Homework 2

This homework is due at **11 PM on Feb 8, 2017**.

Question 1:

Java (as well as C and C++) allows comments delimited by "`/*`" and "`*/`". This kind of comment can be defined in English as follows:

A comment consists of three parts:

1. a slash followed by a star, followed by
2. the body of the comment, followed by
3. a star followed by a slash.

The body of the comment can be empty, or can contain any characters except the two-character sequence "`*/`".

Note that the body of a comment ***can*** include stars and slashes, just not "`*/`".

Assume that the following JLex macros have been defined:

```
SLASH     = [/]
STAR      = [*]
```

Label each of the following JLex patterns as being correct/incorrect for the type of the comment described above.

1. `{SLASH}{STAR}[^(*/)]*{STAR}{SLASH}`
2. `{SLASH}{STAR}(.)*{STAR}{SLASH}`
3. `{SLASH}{STAR}([^*]*{STAR}+[^*/])*{STAR}+{SLASH}`
4. `{SLASH}{STAR}([^*]|[^/])+{STAR}{SLASH}`
5. `{SLASH}{STAR}[^*]*{STAR}+{SLASH}+`
6. `{SLASH}{STAR}([^*]|({STAR}+[^*/]))*{STAR}+{SLASH}`

For each incorrect pattern, explain what is wrong. For example, you might say *The pattern does not allow the comment body to include stars*, or *The pattern does allow the comment body to include */*.

If the pattern both disallows some "good" comments and allows some "bad" comments, give two explanations, one for each problem with the pattern.

Then, for each problem with the pattern, give a string that illustrates the problem; i.e., give a string that is ***not*** matched by the pattern but is a "good" comment and/or give a string that ***is*** matched by the pattern but is a "bad" comment.

Be sure that it is clear whether your example strings are intended to be "good" or "bad" comments.

Question 2:

In class, we looked at JLex, a scanner generator that produces Java code given a specification. Here is a simple application of JLex. Imagine that there is a time-intensive function `myFunc` that you are allowed to use only a limited number of times, due to performance concerns. A scanner can count the number of calls to this function. Your task is to write a JLex specification that would generate this scanner.

You should fill in this PDF appropriately and upload it for the solution.

Here are the specifications

- The scanner should print the line number **every** time it encounters a call to `myFunc` .
- The scanner should finally print the *total number of calls* to `myFunc` .
- You can assume that the source file to be scanned is in Java. (This defines the format for function calls).
- There may potentially be other variables named `myFunc`. There may also be other functions that contain `myFunc` in their names.
- However, you can assume that the declaration/definition of `myFunc` is not present in the source file to be scanned - only calls to `myFunc` are present.
- You do not know the return type of `myFunc`. So, it may appear anywhere in a statement. This might seem odd since you usually know the return type if you know the name of the function. But by assuming that any return type is possible, you generate a more generic scanner.