Algorithms used in the JAVACC

1. Finite Automata
   A finite Automata, a plural form of "automation", is considerable powerful in computer programs and analyzing sequential logic.
   This comes in handy when it comes to determining if an arbitrary string that we are going to give is already stated in the language.
   Given we are given a regular expression that describes the number of tokens in our programming source code, thereafter JavaCC acts on creating token managers and will turn them into finite automata. So that the token managers can utilize that created finite automata to identify the various tokens which appear in our source code.

2. Deterministic Finite Automata
   DFA, an abbreviation for Deterministic Finite Automata, will set up a state name for each. Now we have a diagram, Figure 1, as for the DFA.
   As mentioned, DFA has a finite number of states represented as a Set of states Q in the diagram. Each leaving arrows are labeled in characters from the input of the DFA, which are Input I {a, b} in this case.
   Since this example examines on only two characters, a and b, each state has the same number of arrows leaving it, a and b. S0, in other words, is the start state where its examination begin, and S2 and S3, indicated as concentric circles are accept states, otherwise reject states.
   The set of inputs or regular expression for that matter has to comply to go through this flow of examination by taking an arrow with matching the next character in the input. The decision on whether or not a string of input will be accepted is only determined by the state where the DFA eventually ended up in. This operation at any time is completely determined. This is where DFA gets its name.

   Take the diagram as an example.

To begin with, we have a string of input I, which are going to be fed into DFA. Then the initial character "a" takes a route from S0 to S1,therefore we are in S1 now. The next character "b" takes its route for S3. Finally the final character "b" takes its route back to S3 recursively. This kind of route taking is happening whenever as long as the next character is "b" at S3, otherwise the next step will be carried out to S4 where the reject state is stated.
The set of strings in use of regular expression that will be accepted, i.e. that will end up in either state S2 and S3, are "(a | b) a" and "(a | b) b b *", respectively.
Therefore, the language defined by this DFA for an accepted state is the following combined expression by the union operator:
(a | b) a | (a | b) b b *

Figure 1

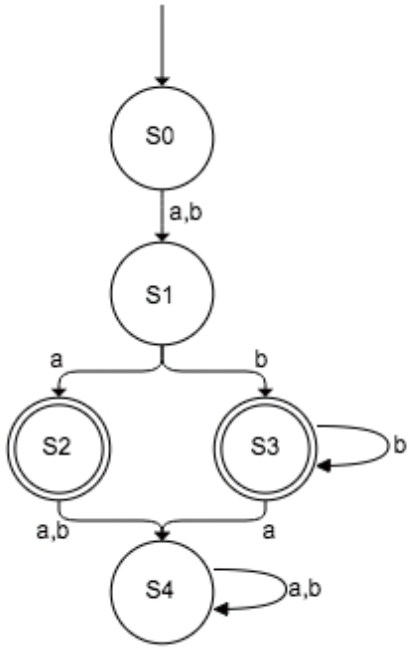


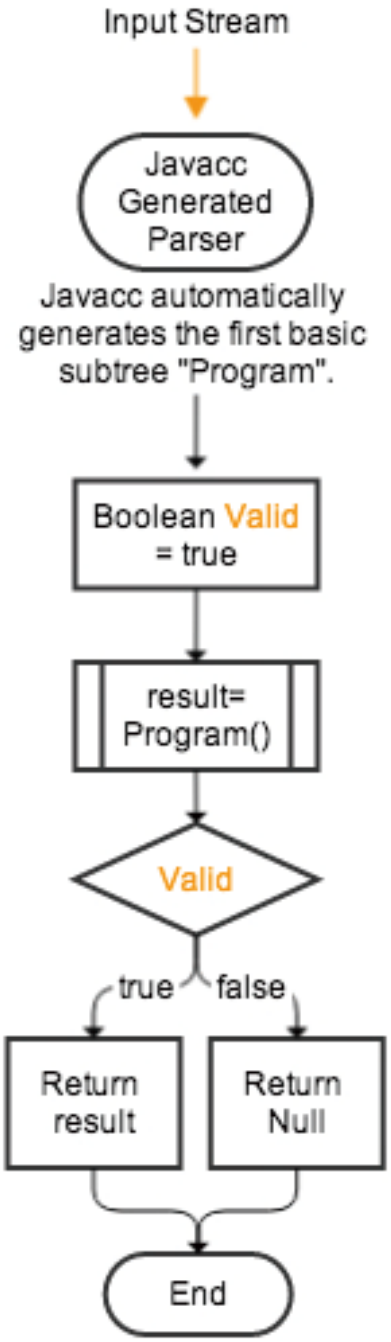
*Figure 1, being made up of following conditions.

