CS3342 –
Assignment 1
due Feb. 8, 2024
2-day no-penalty extension until: Feb. 10, 11:59pm

1. (10pt) Consider a programming language that has comments as follows:

- comments start with a line beginning with three consecutive equal signs, "===", followed by any characters until the end of the line (and a newline '\n' at the end)
- comments end with a line consisting of equal signs only, at least three (and a newline '\n' at the end)

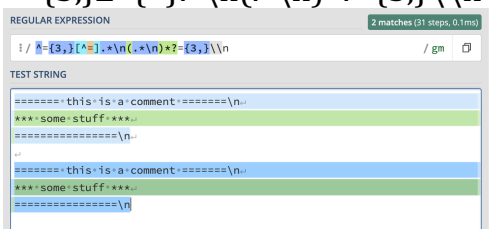
Example of a comment (newlines are shown for clarity):

```
======= this is a comment =======\n
*** some stuff ***
================\n
```

Write a regular expression that describes precisely the above described comments. You can use the notation $\Sigma - \{a\}$ to describe the set of all characters different from a.

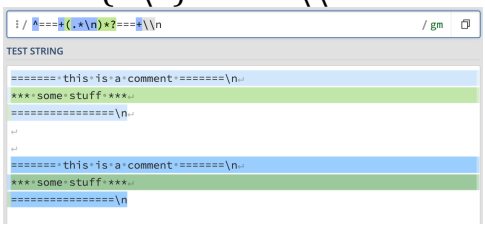
The regular expression would be:

^={3,}Σ−{=}.*\n(.*\n)*?={3,}\\n



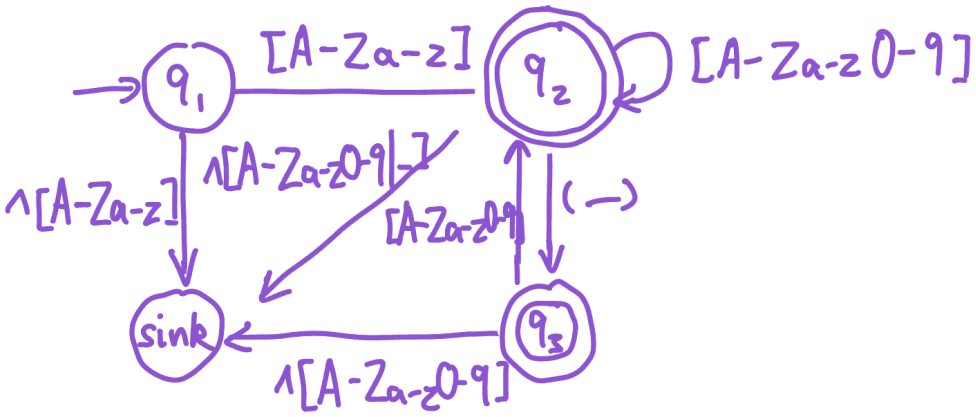
Or

^===+(.*\n)*?===+\\n



Reference: works on regex101.com

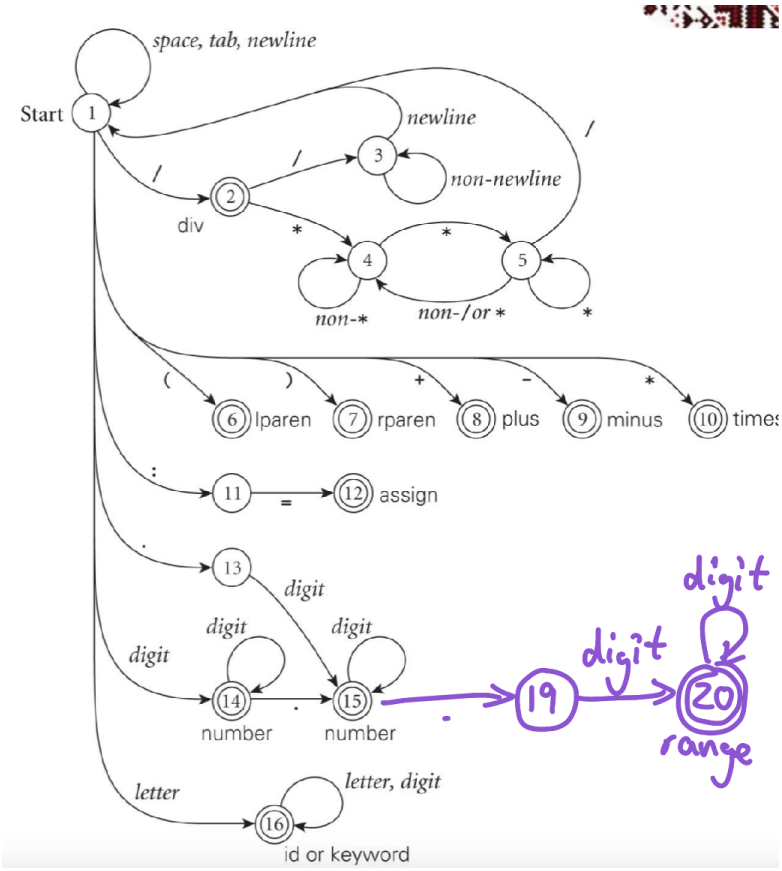
2. (5pt) A scanner is built for a language where the identifiers start with a letter followed by any number of letters, digits, or underscores, such that there are no consecutive underscores. Draw a deterministic finite automaton (DFA) with fewest states that accepts all identifiers and nothing else.



