Assignment 2 & 3

1. [10 points] Describe some strings that are represented by the following regular expressions. **NOTE:** each bullet point contains a *single* regular expression
   - -?[0-9]+(*10\^)?[1-9]*

     -3*10^45
     678
     -20*10^1

   - [A-Z]+ and ([A-Z]+|\.\.\.)

     H and M…
     VL and CD…

2. [10 points] Build the regular expression for the following descriptions. Don’t forget to take the space character in description two into account. **NOTE:** This question *must* be submitted in a digital format (no scans)
   - Identifiers in a language that must start with an underscore character and must end in a numeric digit. The length can be any size and values in between can be any alphanumeric character.

     _ ([A-Z] | [0-1]) *

   - A phone number with either the following formats: (888) 888-8888 or 888-888-8888

     (\([0-9] [0-9] [0-9] \) | [0-9] [0-9] [0-9]) \- [0-9] [0-9] [0-9] \- [0-9] [0-9] [0-9] [0-9]

   - The VCU V Number

     V00 [0-9] [0-9] [0-9] [0-9] [0-9] [0-9]

NOTE: For all following grammars, only the →| { } characters are valid metasymbols. If these characters are enclosed in single quotes, they should be treated as terminals and not metasymbols. All other characters should be treated as terminals.

3. [10 points] Using the grammar below, provide a leftmost derivation for the following statement:

    - if (hello == 8) { world = 16 ^ 2 }

```
if-statement → if ( condition ) '{' assign-stmt '}'
condition →var == int
assign-stmt →  var = stmt
stmt → int + int | int - int | int ^ int
var → hello | world
int → int digit | digit
digit → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

```
If-statement -> if ( condition )  ‘ { ‘  assign-stmt  ‘ } ‘
If-statement -> if ( var == int )  ‘ { ‘  assign-stmt  ‘ } ‘
If-statement -> if ( hello == int )  ‘ { ‘  assign-stmt  ‘ } ‘
If-statement -> if ( hello == digit )  ‘ { ‘  assign-stmt  ‘ } ‘
If-statement -> if ( hello == 8 )  ‘ { ‘  assign-stmt  ‘ } ‘
If-statement -> if ( hello == 8 )  ‘ { ‘  var = stmt  ‘ } ‘
If-statement -> if ( hello == 8 )  ‘ { ‘  world = stmt  ‘ } ‘
If-statement -> if ( hello == 8 )  ‘ { ‘  world = int ^ int  ‘ } ‘
If-statement -> if ( hello == 8 )  ‘ { ‘  world = int digit ^ int  ‘ } ‘
If-statement -> if ( hello == 8 )  ‘ { ‘  world = 1 digit ^ int  ‘ } ‘
If-statement -> if ( hello == 8 )  ‘ { ‘  world = 16 ^ int  ‘ } ‘
If-statement -> if ( hello == 8 )  ‘ { ‘  world = 16 ^ 2  ‘ } ‘
```

- if (myVar <= 5) { otherVar = 256 % 8 }

```
if-statement → if (condition) '{' assign-stmt '}'
condition → var == | <= | >= | < | > int
assign-stmt → var = stmt
stmt → int [+\-*\/%^] int
var → [a-zA-Z]*
int → [0-9]+
```

If-statement -> if (condition) ‘ { ‘ assign-stmt ‘ } ‘
If-statement -> if (var <= int) ‘ { ‘ assign-stmt ‘ } ‘
If-statement -> if ([a-zA-Z]*<= int) ‘ { ‘ assign-stmt ‘ } ‘
If-statement -> if (myVar <= int) ‘ { ‘ assign-stmt ‘ } ‘
If-statement -> if (myVar <= 5) ‘ { ‘ assign-stmt ‘ } ‘
If-statement -> if (myVar <= 5) ‘ { ‘ var = stmt ‘ } ‘
If-statement -> if (myVar <= 5) ‘ { ‘ var = int [+\-*\/%^] int ‘ } ‘
If-statement -> if (myVar <= 5) ‘ { ‘ var = [0-9]+ [+\-*\/%^] int ‘ } ‘
If-statement -> if (myVar <= 5) ‘ { ‘ var = 256 [+\-*\/%^] int ‘ } ‘
If-statement -> if (myVar <= 5) ‘ { ‘ var = 256 % int ‘ } ‘
If-statement -> if (myVar <= 5) ‘ { ‘ var = 256 % [0-9]+ ‘ } ‘
If-statement -> if (myVar <= 5) ‘ { ‘ var = 256 % 8‘ } ‘

5. [10 points] Using the grammar below, how many legal sentences are there? Why is that? Suppose white space was completely ignored in the grammar so that sentences could be written as “thecompetitorseesawin.” Can this grammar still be parsed? Explain