- Set of states Q = {S0,S1,S3,S4}
- Input I = {a,b}
- Initial state S0 ∈ Q
- Set of accept states A = {S2,S3} ⊆ Q

Algorithms used in the MARVEL

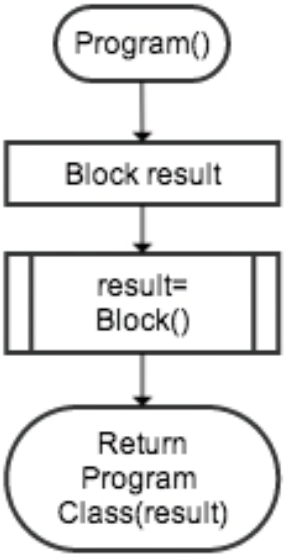
1. Main stream.

User Input will be fed into a Parser that JAVACC automatically generated. Parser Class contains global variable namely “Valid” which accounts for the decision making whether its parsing have succeeded properly as a consequence.



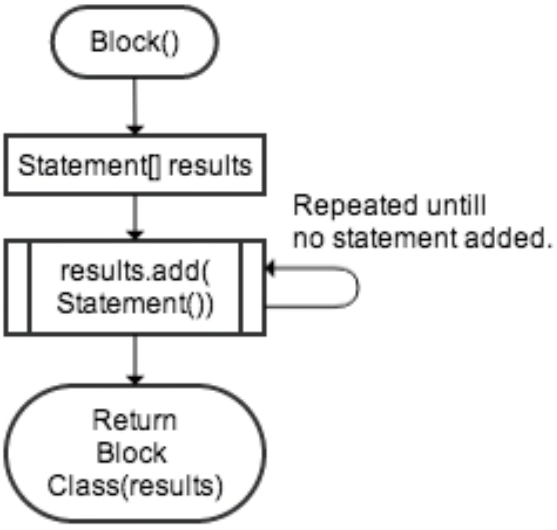
2. Subprogram (Program)

Being first called by the main method of Parser Class. This entity acts as an uppermost level of the sub-tree the program maintains.



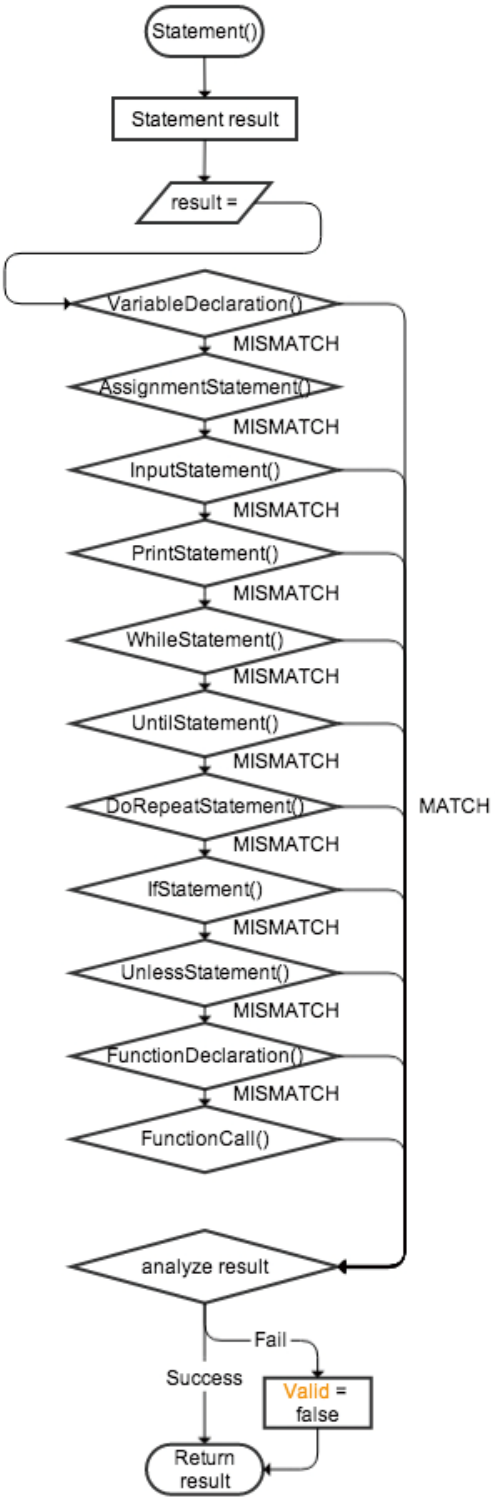
3. Subprogram (Block)

This level of sub-tree “Block” organizes the sequential statements which will be accumulated in a array “results” and returned at the end.



