TUTORIAL 2

DATE: January 23, 2023

1. Consider a fragment of the grammar of a dialect of Fortran:

   $$\begin{array}{lcl} st & \rightarrow & assgn \mid do \mid if \mid labassgn \mid gotostmt \\ term & \rightarrow & \texttt{id} \mid \texttt{num} \mid \texttt{real} \mid term{+}term \\ if & \rightarrow & \texttt{if}\ (cond)\ gotostmt \\ do & \rightarrow & \texttt{do num id} = \texttt{num, num} \\ assgn & \rightarrow & \texttt{id} = term \mid \texttt{id(id)} = term \\ labassgn & \rightarrow & \texttt{num}\ assgn \\ cond & \rightarrow & \texttt{id} \mid term\ \texttt{.ge.}\ term \\ gotostmt & \rightarrow & \texttt{goto num} \end{array}$$

   `id` is defined in the usual way. A `num` is a natural number, and a `real` is, in general, two nums separated by a decimal point. Only one of the two nums may be absent.

   Draw a single DFA for all the tokens in the language. Against each final mention the token recognized.

2. Consider the following Lex-like specification of three distinct tokens:

$(aba)+$	Token 1
$ab*a$	Token 2
$a(a\|b)$	Token 3

   (a) Draw an appropriate syntax tree for the specification which can be directly converted to a DFA. Be careful, because the resulting DFA has to distinguish between lexemes corresponding to the three tokens.

   (b) Number the leaves of the syntax tree from left to right. Now construct the DFA labeling each state with sets of positions.

   (c) Indicate the token(s) found against each accepting state. Are there any clashes? If yes, why?

3. The time taken to do lexical analysis, it is generally thought, is linear in the length of the input string. The question below shows that this is not always the case.

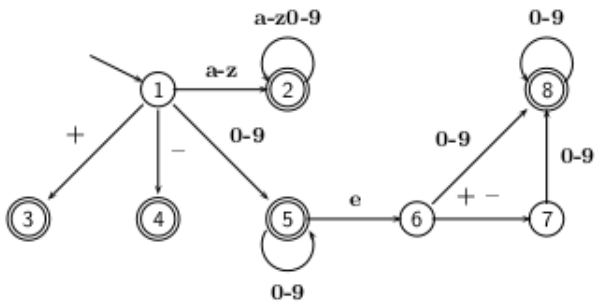
   Consider the following Lex-like description of two tokens:

   abc – Token 1
   $(abc)^*d$ – Token 2

   (a) Draw the DFA corresponding to this regular expression. Indicate against each final state, the token recognised by the state.

   (b) Give an input string whose complete tokenization (by complete tokenization we mean extraction of all the tokens in the given string) would take time that is quadratic in its length.

4. Suppose the DFA shown below were used to find the tokens in an input file. What is the maximum number of characters that lex (or flex++) might have to examine past the end of a token to decide that the token has been found. Show an input string in which the token that illustrates this feature.



6. Do some experiments to figure out how string literals are represented in C. In particular, find out what happens if you want to represent a " in a string. Also find out how you can represent strings that spread over multiple lines.

   Now write a regular expression for strings in flex++. For both the problems above, you must think of arguments to show why your regular expressions are correct.

7. Explain why the following flex++ pattern which is supposed to represent a comment is incorrect:

   ```
   "/*"[^*]*(\*([^/][^*]*)?)*"*/"
   ```

8. Show how the DFA represented below using a 2d-array and having $7 \times 6 = 42$ entries can be stored in less space using the 4-arrays scheme. How much space (in terms of the number of entries) is being used by your scheme.

Symbol State	a	b	c	d	e	f
1	_	_	2	_	4	7
2	_	_	_	7	6	_
3	_	6	3	_	_	_
4	_	_	1	_	4	3
5	5	6	3	_	_	_
6	_	_	2	_	4	3
7	7	6	3	_	_	_