

Glossary

- overloading** – the act of creating an overload.
- overriding** – providing, for a **virtual** function declared in a base type, a suitable implementation specific to a derived type. [Inheriting Ctors \(539\)](#)
- owned resource** – one, such as dynamic memory, a socket, or a shared-memory handle, that is managed by an object (a.k.a. the *owner*), typically with the expectation that the owner will release the resource when it no longer needs it, e.g., in the owner’s destructor. Move operations typically transfer an **owned resource** from one owner to another. On occasion, a resource can have more than one owner — such as in the case of `std::shared_ptr` — in which case the last owner to be destroyed is typically responsible for releasing the resource. [Rvalue References \(741\)](#)
- pack expansion** – the process of expanding a syntactic pattern followed by an ellipsis (...) that contains the name of at least one **parameter pack** into a comma-separated list of instantiations of that pattern, appropriate for that pack expansion context. [Lambdas \(590\)](#), [Variadic Templates \(882\)](#), [constexpr Functions '14 \(964\)](#)
- pack-expansion context** – the syntactic position at which a **pack expansion** occurs, which impacts both the kinds of **parameter packs** that can be expanded and the semantics of that expansion. [Variadic Templates \(883\)](#)
- package prefix** – an organization-wide unique (ideally very short) identifier that denotes the smallest unit of physical design larger than a component (e.g., `std`, `bs1s`, `bs1ma`, `ball`). This name is often used for the **namespace** containing a collection of related logical entities, such as classes and functions. This same name can be used as the initial part of the name of a physical entity, such as a directory, a component, or a library, to associate it with the **namespace** that comprises its logical content; see [lakos20](#), section 2.4, “Logical and Physical Name Cohesion,” pp. 297–333, specifically section 2.4.10, “Package Prefixes Are Not Just Style,” pp. 322–326.
- padding byte** – one supplied by the compiler in the footprint of an object of class type, typically to satisfy alignment requirements of individual data members or for the object as a whole. Each padding byte is of indeterminate value and, like a **vtable pointer** or **virtual base pointer**, does not contribute to the value representation of the object. [Generalized PODs '11 \(475\)](#)
- parameter** – the (formal) name declared within the defining declaration of an entity, such as a function or template, used locally within its definition to refer to the corresponding (actual) argument supplied when called or instantiated, respectively.
- parameter declaration** – The declaration of a parameter to a function within a function’s parameter list. [Variadic Templates \(888\)](#)
- parameter list** – a sequence of formal parameter declarations, such as a function parameter list or a template parameter list.
- parameter pack** – a function parameter pack or a template parameter pack. [Variable Templates \(159\)](#), [Variadic Templates \(873\)](#), [constexpr Functions '14 \(964\)](#)
- partial application** – transforming an N-ary function into another of smaller arity, specifically by binding one or more **parameters** of the original function to fixed **arguments**, as commonly occurs in C++ when trailing **parameters** of a given function are declared to have default arguments or, more generally, when a higher-order function, e.g., $h(F, v)$, is applied to some function, e.g., $f(x, y, z)$, to yield another function, e.g., $g(x, y) = h(f(x, y, z), k)$, such that, for all a and b , $g(a, b) = f(a, b, k)$. [Lambdas \(597\)](#)
- partial class-template specialization** – the partial specialization of a class template. [constexpr Functions '14 \(963\)](#)