4. Subprogram (Statement)

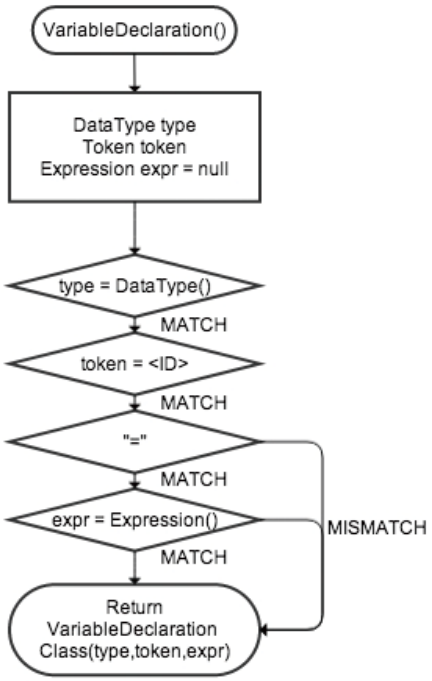
This level of sub-tree “Statement” maintains each type of statements and holds the result of whatever the each type meets its own condition. After that, analyzes that holding result for the intrinsic compatibility. In case the analysis fails , sets the variable “Valid” to false.



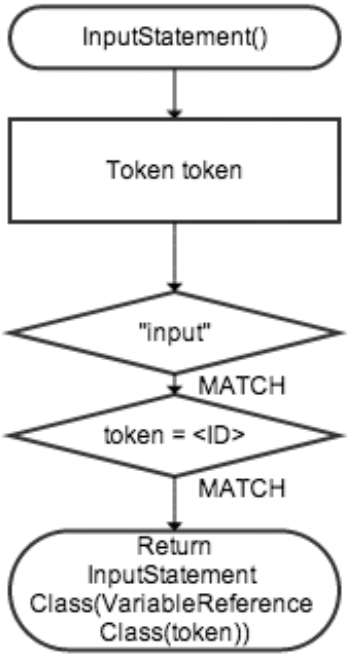
5. Subprograms extending Statement

These are sub-typed Statements which will be called by the Statement method.

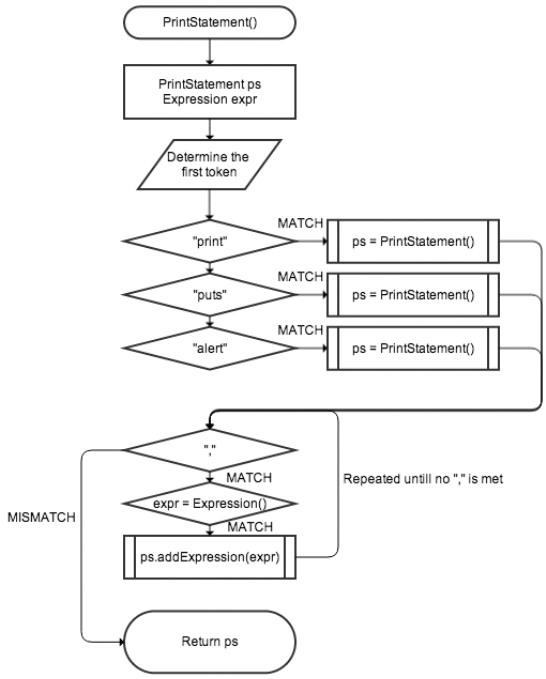
- VariableDeclaration
  This VariableDeclaration program captures a variable declaration statement, and also allowed to have initial assignment to it.



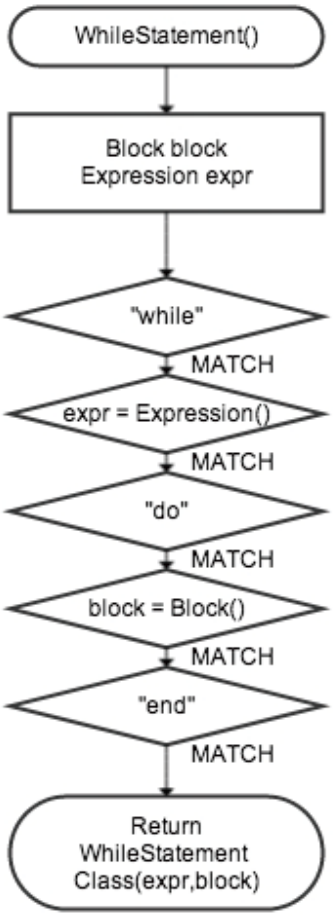
- InputStatement
  This ImputStatement take input identifier to have value in it.



