1126)

exception-free path – one in which the flow of control (e.g., leading up to an invocation of a function) does not involve the throwing (and potentially catching) of any exceptions. **`noexcept`** Specifier (1134)

exception safe – implies, for a given function, that no defects will manifest nor resources leak when an exception escapes the function, even when that exception is thrown by a different function. More generally, a class, template, component, or library is exception safe if no defects or resource leaks occur as the result of an exception thrown during the evaluation of any function they define. **`noexcept`** Operator (644), **`noexcept`** Specifier (1126)

exception specification – one of two forms — dynamic exception specification (deprecated) or **`noexcept`** (as of C++11) — used for a given function to indicate which or just whether, respectively, exceptions may be thrown by that function. Lambdas (593), *Rvalue* References (733)

excess N notation – a representation of a (typically *signed*) value v stored instead as $v + N$ in an *unsigned* integer or bit field; the range for v is $-N$ through $M - N$, where M is the largest value representable in that storage. IEEE floating-point formats make use of a form of this notation in which N (a.k.a. the *bias*) is one less than half the range of the unsigned storage: excess 127 [-126 to +127] for **`float`** and excess 1023 [-511 to +512] for **`double`**. Note, however, that the smallest and largest values are reserved and have special meaning, thereby reducing the range of representable exponents by 2. Digit Separators (155)

executable image – the representation of the program in relocatable binary format, typically stored in a file, that is (at least partially) loaded by the operating system into memory prior to execution of that program. **`noexcept`** Specifier (1135)

execution character set – that comprising all characters that can be interpreted by a running program on a specific platform including, at minimum, all of the characters in the source character set; control characters representing alert, backspace, and carriage return; and a null character (or null wide character), whose value is 0. Each character in the execution character set has an implementation-defined non-negative integral value, typically specified by a standard such as ASCII or Unicode. User-Defined Literals (844)

expiring object – implies, for a given object, that it no longer needs to hold either its value or the resources used to represent that value beyond the expression in which it is used. An object can be explicitly marked as expiring (e.g., using `std::move`), and expressions that designate objects so marked are xvalues. An object may also be implicitly deemed by a compiler to be expiring (e.g., for a temporary). *Rvalue* References (713)

explicit instantiation declaration – a directive (see Section 2.1."**`extern template`**" on page 353) that suppresses implicit instantiation of a function template for the specified template arguments in the local translation unit, instead relying on an explicit instantiation definition, provided in exactly one translation unit within the program, potentially reducing compilation time and object-code size (but having no effect on the linked program's executable size or run time); see also explicit instantiation definition. **`extern template`** (353)