3. The type has no nonstatic data members employing a **default member initializer** (see Section 2.1."Default Member Init" on page 318):

```
// Type                                    Is trivial?
struct S2a { int i; };                   // yes
struct S2b { int i = 0; };               // no, default member initializer
struct S2c { int i = {}; };              // no, default member initializer
struct S2d { static const int i = 0; };  // yes, static const data member
struct S2e { static const int i{}; };    // yes,    "     "     "     "
```

4. The type has no user-provided default constructors:

```
struct S3a { int i; };         // yes
struct S3b { int i; S3b(); };  // no, user-provided default constructor
```

5. The type has at least one nondeleted **trivial default constructor**. A trivial constructor is one that is not user-provided and invokes a trivial default constructor for each base class and nonstatic data member. Additionally, the presence of **virtual** functions, **virtual** base classes, or default member initializers (items 1–3 above) prevents the default constructor from being trivial, with the **virtual** entities necessitating the constructor to properly initialize the vtable pointer or the virtual base pointer:

```
// Type                              Is trivial?
struct S4a { };                    // yes
struct S4b { S4b(); };             // no, user-provided default constructor
struct S4c { S4c() = default; };   // yes, defaulted default constructor
struct S4d { S4d() = delete; };    // no, deleted default constructor
struct S4e { S4e() = default; S4e(int = 0) = delete; };  // yes, but ambiguous
struct S4f { S4f() = delete; S4f(int = 0) = default; };  // Error, bad syntax

// Type                     Is trivial?
struct S4g : S4a { };    // Yes, S4a base class has trivial default constructor.
struct S4h : S4b { };    // No, S4b base class has non-trivial default ctor.
struct S4i { S4c c; };   // Yes, S4c member has trivial default constructor.
struct S4j { S4d d; };   // No, S4d missing nondeleted default ctor.
struct S4k : S4e { };    // No, S4e default constructor is ambiguous, so S4k
                         //      default constructor is implicitly deleted.
```

Note that S4e above is trivial, but the default constructor is ambiguous and therefore cannot be used. This unusable default constructor prevents S4k from being trivial because the compiler cannot synthesize a default constructor for S4k. Also note that S4f(int = 0) cannot be defaulted and is thus ill formed; see Section 1.1."Defaulted Functions" on page 33.

6. The class has a trivial destructor — i.e., a destructor that is not user-provided, is not **virtual**, and for which each base class and nonstatic member destructor is trivial.