Counter-Controlled Loops

Inside a function, most commonly, you'll calculate the size of an array at the point where it is declared, and then **pass that size as an additional argument** when you **call** the function.

For instance, here is a function which sums the elements in a `vector`:

```
int sum(const vector<int>& v)
{
   int sum = 0;
   for (size_t i = 0, len = v.size(); i < len; ++i)
      sum += v.at(i);
   return sum;
}
```

Notice that the function only requires one argument, since the `vector` "carries" its size along with it. With an array, you'd need to write the same function like this:

```
int sum(const int array[], size_t len)
{
   int sum = 0;
   for (size_t i = 0; i < len; ++i)
      sum += array[i];
   return sum;
}
```

Unlike `string` and `vector`, arrays have no built-in `size()` member function. And, because `array` is really a pointer, you can't use the `sizeof` "trick" inside the function. You must pass the length as an argument **when calling** the function. Note, also, that unlike `vector` you have no range-checked `at()` function. You must use the built-in subscript operator.