Vectors & Loops

The modern C++ range-based loops work with `vector` as well as with `string`.
This loop automatically visits every element in the `vector`:

```
for (auto e : v) {...}          // e is a copy
for (auto& e : v) {...}         // no copy; may modify
for (const auto& e : v) {...}   // no copy; cannot modify
```

1. The **local variable `e`** is initialized with **a copy** of the next value in `vector` `v`.
2. Here, `e` is a **reference** to the next element in `v`. When you modify `e` you are actually modifying the element inside the `vector` `v`.
3. If `v` is a `vector<string>`, for example, you **don't want to make a copy** of each of the elements. And, if **you want to prevent any changes**, then use this version of the range-based ***`for`*** loop.

Counter-controlled Loops

The **general pattern** for **manually** iterating through a vector looks like this:

```
for (size_t i = 0, len = v.size(); i < len; ++i)
{
    // Process the vector elements here
}
```

Some notes about this loop:

- Instead of calling `v.size()` each time in the loop, call it once and **save the value in a variable**; your loop initializer will thus have **two** variables.
- Use `size_t` to avoid the lengthy declaration of `vector::size_type`.
- **At all costs** avoid comparing an `int` to the value returned from `v.size()`. **Mixing signed and unsigned numbers is error prone**.

Next, let's look at a few common vector algorithms.