explicit instantiation definition – a directive (see Section 2.1."**extern template**" on page 353), for a given template and specific template arguments, to instantiate and emit, in the current translation unit, any associated object code for entities that the template defines. Note that at most one such directive per template specialization may appear in a program, *no diagnostic required*; see also explicit instantiation declaration and Ill Formed, No Diagnostic Required. **extern template** (353)

explicit instantiation directive – either an explicit instantiation declaration or an explicit instantiation definition. **extern template** (353)

explicit specialization – a declaration or complete definition of a template specialization, used instead of instantiating the corresponding *primary template* (or any partial specialization) that might be selected when supplied with those same arguments at the point of use; see primary class template declaration and partial ordering of class template specialization.

explicit template-argument specification – the specification, when invoking a function template — e.g., `template <typename T, typename U> func()` — with a sequence of template arguments (e.g., `int`, `double`) surrounded by `<` and `>`, e.g., `func<int, double>(0, 0)`; such explicitly specified template arguments will be used as is and will not require template argument deduction. Variadic Templates (895)

explicitly captured – implies, for a given variable, that it is named in the capture list of a lambda. Lambdas (582)

expression – a valid sequence of operators and operands that specifies a computation; the evaluation of an expression may produce a value or cause side effects. Unlike a statement, expressions may be nested; see also outermost expression.

expression alias – an often considered, potential future feature of C++ that would support a parameterized alias for an expression. Such a feature would substitute the expression in-place, much like a hygienic macro, behaving like a forced inline function having automatically deduced result type and exception specification (see Section 3.1."**noexcept** Specifier" on page 1085), but without the possibility of separating declaration and definition. **noexcept** Specifier (1146)

expression SFINAE – the use of SFINAE to exclude a function template specialization from consideration during overload resolution or a (class) template partial specialization during template instantiation, based on the validity of a particular expression. This form of SFINAE enables programming patterns such as the detection idiom. **decltype** (29), **static_assert** (122), Trailing Return (126)

expression template – a template metaprogramming pattern in which overloaded operators return compound types that capture, within their template parameters, an entire expression. When these complex types are converted to a desired result type, an optimized implementation of the entire expression will be evaluated, instead of a potentially much less efficient evaluation of each individual subexpression. This general technique has been used often in libraries such as Eigen (**eigen**) to optimize computations involving large matrices. **auto** Variables (202)

extended alignment – an alignment larger than the alignment of `std::max_align_t`. **alignas** (168)

external linkage – linkage that allows a name to refer to the same entity across translation units; see also internal linkage.

factory function – one whose purpose is to construct, initialize, and return an object, often by value. *Rvalue* References (778), User-Defined Literals (836), Variadic Templates (929)