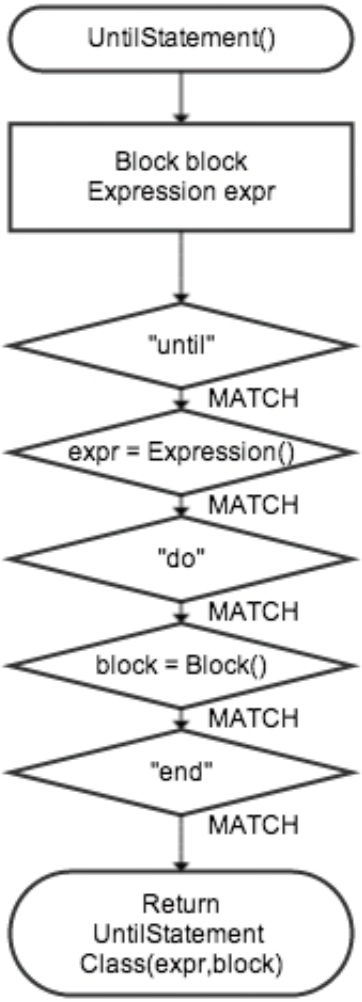
- PrintStatement
  This PrintStatement allow to take three types of syntax , “print” ,” puts” ,and “alert” and also allow to have multiple following expressions separated by “,” .



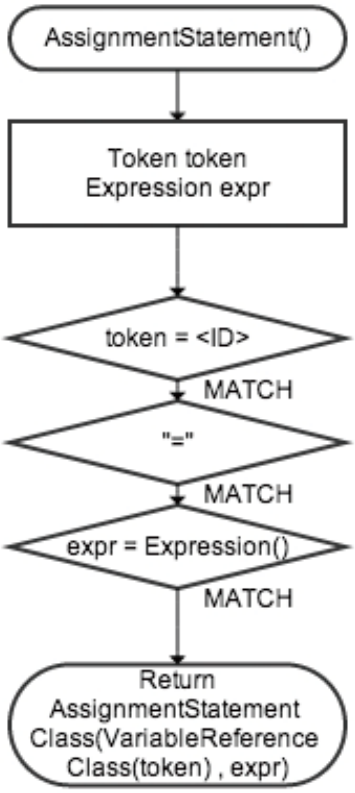
- WhileStatement
  This WhileStatement ,will have an expression in between “while” and “do” , and condition after “do” ,have to terminate with “end” .



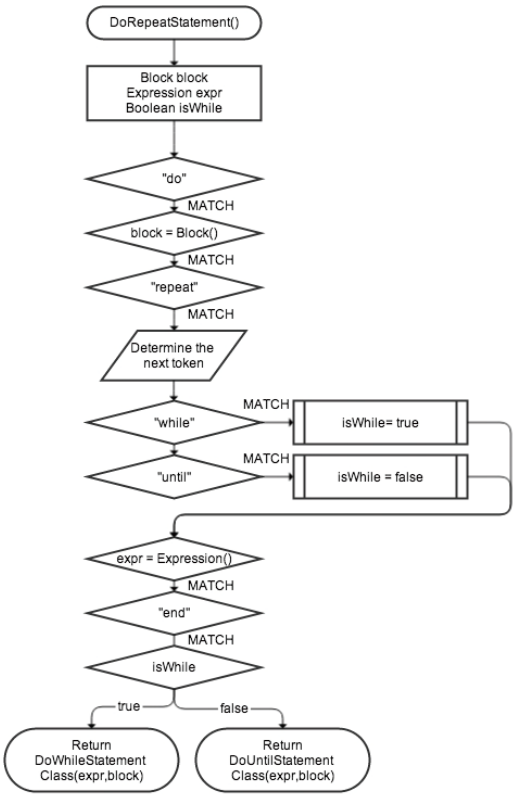
- UntilStatement
  UntilStatement,basically the syntax of which is the same as WhileStatement.



