Comp 2140 Assignment 2: DFA and Scanner Generator (8 points) (updated)

1 Submission and Due date

You should submit your programs at the submission site (you can click the left highlighted words to go to the submission site). The due date is Friday midnight of the due week. It is worth 8 points. There are two parts, A21 and A22.

2 A21: DFA (3 marks)

2.1 Purpose

Automaton is a machine that can run by itself. Understand how it runs by implementing it.

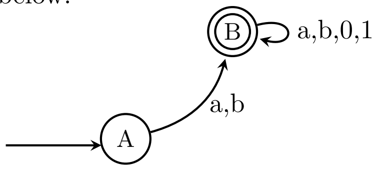
2.2 What you will write

In this assignment you need to write the DFA Simulator that can run a DFA against an input string. Given a DFA and an input string, the simulator will return yes if the DFA accepts the string, return false if the string is rejected. The Simulator algorithm in pseudo code is listed in Algorithm ??. You need to rewrite it into Java code, and make it work together with the DFA.java code. You can click the high-lighted link to download DFA.java. It is also listed at the end of this document. The code you should have the same class name and method name as is listed below:

```
public class Simulator {
        public static boolean run(DFA dfa, String input) {
        \\you need to fill in the missing part here.


        }
}
```

Your code should work for any DFAs and input strings. You should test your code using sample DFAs and input strings. One example is the DFA that corresponds the regular expression $(a|b)(a|b|0|1)*$, whose transition diagram is as below.



Its corresponding dfa.txt is:

```
a b 0 1
A B
A
B
A a B
A b B
B a B
B b B
B 0 B
B 1 B
```

The format of the dfa.txt is like below:

```
alphabet
states
start state
final state(s)
transition_1
transition_2
...
transition_n
```

The first line is a set of characters, the second line is the set of states. The third line is the start state, the fourth line is the set of final states (in this case it happens that there is only one final state).

```
Input: An input string $x$, a DFA with start state $s_0$, $move(s,c)$ function that moves state s to a new state on
       input $c$, accepting states $F$.
Output: "yes" if $D$ accepts $x$, "no" otherwise.
$s = s_0$;
while (c=nextChar())!=eof do
    s= move(s,c);
end
if $s \in F$ then
    return "yes";
end
return "no";
```

Algorithm 1: Simulate DFA (dragon book p. 151)

3 A22: Scanner generation using JLex (5 marks)

3.1 Purpose

Understand the lexical definition of a language. Generate a scanner using JLex.

3.2 Assignment Specification

The task is to write a JLex specification for TINY language. Please note that in this assignment we don't need to use all the grammar definitions there. Only the lexical part is needed. You will write a JLex specification named "A2.lex". We will run the following commands to generate the scanner, compile it, and run it. JLex installation instruction is documented here. You can find some other useful links such as Simple.lex here.

```
> java JLex.Main A2.lex
> javac A2.lex.java
> java A2
```

You should take extra care on the file names. Make sure all the three commands can run without problem, and the A2.output file is generated. If any of the three commands fails, you will receive very low marks, even 0, no matter how good the remaining part of your program is.

The A2.class program will read a text file named "A2.input", and produce a file named "A2.output" which contains following five lines:

```
numbers: NumberOfIntergersOrRealNumbers
comments: NumberOfComments
lines: NumberOfLines
quotedString: NumberOfQuotedStrings
identifiers: NumberOfIdentifiers
```

Here are the sample A2.input and the corresponding output file A2.output. Note that this time you only need to count the occurrences of the identifiers, keywords, etc. You do not need to remove the duplicates as in last assignment. Note that you don't need to write any Java programs. The scanner is generated from your lex specification.