```
sentence → noun-phrase verb-phrase
noun-phrase → article noun
article → a | and | the
noun → girl | competitor | win | dog | comp
verb-phrase →  verb noun-phrase
verb → sees | permits | objects
```

sentence → noun-phrase verb-phrase

sentence → article noun verb-phrase

sentence → a | and | the noun verb-phrase

sentence → 3 * noun verb-phrase

sentence → 3 * girl | competitor | win | dog | comp verb-phrase

sentence → 3 * 5 * verb-phrase

sentence → 3 * 5 * verb noun-phrase

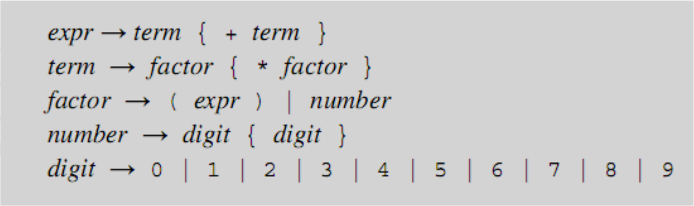
sentence → 3 * 5 * sees | permits | objects noun-phrase

sentence → 3 * 5 * 3 * noun-phrase

sentence → 3 * 5 * 3 * 3 * 5

= 675

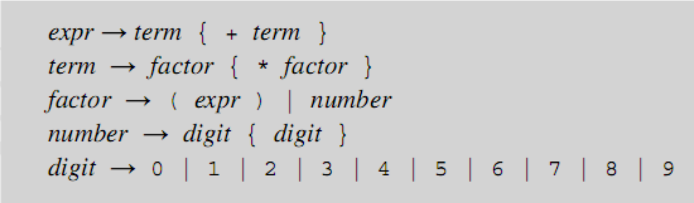
6. [20 points] Add the following three operations in the proper location for the order of operations to apply to the EBNF grammar below. Please provide a complete EBNF grammar for your answer.
    - subtraction,
    - division,
    - exponents



```
expr → term { + term }
term → factor { * factor }
factor → ( expr ) | number
number → digit { digit }
digit → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

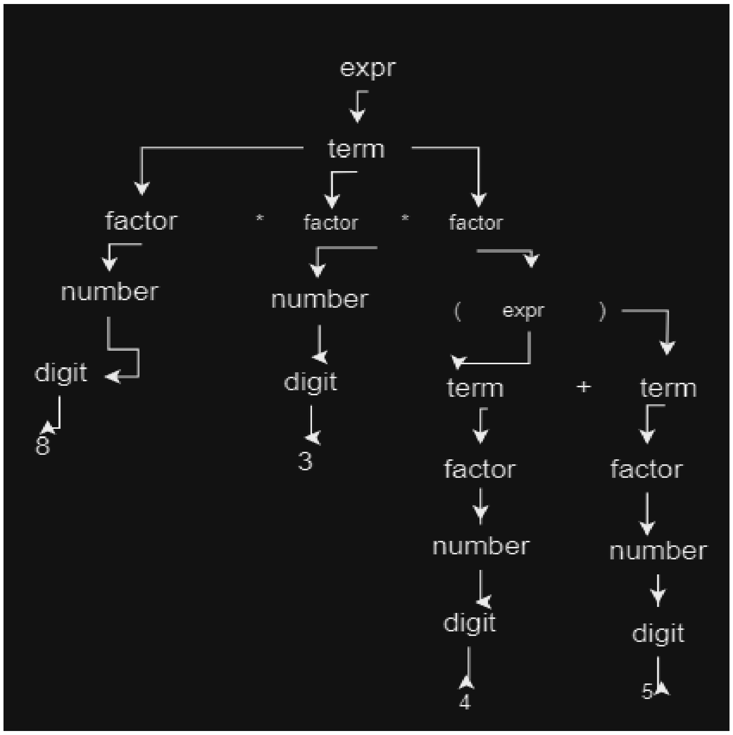
```
expr -> term { + | - term }
term -> factor { * | / factor }
term -> { expr } | number { ^ number }
number -> digit { digit }
digit -> 0 | 1 |2 |3 |4 |5 |6 |7 |8 |9
```

7. [20 points] Using the grammar below, draw the following:

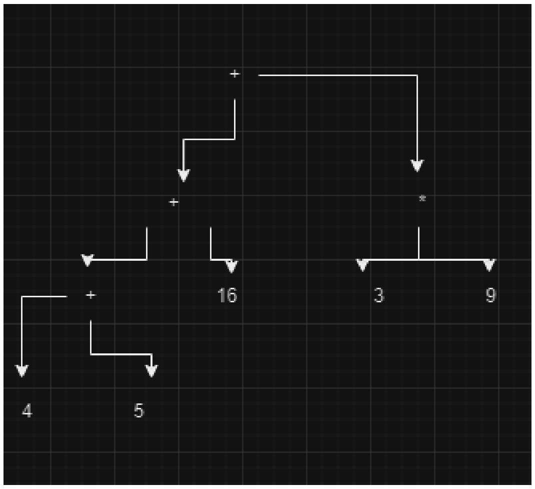


expr → *term* { + *term* }
term → *factor* { * *factor* }
factor → (*expr*) | *number*
number → *digit* { *digit* }
digit → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