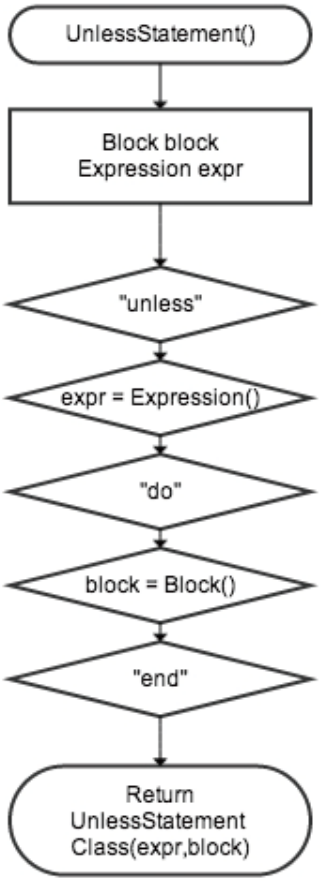
- AssignmentStatement
  This AssignmentStatement will take the value followed by “=” to assign to previously stated variable.



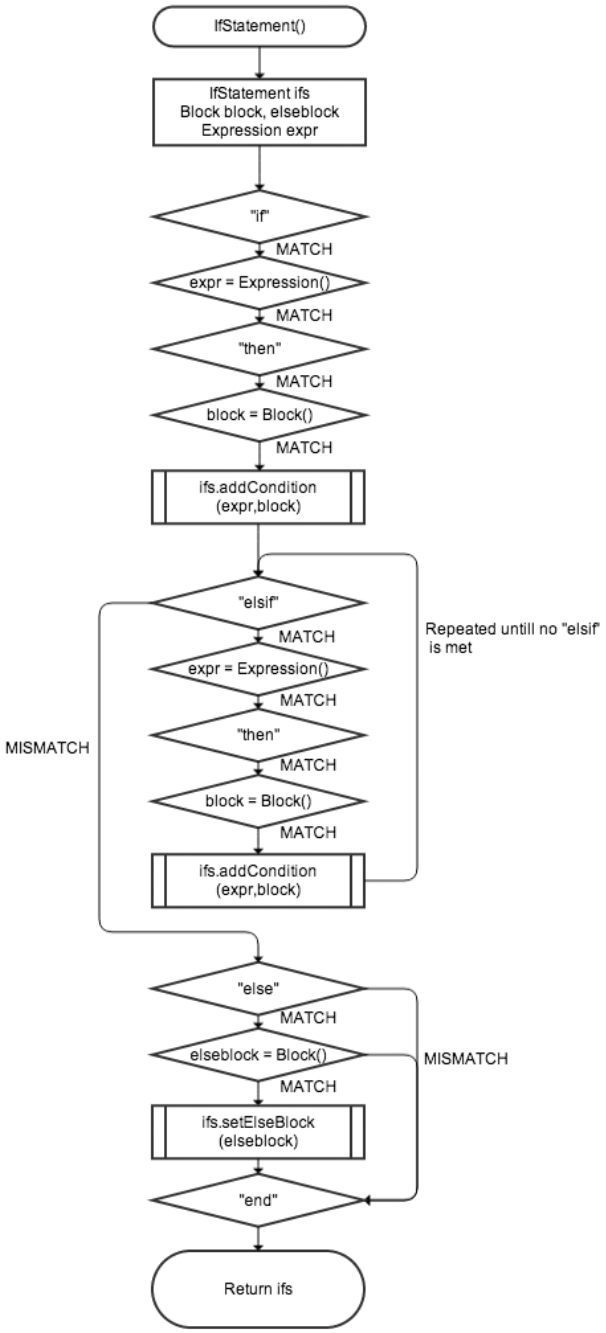
- DoRepeatStatement
  This DoRepeatStatement will allow to have two deferent types of syntax in it. The decision will be made on the occurrence of either token “while” or “until”. The boolean variable “isWhile” accounts for it.



