String Members

Below are the member functions you should memorize:

String members	
`size`	the number of characters in the `string` (may also use `length`)
`empty`	true if the `string` contains no characters
`at`	an individual character at a particular position (may also use `[]`)
`front`, `back`	the character at the front, and at the back (C++11)
`substr`	a new `string` created from a portion of an existing `string`
`find`, `rfind`	index of the substring searched for (from front or back)

You can look up the rest.

The *size* Member Function

`s.size()` returns the **number of characters** in the `string s`. For historical reasons, you can also use `length()`, but all of the other collections in the library use `size()`, so you should probably get used to using that. (Plus, it's less typing 😄).

The `size()` member function returns an **unsigned integer**, not an `int` as it does in Java, which may be defined differently on different platforms.

- On an embedded platform, with little memory, `size()` could return a 16-bit `unsigned short`.
- More commonly, strings can be as big as 4 billion characters, so an `unsigned int` is often large enough.
- However, you can't assume that is true. I recently recompiled some older code and discovered several places where I had assumed that `size()` returned an `unsigned int`, but the platform I was on used a 64-bit `unsigned long` instead.

This seems complex, since you don't want to re-edit your code each time you move to a new compiler. Here are three different ways to store the value returned from calling `size()` that work regardless of the platform:

```
string str{...};  // string of any size
string::size_type len1 = str.size();
auto len2 = str.size();
size_t len3 = str.size();
```

1. To be slavishly, pedantically correct, use `string::size_type`.
2. Use `auto` which **infers** the type from the initializer. (You must use `=`, not braces.)
3. Use the type `size_t`. This is the `unsigned` machine type, so your code will be adjusted automatically for each platform.

I believe that the easiest method is the last, and that's what I'll do in this class.

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.