Midterm Review Questions

In assignment 1 and 2, you are required to write several regular expressions. Please write regular expressions for recognizing :

1. Quoted Strings
2. Identifiers
3. Real numbers

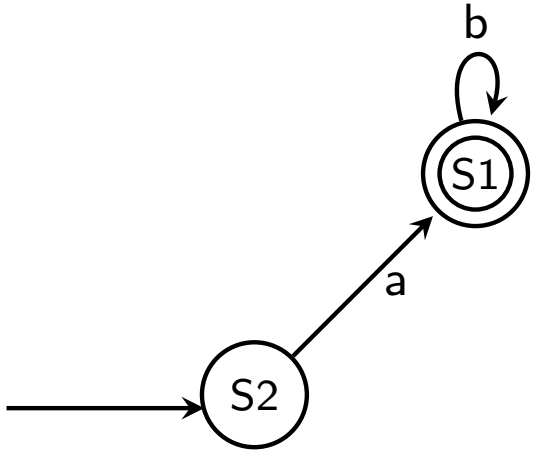
Fill in the missing code for processing comments in programming languages

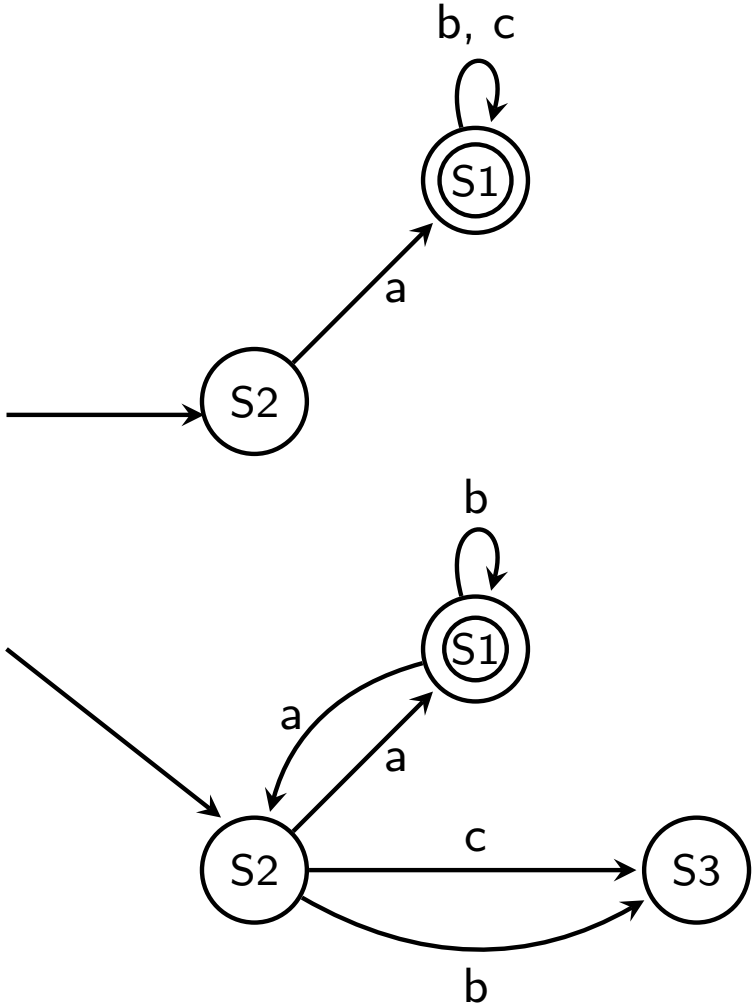
```
%state comment;
%%
<YYINITIAL>"/**" { ---------------------- }
<comment>"**/" {   ---------------------- }
\r|\n {}
. {}
```

A grammar is formally defined as a 4-tuple (Σ, N, P, S). Write the names for those four components:

1. Σ:
2. N:
3. P:
4. S:

Given the following transition diagrams. Write the corresponding regular expressions.




b, c
S1
a
S2
b
S1
a
a
S2
c
S3
b

Given the following grammar, where `A` is a non-terminal, `a` and `b` are terminals:

$$A \rightarrow aAb|\epsilon$$

1. What kind grammar is it in Chomsky Hierarchy? Is it a regular grammar, context free, or context sensitive grammar?

2. Write one example string that can be derived from the grammar.

3. Is it possible to have an equivalent regular expression?

Write *grammars* (production rules only) for the following languages over alphabet a and b.

1. all possible palindromes over alphabet $\{\mathtt{a}, \mathtt{b}\}$, i.e., the strings read the same backward and forward, such as bb, bab, aabbabbaa.

2. Strings that have twice number of a's as b's, such as aab, aababa, and aba.

3. Strings with one a and even number of b's, such as abb, abbbb, babbb.

Write *regular expressions* for the following languages:

1. Strings over **a** and **b** of odd length, such as **a, aba, bbabb**

2. Strings for all odd numbers , such as **3, 13, 23, 1113**.

Spell out the following acronyms used in this course:

1. NFA
2. DFA
3. FSM
4. CFG
5. RE

Given the following grammar where `S`, `A` are non terminals and `a`, `b` are terminals.

$$S \rightarrow bA \tag{1}$$
$$A \rightarrow aA \tag{2}$$
$$A \rightarrow \epsilon \tag{3}$$

1. Draw an automaton for the grammar;
2. Write its corresponding regular expression.

Given the regular expression $(a|b)(a|b|c)^*$,

1. draw its corresponding automaton;
2. write the corresponding *regular* grammar;
3. write one string that matches this regular expression.

The following is the algorithm to simulate DFA.