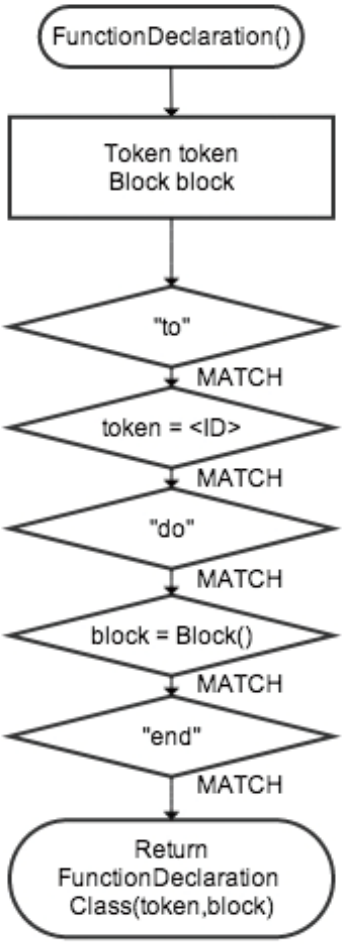
- UnlessStatement
  This UnlessStatement evaluates the condition on counter standpoint.



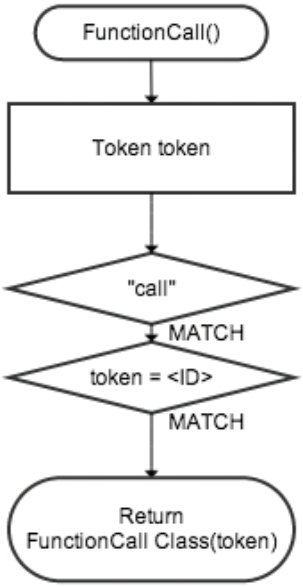
- IfStatement
  This IfStatement will allow for three types of syntax , “if” ,” elsif” ,and “else” . The bracket for “elsif” can be repeated as many as possible. However you are only allowed to have “else” once to state.



- FunctionDeclaration
  This FunctionDeclaration will take an identifier for the function call.

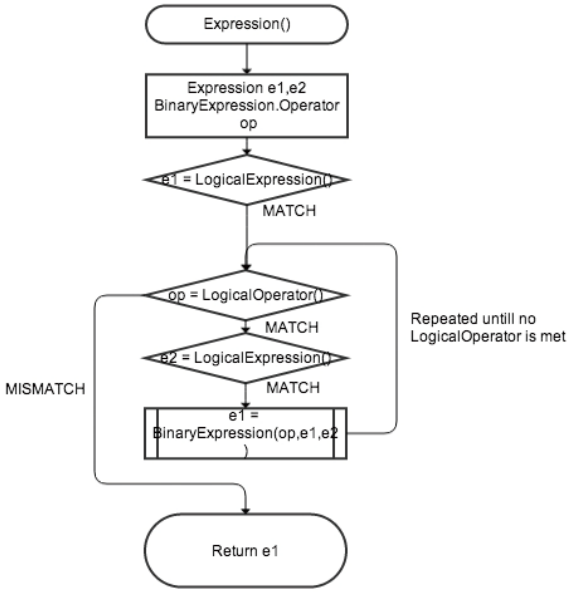


- FunctionCall
  This FunctionCall accommodates the function name to be called followed by a syntax “call”.



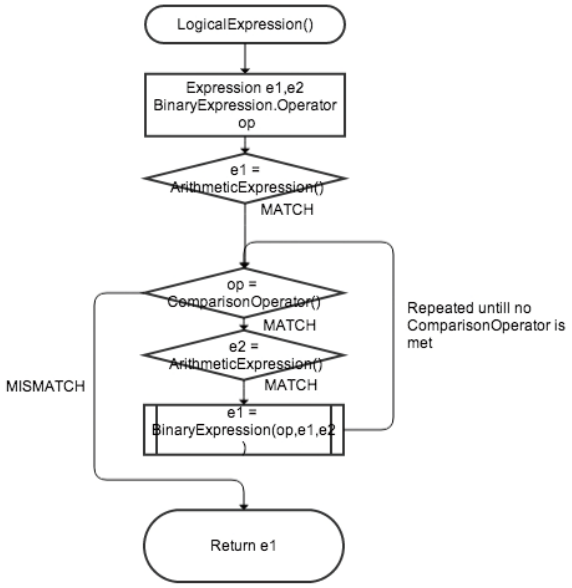
6. Subprogram (Expression)

This level of tree “Expression” accommodates LogicalExpression ,and in case there exists a LogicalOperator in between two of LogicalExpression, it will be allowed to contain to evaluate them.



