Copying an element is not always erroneous, but it might be wise to habitually declare the loop variable as a reference, making deliberate exceptions when needed:

```
void f4(std::vector<std::string>& vec)
{
    for (std::string& s : vec)
    {
        process(s);  // OK, call process on reference to string element
    }
}
```

If we want to avoid copying elements but also want to avoid modifying them, then a **`const`** reference will provide a good balance. Note, however, that if the type being iterated over is not the same as the type of the reference, a conversion might quietly produce the (undesired) copy anyway:

```
void f5(std::vector<char*>& vec)
{
    for (const std::string& s : vec)
    {
        // s is a reference to a copy of an element of vec.
    }
}
```

In this example, the elements of `vec` have type **`char`**`*`. The use of **`const`** `std::string&` to declare the loop variable `s` correctly prevents modification of any elements of `vec`, but there is still a copy being made because each member access is converted to an object of type `std::string`.

Although the copying conversion in the examples above ~~are~~ discoverable with relative ease, iterating over an elaborate container coupled with implicit converting constructors can make subtle inadvertent copies difficult to detect. A classic example is that of iterating over the elements of an `std::map` or `std::unordered_map`. Suppose, for example, we define an IP table that maps 32-bit IPv4 addresses to domain name aliases; note that our use of digit separators (`'`) in the IP addresses is valid only as of C++14, but can be omitted in C++11 without changing the meaning of the program (see Section 1.2."Digit Separators" on page 152):

```
#include <cstdint>        // std::int32_t, std::uint32_t
#include <string>         // std::string
#include <vector>         // std::vector
#include <unordered_map>  // std::unordered_map

using IPTable = std::unordered_map<std::int32_t, std::vector<std::string>>;

IPTable iptable =
{
    { 0x12'dd'c3'31, { "domain.com", "www.domain.com" } },
```