3.3 Marking scheme

This assignment will give you bonus up to 0.5 point depending on the length of your program. If your program counts everything correct, you shall receive 5. On top of that, there is a bonus calculated using your program length.

```
yourMark=0;
if ( A2.lex file is not sent properly or file name is incorrect) return;
if (java program named A2.lex.java is generated from your lex file)
   yourMark+=0.5;
if ( generated program is compiled correctly && A2.class is generated) {
    yourMark+=0.5;
    if (your java program reads A2.input && generates result file A2.output){
            for (each of the 5 counts in A2.output) {
                  if (count is correct) {
                        yourMark+=0.8;
                   }
```

```
                        }
                }
        }
        yourMark += (lengthOfYourProgram>140)?0:96/lengthOfYourProgram;  // length is counted by
            PHP in terms of words; Note that different programs count the words in different ways.
        for (each day of your late submission)  yourMark=yourMark*0.7;
```

3.4 How to get started

Before starting this assignment, you should run our examples in the slides, especially the simple.lex example and the one immediately after this example which explains how to read from a file. Based on this example, change one thing at a time, and morph this example into A2 step by step.

3.5 Common Errors

The following are the common errors you will see during your debugging.

1. **main method not found**: Probably your code does not have a main method, or you forget to change class name from MyLexer to A2 using the following JLex directive:

   ```
    %class A2
   ```

2. **Infinite loop**: when you encounter an infinite loop, you need to check the terminating condition in your program. make sure that your code should look like as below

   ```
    while (yy.yylex()>=0){};
      ...
   %integer
   ```

   Note that while (true) in simple.lex will cause an infinite loop. It keeps waiting for an input from keyboard.

3. **"Unmatched" error**: Some input symbols can not be matched by any of your regular expressions. To ensure that all the symbols are matched by at least one of your regular expressions, you need to add the following, meaning that matching with any symbol except for new line (that is the meaning of dot) and new line.

   ```
    .|\n|\r {}
   ```

4. **ID count incorrect**: probably your regular expression for ID is wrong. In some examples we have "[a-zA-Z_][a-zA-Z0-9_]*" as the RE for ID, which is incorrect according to our language definition: we do not allow underscore, hence you need to change the RE accordingly. There are some other subtle errors, e.g., in [a-za-Z] the second 'a' should be in upper case. Another possible cause is that you put ID in front of KEYWORD like below:

   ```
   {ID} {idCount++;}
   {KEYWORD} {keywordCount++}
   ```

   In this case, keywords like INT will be recognized as an ID instead of a keyword. To solve the problem, put the line for KEYWORD in front of the line for ID.

5. **Number count incorrect**: double check your RE for number. Most common error is [0-9]+(.[0-9]+), where the dot should be replaced by "\." . A dot means matching with any character.ṁeans to match with a dot. Another common error is to write the RE as

   ```
   [0-9]+(\.[0-9]+)*
   ```

   In this case 2.3.4.5 would be recognized as one integer, which is wrong. To solve this problem, change * to ?. '?' means repeating zero or once.

6. **Comment count incorrect**: you need to use state like below. The details are in the slide. Regular expression alone won't be good enough to capture comments.

   ```
   %state comment;

   <YYINITIAL>"/**" {...}
   <comment>"**/" {... }
   ```

4 Academic Integrity

By submitting on the web site, you verify that the submitted work is your own and adheres to all my Academic Rights and Responsibilities as outlined in the Student Code of Conduct.

5 Appendix: DFA.java

```
import java.util.TreeMap;
import java.util.Arrays;
import java.util.HashSet;
import java.util.Set;
import java.io.*;

public class DFA {
        public static String [] alphabet;
        public String startState;
        public Set<String> finalStates;

    public static TreeMap<String, String> transitions=new TreeMap<String, String>();

    /** Construct a DFA from a text file
     */
    public DFA(String filename) throws Exception{
        BufferedReader br  = new BufferedReader(new FileReader(filename));
        alphabet=br.readLine().trim().split(" ");
        String[] states=br.readLine().split(" ");
        startState=br.readLine().trim();
        String[] finals=br.readLine().trim().split(" ");
        finalStates= new HashSet<>(Arrays.asList(finals));
        String line="";
        while ((line=br.readLine())!=null) {
                String[] transition=line.trim().split(" ");
                transitions.put(transition[0]+"_"+transition[1], transition[2]);
        }
    }

    public static void main(String[] args) throws Exception{
        DFA dfa = new DFA (args[0]);
        String input = new BufferedReader(new FileReader(args[1])).readLine().strip();
        boolean result=Simulator.run(dfa,input);
        BufferedWriter bw=new BufferedWriter(new FileWriter(args[2]));
        bw.write(result+"");
        bw.close();
     }
}
```