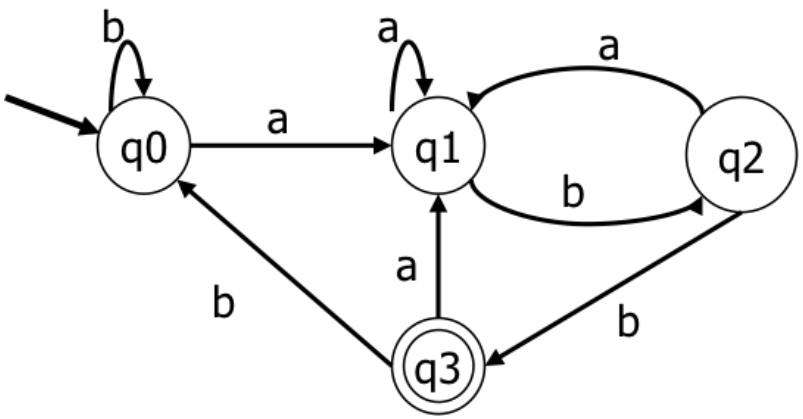
Algorithm 1: Simulate DFA (dragon book p. 151)

Input: An input string x, a DFA with start state s_0, accepting states F.
Output: "yes" if D accepts x, "no" otherwise.

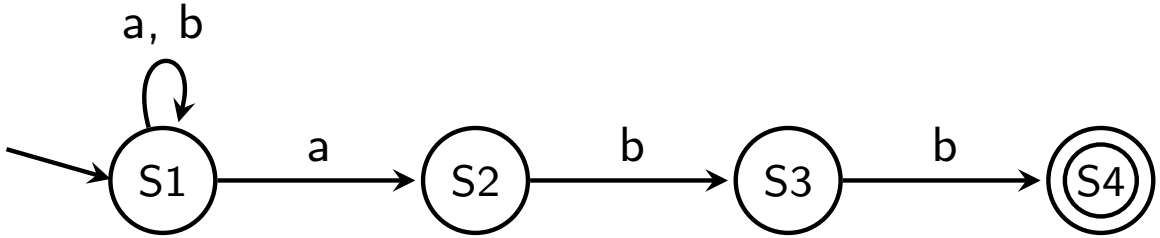
```
s = s0;
while (c=nextChar())!=eof do
    s= move(s,c);
end
if s ∈ F then
    return "yes";
end
return "no";
```

Given a DFA depicted in the digram below, and a string abbabb.

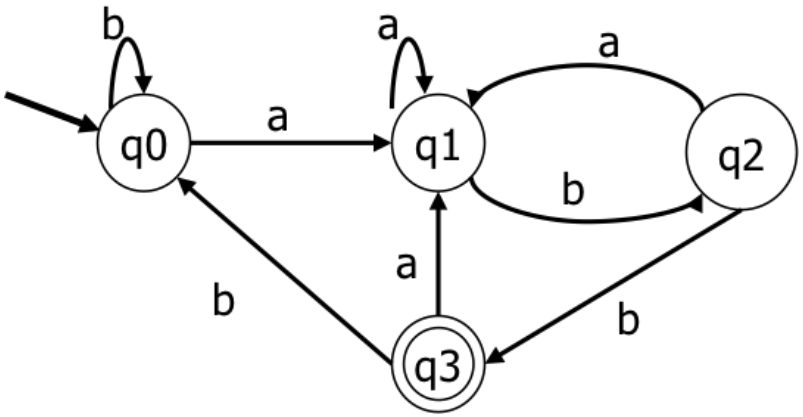
1. Write down the trace to run the DFA on that string.
2. Whether the string can be accepted by the DFA?



Transform the following NFA to DFA



Solution:



The following grammar for If statement is ambiguous. Rewrite the grammar to remove the ambiguity, by enforcing that *else* always matches with the closest *then*.

$$
\begin{aligned}
\mathit{ifStmt} &\rightarrow \mathit{if}\ \mathit{expr}\ \mathit{then}\ \mathit{stmt} && (1)\\
&\mid \mathit{if}\ \mathit{expr}\ \mathit{then}\ \mathit{stmt}\ \mathit{else}\ \mathit{stmt} && (2)\\
\mathit{stmt} &\rightarrow \mathit{ifStmt} && (3)\\
&\mid \mathit{assignmentStmt} && (4)
\end{aligned}
$$

Write the second part of assignment one (A12.java). Note that in the second part you are required to use Regex.

The following questions are relevant to assignment A2. Suppose that KEYWORD is the macro that defines the list of keywords correctly.

1. Given the following test input file and the regular expressions in the lex file, what are the values of `id` and `keyword`?

```
BEGIN x1
END
```

```
%%
<YYINITIAL>\"[^\"]*\"  {quotedstring++;}
<YYINITIAL>[a-zA-Z][a-zA-Z0-9]*  {id++;}
<YYINITIAL>{KEYWORD}  {keyword++;}
<YYINITIAL>[0-9]+  {num++;}
<YYINITIAL>"/**"  {yybegin(comment);}
<comment>"**/"  {yybegin(YYINITIAL);}
\r|\n  {}
.  {}
```

Given the following test case and regular expressions, there will be an error message `unmatched input`. Explain what caused this error. How to correct the lexer so that it can still count the identifiers etc., even though the input contains illegal characters?

```
BEGIN x1 & @
END
```

```
%%
<YYINITIAL>\"[^\"]*\"  {quotedstring++;}
<YYINITIAL>{KEYWORD}  {keyword++;}
<YYINITIAL>[a-zA-Z][a-zA-Z0-9]*  {id++;}
<YYINITIAL>[0-9]+  {num++;}
<YYINITIAL>"/**"  {yybegin(comment);}
<comment>"**/"  {yybegin(YYINITIAL);}
\r|\n  {}
```

Whether the `unmatched input` will occur for the following input? Explain your answer.

```
BEGIN x1 "&" "@"
END
```