3. (10pt) Consider the simple calculator language (slide 16 of scanning) and add to it ranges:

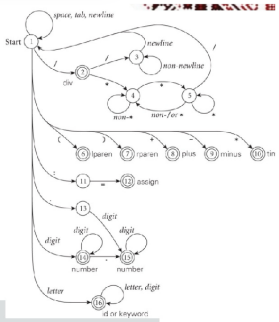
$$range \longrightarrow digit\ digit^* \ldots digit\ digit^*$$

Modify the scanner table and DFA on slide 28 to reflect this change. (Note that the DFA already requires some small changes to match the table, as discussed in class.) To save time, you are allowed to cut-and-paste the table and DFA (as figures) and only indicate the changes.



Scanning

- Scanner table used by previous code
  - state 17: white spaces; state 18: comments
  - `scan_tab`: entire table but last column
  - `token_tab`: last column
  - `keyword_tab` = {**read**, **write**}



State	space, tab	newline	/	*	(	)	+	−	:	=	.	digit	letter	other	
	Current input character														
1	17	17	2	10	6	7	8	9	11	–	13	14	16	–	
2	–	–	3	4	–	–	–	–	–	–	–	–	–	–	div
3	3	18	3	3	3	3	3	3	3	3	3	3	3	3	
4	4	4	4	5	4	4	4	4	4	4	4	4	4	4	
5	4	4	18	5	4	4	4	4	4	4	4	4	4	4	
6	–	–	–	–	–	–	–	–	–	–	–	–	–	–	lparen
7	–	–	–	–	–	–	–	–	–	–	–	–	–	–	rparen
8	–	–	–	–	–	–	–	–	–	–	–	–	–	–	plus
9	–	–	–	–	–	–	–	–	–	–	–	–	–	–	minus
10	–	–	–	–	–	–	–	–	–	–	–	–	–	–	times
11	–	–	–	–	–	–	–	–	–	12	–	–	–	–	
12	–	–	–	–	–	–	–	–	–	–	–	–	–	–	assign
13	–	–	–	–	–	–	–	–	–	–	–	15	–	–	
14	–	–	–	–	–	–	–	–	–	–	15	14	–	–	number
15	–	–	–	–	–	–	–	–	–	–	19	15	–	–	number
16	–	–	–	–	–	–	–	–	–	–	–	16	16	–	identifier
17	17	17	–	–	–	–	–	–	–	–	–	–	–	–	white_space
18	–	–	–	–	–	–	–	–	–	–	–	–	–	–	comment

19
20

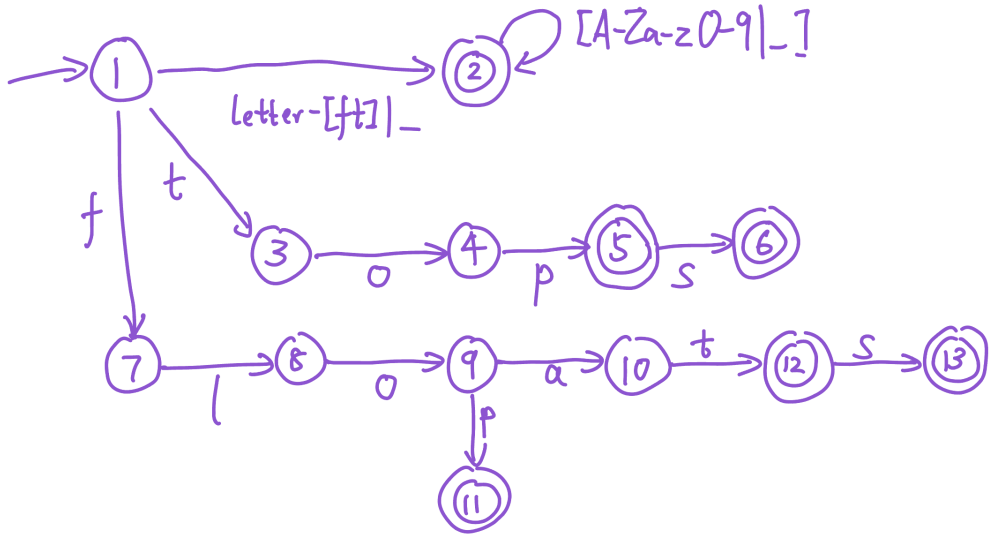
20
20

range

4. (10pt) Keywords fit the definition of identifiers and scanners identify them as such, and then look them up a table of keywords, because otherwise the DFA is unnecessarily larger. This question addresses the size of this DFA. Assume that the identifiers and the (made-up) keywords are defined as follows:

