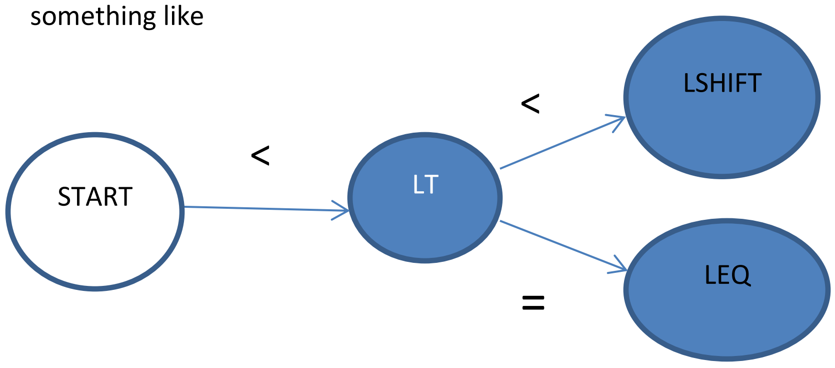
COP5555 Fall 2013
Assignment 1

Assigned August 28
Due: Sept 11 at 11am

1. Consider the attached lexical specification.
   a. Give a list of the characters that form a single character token that can be immediately recognized
   b. For each of the characters in the alphabet not listed in part a, draw the fragment of the DFA starting from seeing that character. For example, for character < you might draw something like



   Of course, when it makes sense to do so, multiple characters may be combined.

2. Write a regular expression to describe string literals in C. These are delimited by double quotes (“) and may not contain newline characters. They may contain double quote and backslash characters only if these are “escaped” by a preceding backslash. Compare with the string literals in the attached lexical specification, i.e. indicate what sorts of strings are allowed by one but not the other and vice versa.
3. Implement a scanner for the attached lexical specification.

The overall structure is similar to that used by eclipse IDEs. The basic idea is that TokenStream class is initialized with char array containing the input characters. It also contains two initially empty Token lists: tokens and comments. Your scanner takes a TokenStream instance and scans its input characters, inserting Token objects into the tokens list. Comments are also put in tokens of kind COMMENT, and these are inserted into the comments list. The Token class is an inner class of the TokenStream class.

The TokenStream class along with two inner classes Token and LexicalException has been provided. You shouldn't need to change anything in the TokenStream class for this assignment. For convenience, it has several constructors that get the input from different sources. (For example, if you want to read from a file, you can pass it a FileReader, or if the input is in a String, pass it the String).

Use the given Scanner class as a starting point and add additional methods and fields as required. Do not remove or change the signatures of anything that is given. The response to an error during scanning should be to throw a LexicalException. The contents of your error messages will not be graded, but you will be much happier later if they are informative.

Submit two files to e-learning:

A pdf file containing your answers to questions 1 and 2.

A jar file called Assignment1.jar containing Scanner.java and any other classes you might have added. You do not need to turn in your test cases or the TokenStream class. **We will recompile and call the public methods from our test program. Do not change the signatures of these methods or the package declarations or you will break the test script.**

Hints:

Work incrementally. First implement **and test** recognition of the single character tokens. Then the two character tokens, then another token kind, etc.

A class with a few JUnit tests has also been provided so that you can ensure that things are set up properly. You can run it in eclipse by right clicking on the file name, selecting Run As in the menu, and then choose JUnit Test.

If JUnit is not in your build path, an easy way to get it there is select new (in the upper left corner) and try to create a new JUnit test. In the process, eclipse will probably ask if you want to update your build path. Using JUnit is optional, but a good thing for Java programmers to know how to do. If you don't want to use JUnit, write a main method that calls the test methods and replace assertEquals(a,b) with assert a.equals(b) and make sure to run it with assertions enabled.

Useful methods **:**

- Character.isJavaIdentifierStart(char ch) == char is in A..Z,a..z,$,_
- Character.isJavaIdentifierPart(char ch) == char is in A..Z,a..z,$,_,0..9
- Character.isDigit(char ch) == char is in 0..9