


Processing Strings

One use of the *for* loop is to process strings. The *for* loop, and the **asymmetric bounds pattern** are ideal, because the subscripts use by strings and arrays all begin at **0**, and the last element is always found at **size() - 1**.

The **canonical classic** *for* loop to process every character in a string, should look something like this:

```
for (size_t i = 0, len = str.size(); i < len; ++i)
{
    char c = str.at(i);    // now, process c in some way
}
```

Note that the **string::size()** member function returns an **unsigned** type. If you are not careful, that can lead to unexpected results like this:

```
 for (size_t len = str.size() - 1; i >= 0; --i) ...
```

This loop intended to **count down** from the last character to the first at index **0**. However, if **str** is an **empty** string, then **i** starts at the largest possible unsigned number, instead of **-1**, since unsigned numbers "wrap around". Even worse, because **i** is an **unsigned** type, the condition **i >= 0** can **never** be false, so you can **never exit the loop**.

Here is a loop that is written correctly:

```
for (size_t i = str.size(); i > 0; --i)
{
    char c = str.at(i - 1);
}
```

Alternatively, **you can store str.size() in an int variable**, as long as you:

- cast the returned value from **str.size()** to an **int** like this:

```
for (int i = static_cast<int>(str.size()); i >= 0; --i) ...
```
- make sure that the **string** you are processing is no longer than the positive range of an **int**. If you do this, your loop will **not work** if you have a large string.



This course content is offered under a [CC Attribution Non-Commercial](#) license. Content in this course can be considered under this license unless otherwise noted.