Some Bad Habits

You may see the idiomatic loop written like this:

```
for (size_t i = 0; i < str.size(); ++i)
    // do something with str[i] or str.at(i)
```

This is a **bad habit** which **assumes** that calling size() is "free"—that is, it executes in constant time and there is no overhead for calling the function. This is close to true for `string::size()`, but **it is not true** for all functions. For instance, when working with C-style strings, using the equivalent `strlen()` function is very expensive. **Don't train your fingers** to do that.

Even worse is combining this bad habit with `int` indexes, like this:

```
for (int i = 0; i < str.size(); ++i)...
```

The comparison `i < str.size()` automatically converts the type of `i` to an `unsigned size_t`. If `i` ever becomes negative, it is compared **as if it were a very large positive number**. Your compiler may warn you if you mix signed and unsigned numbers like this, but it's easier to remember: **Just don't do it!**

Since `size()` never changes in the loop, **store the length in a variable**, and **use the variable** in your test. Here is an example:

```
for (size_t i = 0, len = str.size(); i < len; ++i)...
```

*Should you use `i++` or `++i` in your loop update expression? With `int` or `size_t` indexes, it makes to difference. The effect is the same either way. However, I prefer the prefix version (`++i`) because I want to "train my fingers" for more the more advanced **iterator** loops you'll work with in CS 250. With iterators, often the `i++` version is much slower, or even non-existent.*