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]*
     {"-99*10^6", "0", "-0", "-043", "-43", "54*10^43243443434343434356", "3*10^"}

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

     {"HI and BYE", "YO and B", "T and Y", "T and …", "HELLO and …"}

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.

     - _\w*[0-9]

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

     - ((\(\d{3}\))|\d{3}-)\d{3}-\d{4}

   - The VCU V Number

     - V[0-9]{8}

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 }

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

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

4. [10 points] Using the grammar below, provide a leftmost derivation for the following statements:

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

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

if (condition) { assign-stmt } -> if (var <= int) { assign-stmt } -> if (myVar <= int) { assign-stmt } -> if (myVar <= 5) { assign-stmt } -> if (myVar <= 5) { var = stmt } -> if (myVar <= 5) { otherVar = stmt } -> if (myVar <= 5) { otherVar = int % int } -> if (myVar <= 5) { otherVar = 256 % int } -> **if (myVar <= 5) { otherVar = 256 % 8 }**

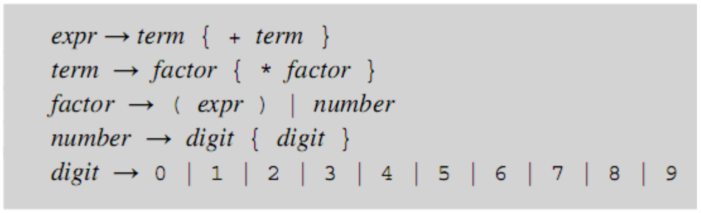
5. [10 points] Using the grammar below, how many legal sentences are there? Why is that? Suppose white space was completely ignored in the grammar so that sentences could be written as “thecompetitorseesawin.” Can this grammar still be parsed? Explain

```
1) sentence → noun-phrase verb-phrase
2) noun-phrase → article noun
3) article → a | and | the
4) noun → girl | competitor | win | dog | comp
5) verb-phrase → verb noun-phrase
6) verb → sees | permits | objects
```

Sentence can be split into noun-phrase and verb-phrase components. Noun-phrase can be split into an article and a noun, while a verb-phrase can be split into a verb and a noun-phrase and its components. So far, a sentence can be expanded to article noun verb article noun. There are 3 articles, 5 nouns, and 3 verbs. The possible combinations can be represented by 3*5*3 = 45 possible syntactically legal sentences as defined by the grammar above. Yes, the grammar can still be parsed by working backwards from the input. The first n characters can be read and matched with a list of valid terminals. Once a match is found, the parser can mark it as an article noun or verb. Then given the generic tokens in the input string, the parser can then determine if the structure is valid according to the given grammar rules based on the order of the tokens in the input string. White space can serve the purpose of acting as a delimiter and increasing readability and writability, but it is not necessary.

6. [20 points] Add the following four 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,

