```
struct D : B { };  // D is non-trivial even though B is.

static_assert(std::is_trivial<D>::value, "");                          // Error
static_assert(std::is_trivially_default_constructible<D>::value, "");  // Error
static_assert(std::is_trivially_copyable<D>::value, "");               // OK

D d;  // Error, default constructor of D is implicitly deleted.
```

Because `D` has a deleted default constructor, an up-to-date conforming C++11/14 (or later) implementation will report `D` (above) as being trivially copyable, but *not* of trivial type, yet older compilers might wrongly allow `D` to pass for a trivial type. Several workarounds exist. A library implementer could, for example, employ the `std::is_trivially_default_constructible` trait to ensure that the default constructor is in fact invocable (as well as being unambiguous and accessible) with respect to the type expression on which the trait is applied. Note that `std::is_trivially_default_constructible` does not distinguish between a type that cannot be default constructed at all (i.e., `std::is_default_constructible` evaluates to **`false`**) and a type whose default construction involves *non*-trivial functions.

Similarly, the definition of standard-layout type has matured since it was made distinct from POD `type` in C++11. After C++14 was released, the Standard clarified the requirement that there be at most one class in the derivation tree of a standard-layout type that "has" one or more *non*`static` data members and extended the definition of a standard-layout type to include *unnamed* bit fields as well.[56]

The type of any base class of a standard-layout type cannot be the same as any nonstatic data member that would be at offset zero within objects of that type; otherwise, uniqueness of object address would be violated. C++17 provides a more rigorous, recursive definition of the set of types of all *non*-base-class subobjects that must be at offset zero, and requires that there be no overlap between this set and any direct or indirect base classes of a type to be considered a standard-layout class.[57] Subsequent Standards also make explicit that a standard-layout class has at most one base class subobject of any given type.[58] These later definitions also clarified what the first nonstatic data member means. Despite all of these clarifications being defect reports against C++14 and, in practice, against C++11, the `std::is_standard_layout` trait might not accurately represent the up-to-date definition of standard-layout type; again, see *Relevant standard type traits are unreliable* on page 527.

Finally, in C++03, allowing the flow of control to bypass (e.g., via a **`goto`**) the declaration of an **automatic variable** required that it be of POD type needing no initialization. As of C++11, the constraints on the type of such a variable were relaxed to no longer require that it be of either standard-layout type or trivial type, so long as the class had both a trivial

[56] See CWG issue 1881; **ranns14**.
[57] See CWG issue 1672 (**smith13**) and CWG issue 2120 (**tong15**).
[58] See CWG issue 1813; **vandevoorde13**.