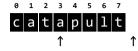
Substrings Redux

Instead of using a loop guard, let's think about the loop in a different way.

You need to loop through **s1**, extracting each substring, and comparing it to **s2**. Rather than writing a *for* loop with **index** refer to the **beginning** of the substring, you can have it point to **the element following the substring**, and then extract the characters **preceding index**.

▶ Watch the video



A picture might help. Suppose s1 is "catapult" and s2 is "tap", here is how that looks. Your loop becomes very easy to write, and you don't need any extra *if* statements:

```
int count{0}; // times s2 found in s1
size_t slen{s2.size()}; // size of string looking for
for (size_t i = slen, len= s1.size(); i <= len; ++i )
{
    string subs = s1.substr(i - slen, slen );
    if (subs == s2) { ++count; }
}
return count;</pre>
```

Things to notice about this loop:

- The loop index (i) starts at slen, where slen is the size of the substring you intend to extract. It does not start at 0.
- When calling substr(index, count), the index is i-slen, which means you are
 extracting the characters before i, not after it.
- Since i points to the first position past your substring, the loop condition is not i <
 len, but i <= len. (Make sure you don't confuse len and slen).

All that's left to do is to compare **subs** to **s2** and update your counter. With C++ strings, you can use **==**; you don't need to use an **equals()** method as you would in Java.



This course content is offered under a <u>CC Attribution Non-Commercial</u> license. Content in