made trivial by an implementation. The default constructors of both `std::pair` and `std::tuple` require that they value-initialize their elements, resulting in operations that can *never* be trivial.

4. **Conditionally defaultable operations** — The remaining operations are defined in such a way that they might be eligible to use `= default` for some potential template arguments. A noteworthy example is the assignment operators of `std::pair` and `std::tuple`, which are required to do a memberwise assignment, even when the member type is a reference. The defaulted assignment operation would be deleted in this case, so for at least some template arguments these operations will not be trivial. It would be possible to conditionally default these operations for nonreference types, but that involves partial specialization and complicated inheritance schemes or the need for an explosion of mildly varying implementations. This significant cost leads most library vendors to not attempt to make these operations trivial, and their being trivial is something that must be left as QoI and not a trait that can be portably depended upon.[64]

The Standard does not describe the layout of its classes, nor does it list the private members that may be used to implement them. In principle, the implementation of `std::pair` could have additional private members or declare `first` and `second` in different public base classes,[65] resulting in a conforming implementation that is never a standard-layout type. Similarly, `std::tuple` is often implemented through inheritance (recursively or through **pack expansion**) of a distinct type for each member element, resulting in a type that cannot be a standard-layout type for anything with more than one element. An implementation *could* provide distinct specializations for standard-layout fixed numbers of elements, but having multiple such specializations would be a labor-intensive solution to achieve standard layout for a subset of potential template arguments and is a QoI choice that standard library vendors do not seem to have made.

See Also

- "Aggregate Init '14" (§1.2, p. 138) introduces the notion of default member initialization to aggregates.
- "Braced Init" (§2.1, p. 215) provides additional insight into aggregates as well as other forms of braced initialization.
- "**`constexpr`** Functions" (§2.1, p. 257) shows how trivial types can be made usable at compile time.

[64] Modern versions of GCC, Clang, and MSVC always implement `std::pair`'s copy and move-assignment operators as user-provided functions.

[65] An example of such an implementation can be found in the BDE open-source library implementation of `pair` maintained by the authors of this book. This implementation partially specializes `pair` for template arguments of reference type such that instantiations are trivially copyable if and only if both template arguments are of trivially copyable nonreference type; see **bde14**, /groups/bsl/bslstl/bslstl_pair.h.