- (8 * 3 * (4 + 5)) – parse tree

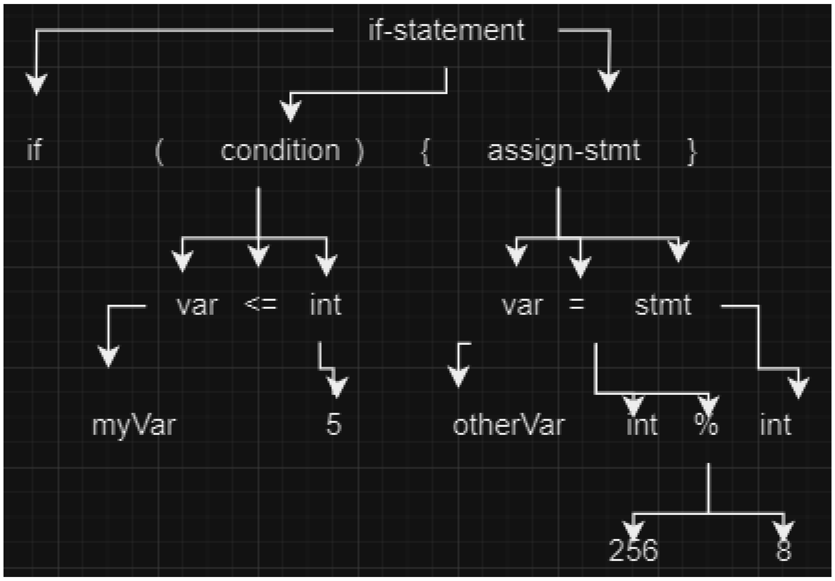


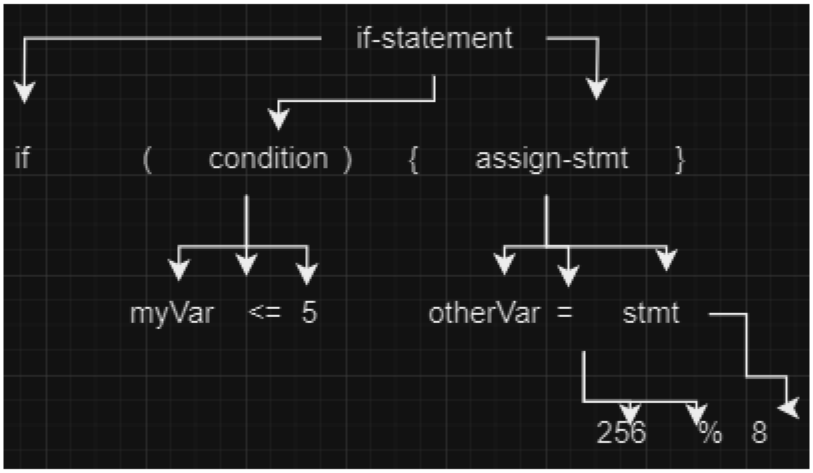
- (4 + 5) + 16 + (3 * 9) – abstract syntax tree

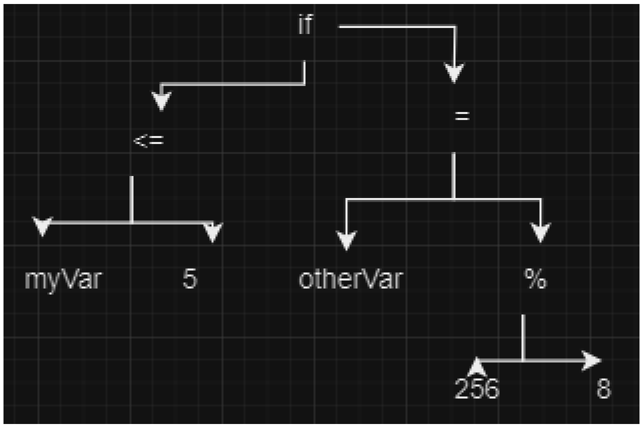


8. [20 points] Using the grammar and statement below, draw a parse tree, abstract syntax tree, and all intermediate steps for converting the parse tree to the abstract syntax tree.
   - if (myVar <= 5) { otherVar = 256 % 8 }

```
if-statement → if ( condition ) '{' assign-stmt '}'
condition → var == | <= | >= | < | > int
assign-stmt → var = stmt
stmt → int [+\-*\/%^] int
var → [a-zA-Z]+
int → [0-9]+
```






if
<=
=
myVar
5
otherVar
%
256
8