The defaulted destructor must be nondeleted, which requires that each base class and non**static** member destructor also be nondeleted and accessible:

```
// Type                                               Is trivial?
struct S5a { S5a() = default; };                   // yes
struct S5b { S5b() = default; ~S5b(); };           // no, user-provided dtor
struct S5c { S5c() = default; ~S5c() = default; }; // yes
struct S5d { S5d() = default; ~S5d() = delete; };  // no, deleted dtor
struct S5e { private: ~S5e() = default; };         // Yes, but dtor is private.

// Type                       Is trivial?
struct S5f : S5a { };      // Yes, S5a base class has trivial destructor.
struct S5g : S5b { };      // No, S5b base class has non-trivial destructor.
struct S5h { S5c c; };     // Yes, S5c member has trivial destructor.
struct S5i { S5d d[5]; };  // No, S5d has a deleted destructor.
struct S5j : S5e { };      // No, S5e base class destructor is not accessible.
```

Note that `S5e` above is trivial, but the destructor is private and cannot be used except by **`friend`**s. The destructor for `S5j` is deleted because it cannot access the destructor for base class `S5e`, making `S5j` non-trivial.

7. The class has no user-provided copy constructors, **move constructors**, copy-assignment operators, or **move-assignment operators**:

```
// Type                                       Is trivial?
struct S6a { };                            // yes
struct S6b { S6b() = default;
             S6b(const S6b&) = default; }; // yes
struct S6c { S6c() = default;
             S6c(const S6c&); };           // no, has user-provided copy ctor
struct S6d { S6d() = default;
             S6d(const S6d&) = delete; };  // yes, no user-provided copy ctor
struct S6e { S6e& operator=(S6e&&); };     // no, user-provided move assignment
```

8. There is at least one nondeleted trivial copy constructor, move constructor, copy-assignment operator, or move-assignment operator. Each of these operations is trivial if it is not user-provided and if it invokes only trivial constructors or **assignment operators** for each base class and non**static** data member. Additionally, the presence of either **`virtual`** functions or **`virtual`** base classes (items ~~0~~ and ~~1~~ above) prevent the copy/move constructors and copy/move-assignment operators from being trivial:

```
// Type                                       Is trivial?
struct S7a
{
    S7a()                         = default;  // trivial default constructor
    S7a(const S7a&)               = delete;   // deleted copy constructor
```