According to the C++ Standard, the value `std::is_standard_layout<T>::value` is intended to be **`true`** for any `T` that is a standard-layout type. There are, however, certain seldom encountered cases where the incorrect value is produced by this trait. In particular, some popular compilers[12] produce the wrong value for the trait (**`true`**) when the compiler is forced to pad an object with empty space to prevent overlapping footprints for an otherwise empty base class and the first nonstatic data member.

The value `std::is_trivial<T>::value` is supposed to be **`true`** for any `T` that is a trivial type. This trait too is often flawed for some edge cases. In particular, on the same contemporary versions of popular compilers,[13] `std::is_trivial` too yields a false positive (evaluates to **`true`**) for a class having a deleted default constructor, deleted copy and move operations, or deleted destructor (e.g., `S4d`, `S4j`, `S4k`, `S5d`, `S5i`, `S5j`, `S7b`, `S7h`, `S7j`, `S7k`, and `S7l` in the examples above). Using these same compilers, `std::is_trivial` yields a false negative (evaluates to **`false`**) when a class deletes a copy or move operation that would invoke a non-**trivial operation** for a base class or a data member (e.g., `S7j`, `S7k`, and `S7l` in the examples above).

For querying if an arbitrary constructor is trivial, the Standard Library provides another type trait, `is_trivially_constructible`, which identifies if construction with a given sequence of arguments and subsequent destruction is well defined and trivial for a given type `T`:

```
namespace std {

template <typename T, typename... Args>
struct is_trivially_constructible
{
    // The value is true if T t(std::declval<Args>()...) invokes no non-trivial
    // operations.
};

}
```

Because it is defined in terms of a variable declaration, this trait tests the triviality of both construction and destruction. The construction combines a **template parameter pack** with the Standard Library utility `std::declval` to specify a variable declaration initialized by a sequence of arguments of the specified types; see Section 2.1."Variadic Templates" on page 873. Three more specific type traits are provided to identify the various types of constructors that could potentially be trivial; note that nothing other than a default constructor, copy constructor, or move constructor can be trivial:

```
namespace std {

template <typename T>
struct is_trivially_default_constructible
    : std::is_trivially_constructible<T> { };           // possible definition
```

[12] E.g., GCC 11.1 (c. 2021) and Clang 12.0 (c. 2021).
[13] E.g., GCC 11.1 (c. 2021) and Clang 12.0 (c. 2021).