Creating the lexical and syntax analyzer for a programming language that will be defined in this problem. This language will be able to create variables, assign them value, calculate basic mathematic operations and relational operations for integers of different types, as well as variables that can be either.

This program should be able to not only recognize the following operations but have a proper evaluation order that conforms to the real-life principles of mathematics for in order operations:

a. Addition
b. Subtraction
c. Multiplication
d. Division
e. Module
f. Less than
g. Greater than
h. Less than Equal To
i. Greater than Equal To
j. Equal To
k. Not Equal To
l. Assignment
m. (There must also be a way to break precedence, this is usually done with the use of parenthesis)

Each program should have a clear beginning and end as well as a way to separate multiple statements.

For integer literals you must be able to specify whether in memory this value should be saved as 1 byte, 2 bytes, 4 bytes or 8 bytes.

Variables should be able to be declared but on in separate lines from the assignment of value.

Variable names can only be 6 – 8 letters, cannot contain numbers, but may contain underscores.

Language must be able to handle keywords to allow for loops, data type declarations, and selection statements.

This question is assignment is 8-fold:

a. (15 Points) Define the rules for recognizing all lexemes as their proper token, and clearly define integer token codes for each token required for this language
    - Should have Regular Grammar, Regular Expression, or Finite Automat defined

b. (15 Points) Define production rules for implementing the mathematical syntax of operators and operands, loops, variable declaration, selection statements
    - Enforce a non PEMDAS (BODMAS) order of operation, must have at least 6 levels of precedence
    - Keywords cannot use the words while, for, do, if, int, short, long
        i. Keywords should be unique, if others share your same words, you may lose more points than this problem is worth
    - You must clearly state the structure of your language with production rules

c. (10 points) Show whether every rule set in your language conforms to the standard of an LL Grammar.

d. (5 points) Make sure it is not ambiguous grammar

e. (15 points) Write a program that process all lexemes in a file by recognizing all tokens in a file, and produces a list of those tokens in order
    - If a group of characters is not defined in your language your program should print an error message stating what went wrong and terminate (stop running)
    - This program should be written in an Object-Oriented fashion
    - This program should have comments to describe each method that is defined

f. (10 points) Write a program or an extension to the above program that determines if the tokens conform to the correct syntax.

g. (10 points) Create 4 test files that have different names where each should have 30 or more lexemes that can be converted into tokens
    - 1 with a at least 5 lexical errors based on the rules you defined
        i. Detail each error and say why it doesn't work
    - 1 with at least 5 syntax errors based on the rules you defined
        i. Detail each error and say why it doesn't work
    - 2 with no errors at all based on the language you created

h. (20 points) Create a LR (1) parse table for your language. And show the trace of 4 code samples. Each must have 6 or more tokens.
    - Table must be provided, and the rules must be listed
    - 2 code samples must have errors
    - Show were these samples fail and pass the test