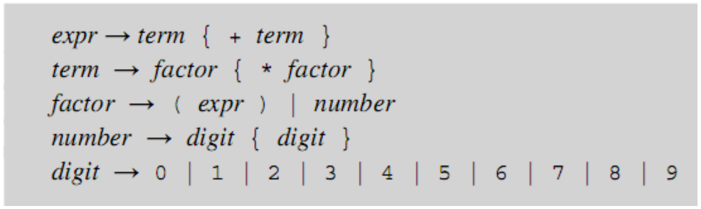
- division,
- exponents



```
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}
term - > expon { * expon } | { / expon }
expon -> factor { ^ factor }
factor -> ( expr ) | 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:
   - (8 * 3 * (4 + 5)) – parse tree
   - (4 + 5) + 16 + (3 * 9) – abstract syntax tree



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

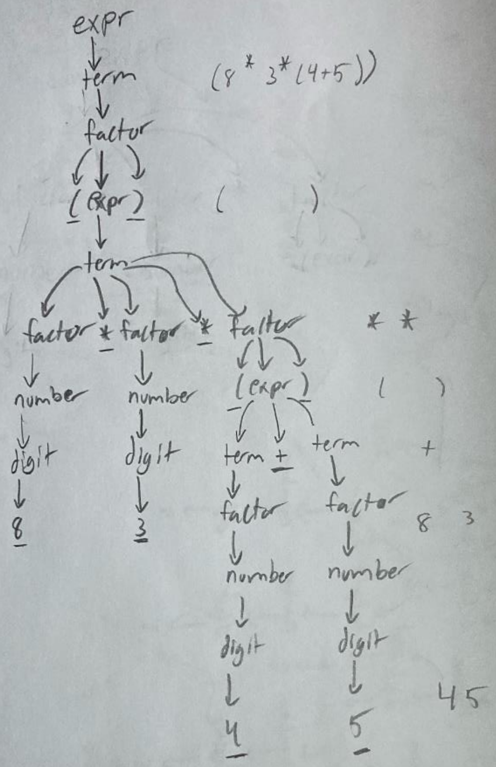
The pictures for problems 7 and 8 are inserted after problem 8.

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 }

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

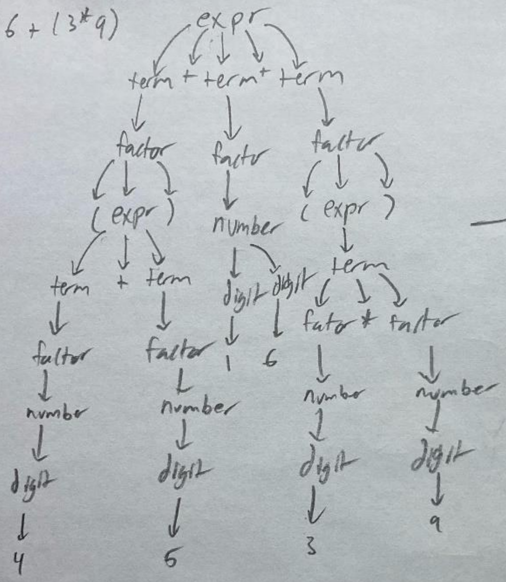
7)

a) (8*3*(4+5))

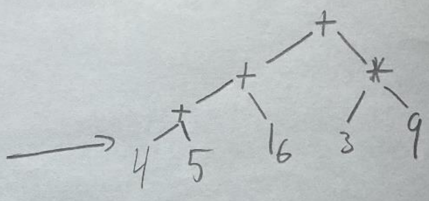


expr
term
(8*3*(4+5))
factor
(expr)
( )
term
factor * factor * factor
* *
number
number
(expr)
( )
digit
digit
term + term
+
8
3
factor
factor
8 3
number
number
digit
digit
4 5
4
5

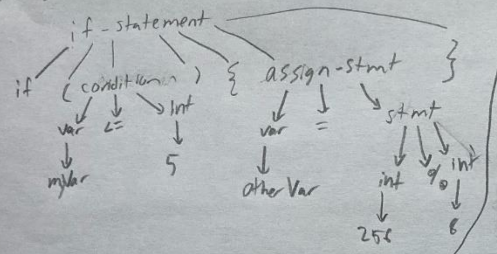
,) (4+5) + 16 + (3*9)


expr
term + term + term
factor factor factor
(expr) number (expr)
term + term digit digit term
factor factor 1 6 factor * factor
number number number number
digit digit digit digit
4 5 3 9

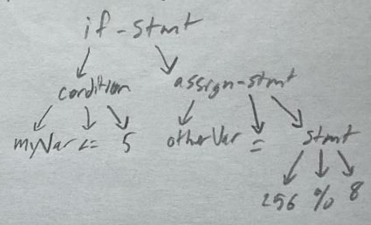
AST


+
+ *
+ 16 3 9
4 5

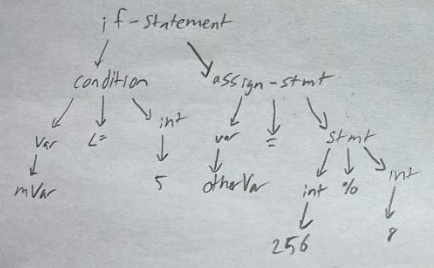
8) if(myVar <= 5) { otherVar = 256 % 8 }



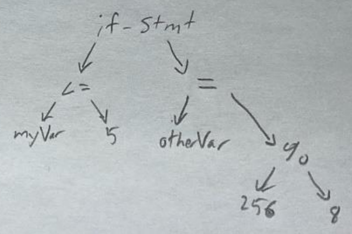
Remove all single-successor nodes



remove all structure defining symbols



make operator tokens nonterminal nodes


if-stmt
<=
=
myVar
5
otherVar
%
256
8