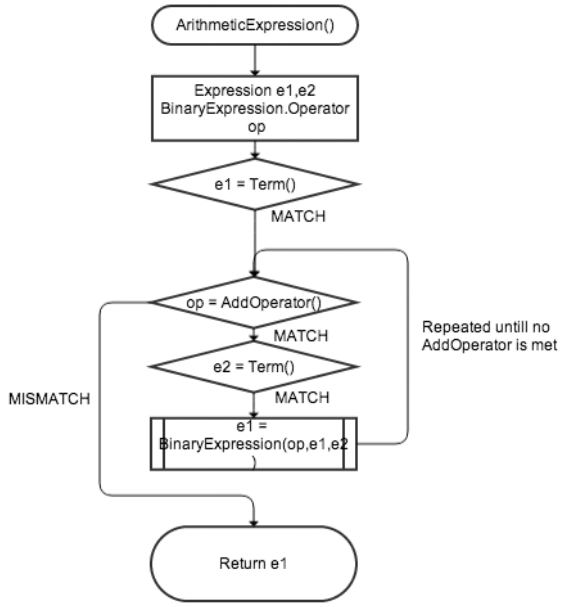
7. Subprogram (LogicalExpression)

This level of tree “LogicalExpression” accommodates the ArithmeticExpression ,and which can be partitioned by ComparisonOperator to compare a state of each.



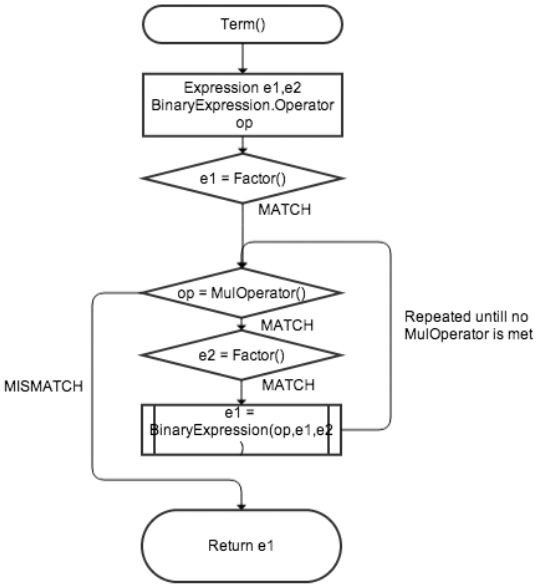
8. Subprogram (ArithmeticExpression)

This level of tree “ArithmeticExpression” accommodates the Term ,and which can be partitioned by the AddOperator to comply with an order of arithmetic operation, times/divide first.



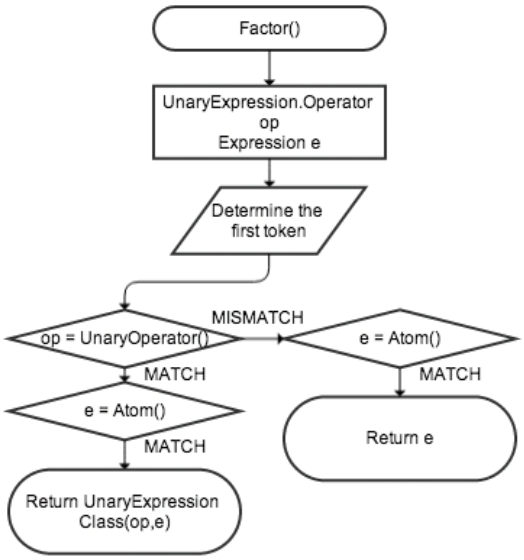
9. Subprogram (Term)

This level of tree “Term” accommodates the Factor,and which can be partitioned by the MulOperator.



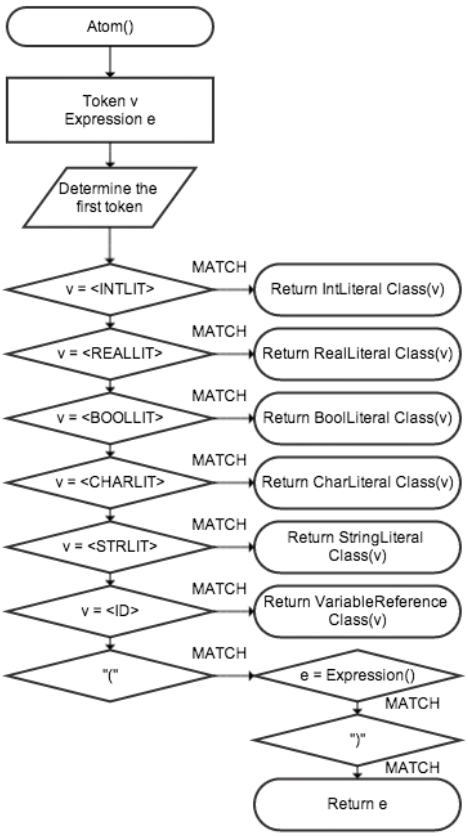
10. Subprogram (Factor)

This level of tree “Factor” accommodates either the UnaryOperator and the Atom ,or simply only one Atom.



