The Strip Filter

Now that you know a little bit about filters, let's use that technique with our homework for this chapter, a comment-stripping filter. Upload the starter code your workspace. Here are the details of the problem.

Write a filter function named `strip` that removes C++ comments from input, sending the uncommented portion of the program to output. Your program should work with both single line (//) comments, and multi-line (/ */) comments.*

- A "/*" sequence is ended by the **first** "*/" that is encountered.
- Comment characters inside string literals "don't count"
- Strings quotes may be escaped using an escape character.
- Single-line comments // may be included inside multi-line comments

Because this is a filter, you program will read from `cin` and write to `cout`.

1. First Draft

That looks pretty straightforward. Let's see what we need to do.

- Include the `<iostream>` header
- Add the basic, unformatted I/O pattern to the function
- Test the function with `make test`

When you test it, you'll see that the function simply echoes every character input to output; this is the same **echo filter** presented in the chapter. To strip out the comments, we'll have to **only** print a character as long as it is **not part of a C++ comment**; in other words, we'll have to **filter out the comment parts**!

2. Second Draft

We have a set of rules that determine when we can print a character, and when we ignore it. At any point, the character we have just read **is in a particular state**:

- It is in a `string` (or not).
- It is in a single-line comment (or not).
- It is inside a multi-line comment (or not).

Now you need some rules based on those three states. The basic rule is this:

```
If you are not inside a comment then
   echo the character
```

Your job is to determine if you are in a comment or not. Here's the pseudocode for this first part of the problem

```
Let "flags" inSingleCmt, inMultiCmt, inString be false
Read character ch from input until end-of-input
    If ch is not inside a comment then
        Echo ch to output
```

Go ahead and translate that to C++. When you test it, you'll see that all of the comments are still printed! To turn off printing, you'll have to monitor the state of the stream, and then toggle the three flags back and forth as the state changes.

If you get stuck, ask on Piazza or come to my office hours.

3. Looking for Comments

Here's a plan for determining this.

1. If you read the `/*` sequence, and you are not inside a `string`, and, you are not already inside another comment, then **start a multiline comment**.
2. If you read the `//` sequence, and you are not inside a `string` or another comment, then **start a single-line comment**.
3. If you read a quote (`"`) and are not inside a `string` or comment, and, if, the previous character read was not an escape character, then **start a string**.
4. If you are inside a `string`, reading another quote (`"`), not preceded by the escape character, ends that `string`.
5. Inside a single-line comment, **reading a newline ends the comment**.

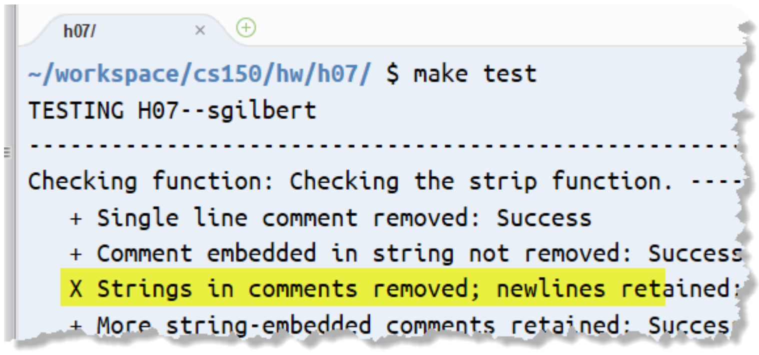
6. Inside a multiline comment, reading a `*/` ends the comment.

Use the Boolean flag variables to keep track of the state and then use those to determine whether the character you've read should be printed.

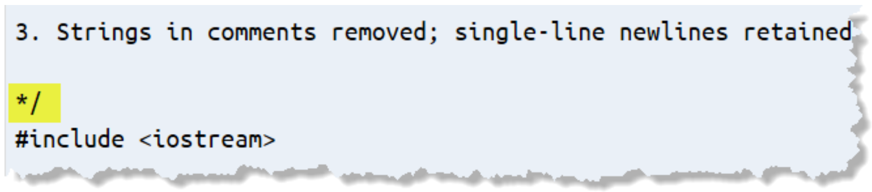
To look for the beginning of a single-line or multi-line comment, we need to either look ahead by one character (using `peek()`), or be ready to put back a character if necessary. Using `peek()` seems a little simpler to me. Here's pseudocode for this section

```
If inSingleCmt and ch is newline then
   inSingleCmt = false
ElseIf inString and ch is a quote then
   inString = false
ElseIf inMultiCmt and ch is'*' and next is '/' then
   inMultiCmt = false
ElseIf not inString, inSingleCmt, inMultiCmt then
   If ch is quote, inString = true;
   ElseIf ch is '/' and next is '*' isMultiCmt = true
   ElseIf ch is '/' and next is '/' isSingleCmt = true
```

This code goes inside the loop, before the code that echoes the output. Implement this on your own and then test it. You should get about 40% of the tests to pass. I can see that it looks like the third test is failing.


```
h07/
~/workspace/cs150/hw/h07/ $ make test
TESTING H07--sgilbert
------------------------------------------------------
Checking function: Checking the strip function. ----
   + Single line comment removed: Success
   + Comment embedded in string not removed: Success
   X Strings in comments removed; newlines retained:
   + More string-embedded comments retained: Succes
```

The test data is in the file `test-input`. You can check this file with `make-stest`. When you look at the output for test 3 on the screen, you'll see:


```
3. Strings in comments removed; single-line newlines retained

*/
#include <iostream>
```

Notice that the multi-line comment closing characters **are still being printed**.

5. Chasing Bugs

If you look back at the code, you'll see that when you encounter the end of a multi-line comment, you set the `inMultiCmt` flag to `false`, but then go ahead and print the next two characters, which seems to be the problem.

Try this for a fix. Whenever you stop a multi-line comment (set `inMultiCmt` to `false`), simply **consume those two characters** using `cin.get(ch)`. (This needs to **replace the second else-if in the pseudocode**.)

Now, when you run the tests I've provided, you'll pass more that 70%.

The eof Problem

It's **not clear at all** what's wrong with test 6 and test 7. One looks fine and the second shows that we are not correctly finishing the escaped quote. Why is that, do you suppose? Because, in the fix for the end of the multi-line comments in the previous section, **we introduced a new bug**. When consuming and ignoring the two characters, you have to think about running out of data, **and we did not**.

You can test for this with the `fail()` function but the simplest test just uses the Boolean value of the input stream itself before printing. Change the last, echo line to look like this, to make sure that we only print characters when the last character read is valid.

```
if (cin && !inSingleCmt && !inMultiCmt)
    cout.put(ch);
```

The Escaped Quote Problem

The second problem has to do with an **escaped quote**, immediately followed by a **begin-comment pair**. The quote turns "string-mode" off and the comment-pair turns on commenting (incorrectly). Here's some code to fix this; this is an additional if-else added to the first set of sequential if-else statements:

```
else if (inString && ch == '\\' && cin.peek() == '"') // \"
{
    cout.put(ch);
    cin.get(ch);
}
```

This fix works by looking for an escape character (when in a `string`), and then **peeking ahead** for a quote. If we find one, we need to print the escape character and read the next character following. This is almost identical to the multi-line end problem, except instead of ignoring the characters, we want to print them.

Because this code fragment is part of the sequential `if` statement, we **won't go into the branch that turns off string mode**, and so the quote will print as a regular character, which is what we want.

You should be able to pass all eight tests at this point, and submit your assignment using `make submit`. If you have difficulty, please ask for help on Piazza or come to my office hours, which you'll find on the syllabus.