thus does not risk giving programmers the false belief that they are modifying the container.

10. Loop with **`const auto`** has the same behavior for both the `IntVec` and `BoolVec` instantiations. That mechanism is the same behavior as for loop with **`auto`** (item 4) except that, because `v` is **`const`**, *neither* instantiation can modify the container.

11. Loop with **`const auto`**`&` also works for both instantiations. For the `IntVec` case, the result of `*__begin` is bound directly to `v`. For the `BoolVec` case, `v` is deduced to be a **`const`** reference to the proxy type; `*__begin` produces a temporary variable of the proxy type, which is then bound to `v`. Lifetime extension keeps the proxy alive. In most contexts, a **`const`** proxy reference is an effective stand-in for a **`const bool`**`&`.

12. Loop with **`const auto`**`&&` fails to compile for `IntVec` but succeeds for `BoolVec`. The error with `IntVec` occurs because **`const auto`**`&&` is always a **`const`** rvalue reference (not a forwarding reference) and cannot be bound to the lvalue reference, `*__begin`. For `BoolVec`, the mechanism is identical to loop with **`const auto`**`&` (item 11) except that loop with **`const auto`**`&` (item 11) binds the temporary object to an lvalue reference, whereas loop with **`const auto`**`&&` (item 12) uses an rvalue reference. When the references are **`const`**, however, there is little practical difference~~s~~ between them.

Note that loop with **`auto`**, loop with **`auto`**`&&`, loop with **`const auto`**, loop with **`const auto`**`&`, and loop with **`const auto`**`&&` (items 4, 6, 10, 11, and 12) in the `BoolVec` instantiations bind a reference to a temporary proxy reference object, so taking the address of `v` in these situations is likely not to produce useful results. Additionally, loop with `T&&`, loop with **`const`** `T&`, and loop with **`const`** `T&&` (items 3, 8, and 9) bind `v` to a temporary **`bool`**. Users must be mindful of the lifetime of these temporary objects (a single iteration of the loop) and not allow the address of `v` to escape the loop.

Proxy objects emulating references to nonclass elements within a container are surprisingly effective, but their limitations are exposed when they are bound to references. In generic code, as a rule of thumb, **`const auto`**`&` is the safest way to declare a read-only loop variable if a reference proxy might be in use, while **`auto`**`&&` will give the most consistent results for a loop that modifies its container. Similar issues, unrelated to range-based **`for`** loops, occur when passing a proxy reference to a function taking a reference argument.

Annoyances

No access to the state of the iteration

When traversing a range with a classic **`for`** loop, the loop variable is typically an iterator or array index. Within the loop, we can modify that variable to repeat or skip iterations. Similarly, the loop-termination condition is usually accessible so that it is possible to, for example, insert or remove elements and then recompute the condition: