The `[[deprecated]]` attribute can be used portably to decorate other entities: **`class`**, **`struct`**, **`union`**, **`type alias`**, **`variable`**, **`data member`**, function, enumeration, template **specialization**.[2]

A programmer can supply a **`string literal`** as an **`argument`** to the `[[deprecated]]` attribute — e.g., `[[deprecated("message")]]` — to inform human users regarding the reason for the deprecation:

```
[[deprecated("too slow, use algo1 instead")]] void algo0();
                                              void algo1();

void f()
{
    algo0();  // Warning: algo0 is deprecated; too slow, use algo1 instead.
    algo1();
}
```

An entity that is initially **`declared`** without `[[deprecated]]` can later be redeclared with the attribute and vice versa:

```
void f();
void g0() { f(); }  // OK, likely no warnings

[[deprecated]] void f();
void g1() { f(); }  // Warning: f is deprecated.

void f();
void g2() { f(); }  // Warning: f is deprecated still.
```

As shown in `g2` in the example above, redeclaring an entity that was previously decorated with `[[deprecated]]` without the attribute leaves the entity still deprecated.

Use Cases

Discouraging use of an obsolete or unsafe entity

Decorating any entity with the `[[deprecated]]` attribute serves both to indicate a particular feature should not be used in the future and to actively encourage migration of existing uses to a better alternative. Obsolescence, lack of safety, and poor performance are common motivators for deprecation.

As an example of productive deprecation, consider the `RandomGenerator` class having a static `nextRandom` **`member function`** to generate random numbers:

[2] Applying `[[deprecated]]` to a specific enumerator or namespace, however, is guaranteed to be supported only since C++17; see **smith15a**.