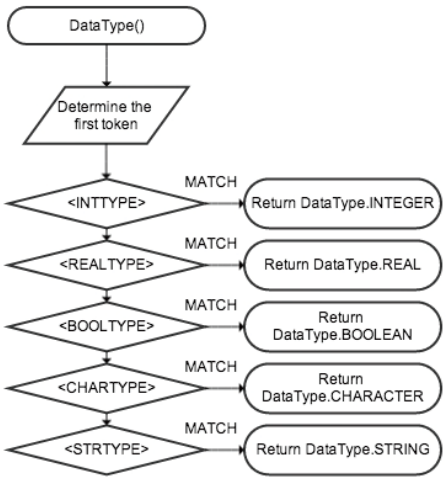
11. Subprogram (Atom)

This level of tree “Atom” accommodates the lowest definitions ,and returns an according value taken out of the token.

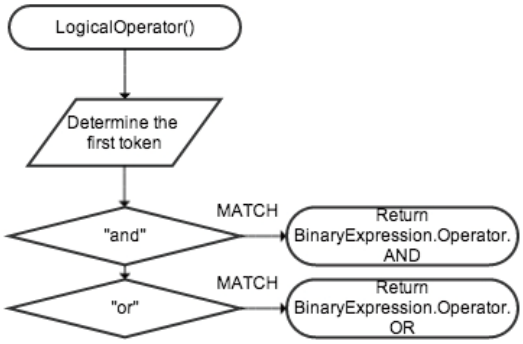


12. Subprogram (DataType)

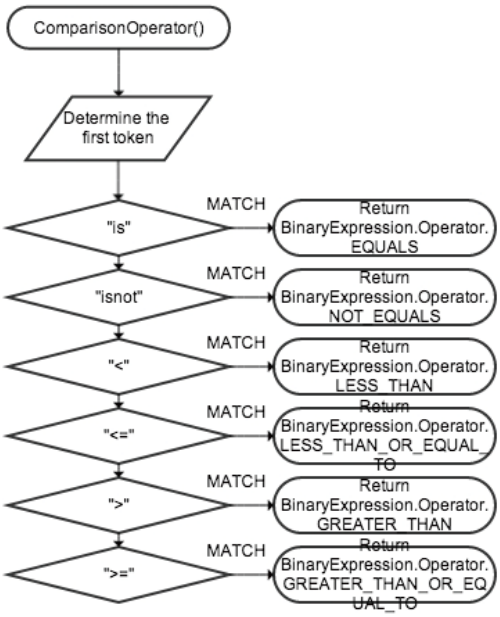
This level of tree “DataType” accommodates the lowest definitions for each type of token.



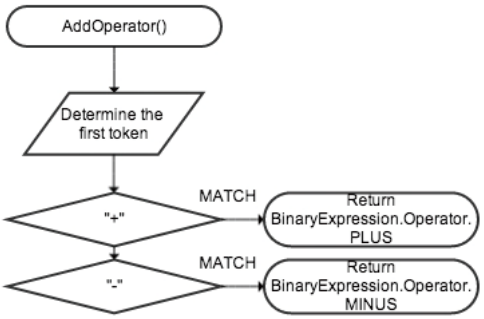
13. Subprogram (Operators)
    - LogicalOperator
      This LogicalOperator returns its atom for a logical operator.



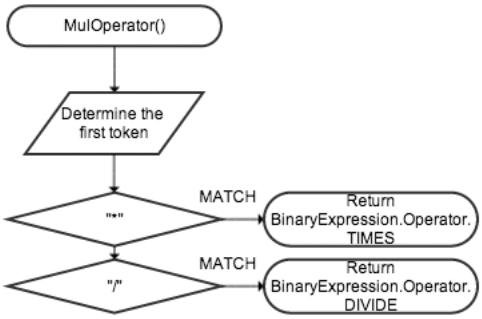
    - ComparisonOperator
      This ComparisonOperator returns its atom for a comparison operator.



    - AddOperator
      This AddOperator returns its atom for an add operator.



    - MulOperator
      This MulOperator returns its atom for a multiple operator.



    - UnaryOperator
      This UnaryOperator returns its atom for an unary operator.