$$
\begin{aligned}
\textit{identifier} &\longrightarrow (\texttt{\_} \mid \textit{letter})(\texttt{\_} \mid \textit{letter} \mid \textit{digit})^* \\
\textit{letter} &\longrightarrow \texttt{a} \mid \texttt{b} \mid \cdots \mid \texttt{z} \\
\textit{digit} &\longrightarrow \texttt{0} \mid \texttt{1} \mid \cdots \mid \texttt{9} \\
\textit{keyword} &\longrightarrow \texttt{float} \mid \texttt{floats} \mid \texttt{flop} \mid \texttt{top} \mid \texttt{tops}
\end{aligned}
$$

Draw a DFA with fewest states that recognizes the above keywords directly. That is, there will be separate accepting states for keywords and different ones for identifiers. You can use any meaningful notation such as $\textit{letter} - \{\texttt{a}, \texttt{b}, \cdots\}$ or $\neq \texttt{a}$ to denote various subsets of characters.



5. (15pt) Consider the following grammar, G:

1.	P	$\longrightarrow$	S `$$`	6.	O	$\longrightarrow$	`else` T
2.	S	$\longrightarrow$	`if (e) then` M O	7.	O	$\longrightarrow$	ε
3.	S	$\longrightarrow$	`other`	8.	T	$\longrightarrow$	`if (e) then` T
4.	M	$\longrightarrow$	`if (e) then` M `else` M	9.	T	$\longrightarrow$	`other`
5.	M	$\longrightarrow$	`other`				

(a) (5pt) Compute the sets FIRST(X) and FOLLOW(X), for all non-terminals X, and the sets PREDICT(p) for all production rules p.

(b) (5pt) Prove that G an $LL(1)$ grammar.

(c) (5pt) We said that there is no top-down grammar for `if..then..else` statements. Does G contradict this? Prove your answer.

X	FIRST(x)	FOLLOW(x)
P	if, other	∅
S	if, other	$$
M	if, other	else, ε
O	else, ε	∅
T	if, other	∅

i	PREDICT(i)
1. P→S$$	if, other
2. S→if (e) then M O	if
3. S→other	other
4. M→if (e) then M else M	if
5. M→other	other
6. O→else T	else

7.	O→ε	$$
8.	T→if (e) then T	if
9.	T→other	other

Q5: LL(1) definition is on slide 17 of LL-parsing. See also slide 19 of LR-parsing.

2) Prove that G an LL(1) grammar.
For any production: A→u|v

First(u) n First(v) = ∅
If v => ε then First(u) n Follow(A)= ∅
At most one of u and v can derive the empty string ε

According to the predict table, since there is no duplicated non-terminal derive the same terminal, which mean LL(1) is hold for this grammar.

6. (50pt) Write a Python program `comm rm.py` to remove all comments from a Python program. The program should work as follows:

Some ideas I had in mind about this question:

- the comments to be removed include single line comments and triple-quoted strings (single or double quotes) acting as comments

- if used in expressions, triple-quoted strings are not to be removed, as they do not act as comments

- you need to consider also:

  - backslash-escaped characters

  - backslash as line continuation character

  - the interaction between comments and single-quoted strings

```
> python comm_rm.py input_file.py
```

where `input_file.py` is any (correct) Python program. The program with comments removed will be saved in the file `input_file_rm.py`.

You are *not allowed* to use the regular expression capabilities of Python, as that would defeat the purpose of the question.

READ ME! Submit all your responses as a single pdf file. Solutions should be typed but high-quality hand-written solutions are acceptable. **Source code, if required, is submitted separately.**

JFLAP: You are allowed to use JFLAP to help you solve the assignment. You still need to explain clearly your solution. Also, make sure you understand what it does; JFLAP will not be available during exams!

LLMs: You are allowed to use LLMs (Large Language Models), such as ChatGPT, but, again, they will not be available during exams.

LaTeX: For those interested, the best program for scientific writing is LaTeX. It is far superior to all the other programs, it is free, and you can start using it in minutes; here is an introduction: https://tobi.oetiker.ch/lshort/lshort.pdf. It is also available online at https://www.overleaf.com/.