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 $$

2. stmt list −→ stmt list stmt

3. stmt list −→ stmt

4. stmt −→ id := expr

5. stmt −→ read id

6. stmt −→ write expr

7. expr −→ term

**21.expr->term modu op term**

**8. expr −→ expr add op expr**

9. term −→ factor

**20. term-> factor expo op factor**

**10. term −→ term mult op term**

11. factor −→ ( expr )

12. factor −→ id

13. factor −→ number

14. add op −→ +

15. add op −→ -

16. mult op −→ \*

17. mult op −→ /

**18. exop op -> ^**

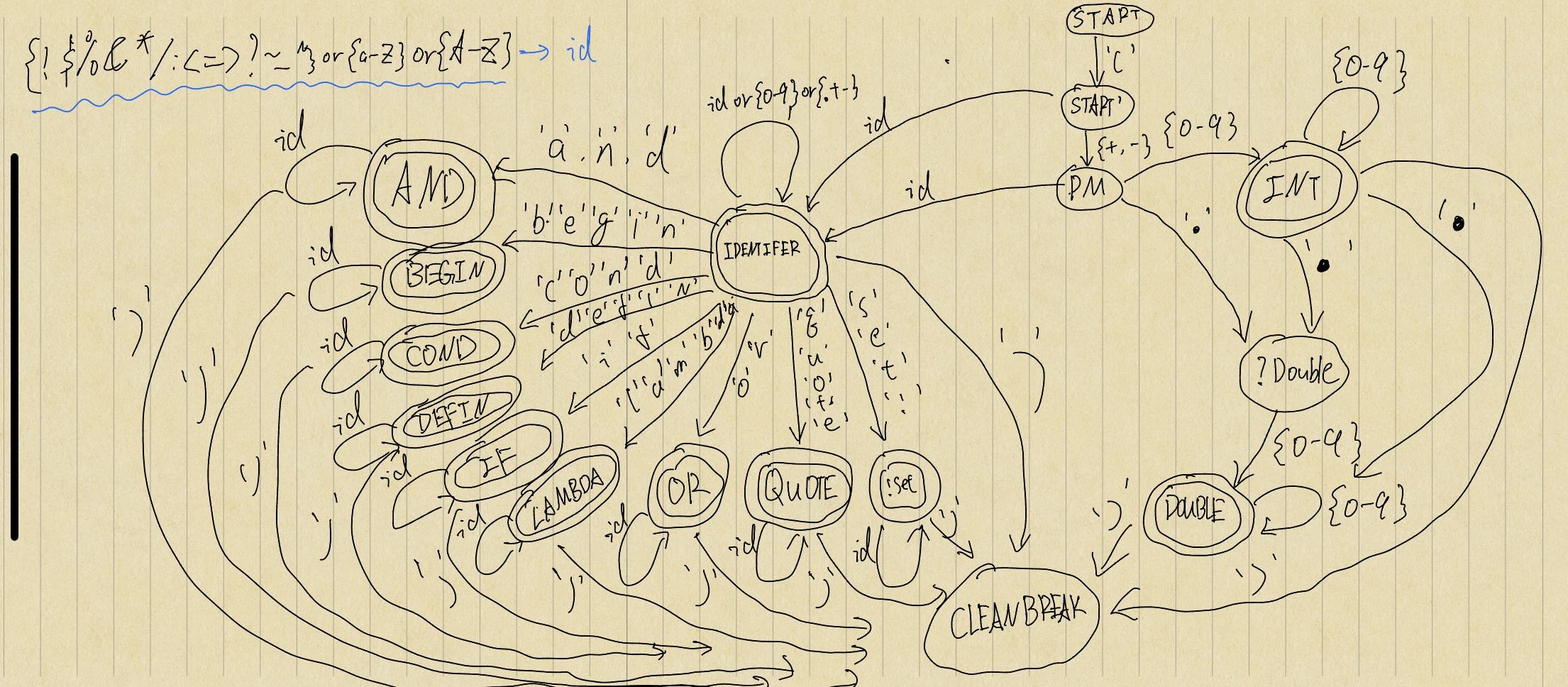
**19. modu op -> %**

Explanation: (everything I did start from

1. Add two target operation in to grammar(^ and %)
2. Add in 20. Since exponential operation has the highest precedence, it will be a state before any other operation. Therefore, I add it as the child state of term
3. Add in 20. Since there are only three circumstances in operation, the child state it will perform operation on will be either an id, a number or an expression with other operations in a parenthesis(). In this case, factor state is the best choice
4. There is no change in the add op, since it has the lowest precedence, and both mult op and expo op are child of term. If they are passed into the add op, they will be performed first.
5. Changed the second state in the mult op to term, which allow expo op perform before mult op
6. Add in 21, since the precedence order is : add < modulus< multi < exponent, therefore, we should create modu op as the child of experiment, preventing the circumstance that it operate before multi or exponent without ()
7. Two state that it would use to perform the operation, can be id, number, result of multi or exponent and add op inside the parenthesis(), therefore, term will be the best choice, which will not refer to add op but the rest state
8. Changed the child state of add op to expre, which will refer to all other states, for the reason that the add op has the lowest precedence

**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*?

1. It is an NFA for the reason that it contains epsilon transition which requires no input to do the state transition. Also, there are more than one transition to different states with same input, for example, if the input is ‘a’ ‘n’ ‘d’, it can be referred to IDENTIFER and AND states.



1. Firstly, I create two state which can be reached by ‘(‘ and ‘)’, which is used to do the transition from any states to the CLEANBREAK. On the right hand side, it is the transition between integer and double. Firstly, we look at the symbol (+,-)， then looking for a number from 0-9 to get into INT state, and continuously reading the rest of the number until no number left, and a ‘)’ will bring it to final state. If, during reading number or read a ‘.’ before INT state, we will dive into the ?Double state where we can get a DOUBLE from there if we read a following number. A closure applied on DOUBLE state to read the rest number, and break when we read ‘)’. On the top left corner, since the input of IDENTIFIER is too long, I just use ‘id’ to refer it. On the left hand side, if we any character belongs to id at beginning or after we get {+.-}, it will make a transition to IDENTIFIER which also applies closure on it self. Within IDENTIFIER, if we read any combination of characters that matches the key of those method, we switch to the corresponding method, and closure to read the parameter of method. If there is nothing left, a ‘)’ will bring us to break.
2. I don’t think it is minimal, since I didn’t follow the method that introduced in the text book to minimize the DFA, which partition the class X in to different equivalent k classes until there is no class to partition. All I did is just give an input to any transition between states.