CSE 262: Quiz #1  
Due September 16th, 2022 at 11:59 PM

The quiz has TWO questions. Please submit your answer by updating this file in the quizzes folder of your Bitbucket account, and then committing and pushing. You should use as much space as you want for each answer. Please be detailed in your answers. Remember: this quiz is worth 9% of your grade, and you will not receive very many points if you do not give detailed answers.

**Question 1:** Extend the grammar from Figure 2.25 with an exponent operator (^) that has higher precedence than multiplication and division, and with a modulus operator (%) that has lower precedence than multiplication and division, but higher precedence than addition and subtraction. Accompany your proposed grammar with text explaining what you did and why.

1. program −→ stmt list $$ // this is the new and extended proposed grammar
2. stmt list −→ stmt list stmt // the highlighted parts are what ive added and changed
3. stmt list −→ stmt // below this grammar is my explanation and
4. stmt −→ id := expr
5. stmt −→ read id
6. stmt −→ write expr
7. expr −→ bar
8. expr −→ expr add\_op bar
9. bar −→ term
10. bar −→ bar mod\_op term
11. term −→ foo
12. term −→ term mult\_op foo
13. foo −→ factor
14. foo −→ foo exp\_op factor
15. factor −→ ( expr )
16. factor −→ id
17. factor −→ number
18. add op −→ +
19. add op −→ -
20. mod\_op −→ %
21. mult\_op −→ \*
22. mult\_op −→ /
23. exp\_op −→ ^

In my new proposed grammar, I made two new symbols called “bar” and “foo”. The bar symbol’s production is meant for the modulus operation (%) and the foo symbol’s production is meant for the exponential operation (^). This grammar before being extended was already written in left recursion (as indicated by the recursive call being on the left in the production of symbols “expr” and “term”), and so I followed the left recursive writing (as indicated by the recursive call also being on the left in the production of the new symbols “bar” and “foo”). In addition, left recursion is good because of reduction and left associativity (evaluated left to right precedence for same type of operators).

To explain the order in which I made my grammar to give it its respective precedence, I would like to explain the parsing tree first. The parsing tree executes from the leaves of the tree (from left to right) up to the root of tree. Therefore, for the respective operation to have a higher precedence than another operation, it would have to be on a lower level in the tree (towards the leaves) than the level of an operation with lower precedence. Therefore, for the exponential operation to have a higher precedence than multiplication and division, the symbol; “foo” had to be on a lower level in the tree than the symbol “term”. That also means that for the modulus operation to have lower precedence than multiplication and division, “bar” had to be on a higher level in the tree than the symbol “term”.

The execution of the non-terminals (assuming each unique symbol is executed) goes from program- > stmt -> expr -> bar -> term - > foo -> factor (from highest level to lowest level of tree) which shows that my new grammar does have the correct level of precedence. Furthermore, in the production of “exp”, “bar”, “term”, and “foo”, they recursively call themselves with their own non-terminal. Therefore, multiple of their own operations are still being done at higher levels in the tree than the operations with higher precedence which should be at a lower level in the tree. (Additional note, taking either the top to bottom approach or bottom-up approach does not matter as it would form the same tree.)

**Question 2:** In your repository’s p1 folder, you will find the files `scanner\_ids\_nums.png` and `scanner\_keywords.png`. When we combine them (by merging the common states), the result is a nondeterministic finite automata. First, you should explain why it is nondeterministic. Then, you should create a new diagram consisting of the composition of the two automata, but with whatever changes would be needed in order to make it deterministic. Be sure to explain what you did, and why. Is your construction *minimal*?

A deterministic finite automaton knows what state to transition to based on a given input. Therefore, there may not be two transitions from a given state with the same input and there also may not be epsilon transitions. Merging the common states of `scanner\_ids\_nums.png` and `scanner\_keywords.png` results in a nondeterministic finite automata because it has epsilon transitions and no transitions at certain states given a certain input.