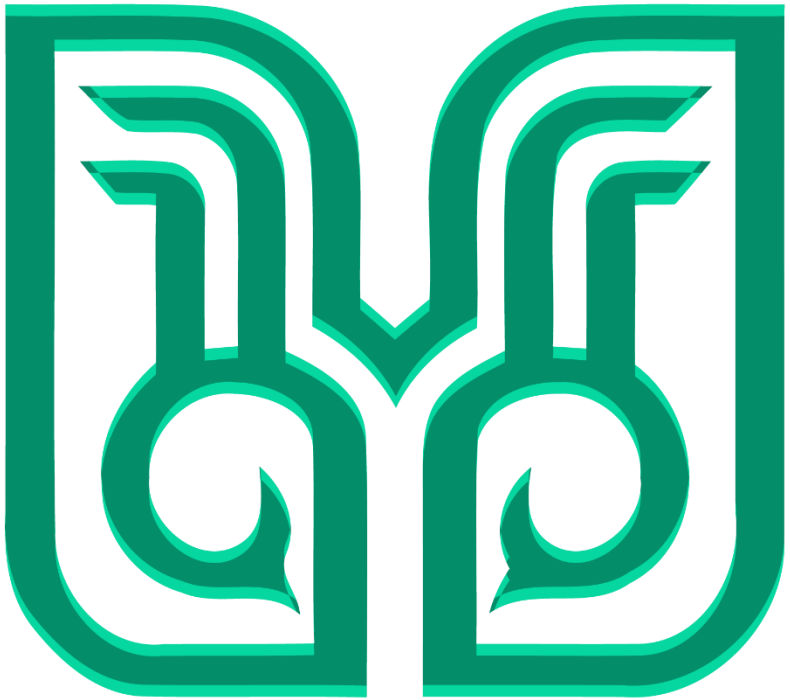


Compiler Design Project

Part I

Lexical Analyzer

(Scanner)

Presented By:

Ardalan Soltanzadeh

400407014

Sepehr Bazmi

400405019

Dr. Fatemeh Yousefi

Lexical Analyzer

A lexical analyzer, also known as a scanner or tokenizer, is a component of a compiler that breaks down the source code into a sequence of tokens. Its purpose is to recognize and extract the smallest meaningful units of the programming language, such as keywords, identifiers, literals, and operators.

The lexical analyzer works by scanning the source code character by character. It uses a set of rules, defined by regular expressions or patterns, to identify and classify different tokens. These rules are typically specified through a combination of regular expressions and finite automata.

It also known as a scanner or tokenizer, is a component of a compiler that breaks down the source code into a sequence of tokens. Its purpose is to recognize and extract the smallest meaningful units of the programming language, such as keywords, identifiers, literals, and operators.

The tokens in this Project are :

1- Identifier

2- Number (int , float)

3- Operator

4- Keyword

5- String

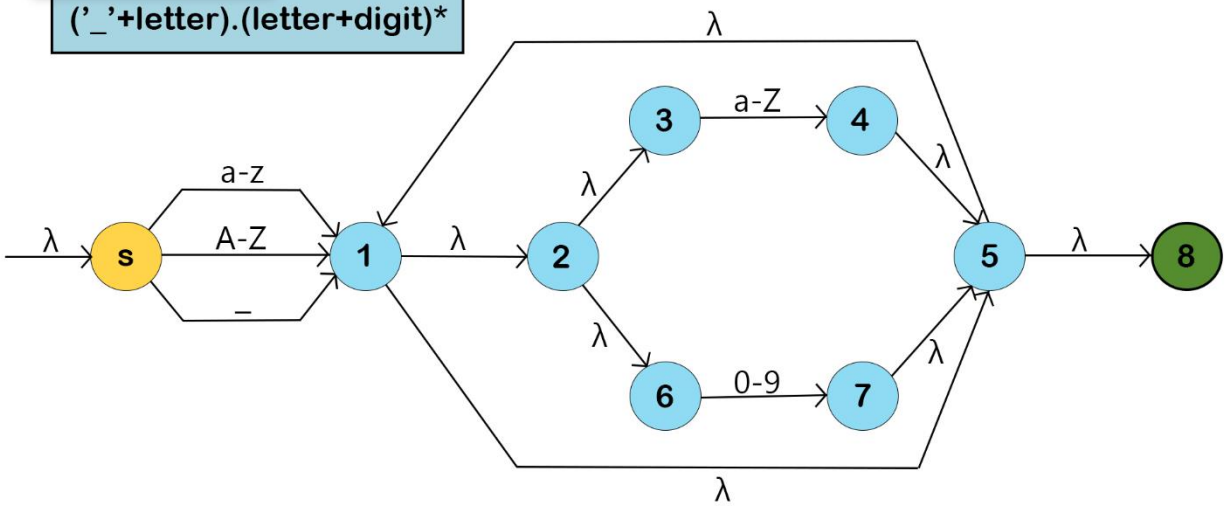
6- Delimiter

7- Comment

First , we design an NFA for each token mentioned above ,then we combine every essential NFA to one whole NFA for the language, lastly we use the epsilon-closure algorithm to convert the NFA to a DFA.

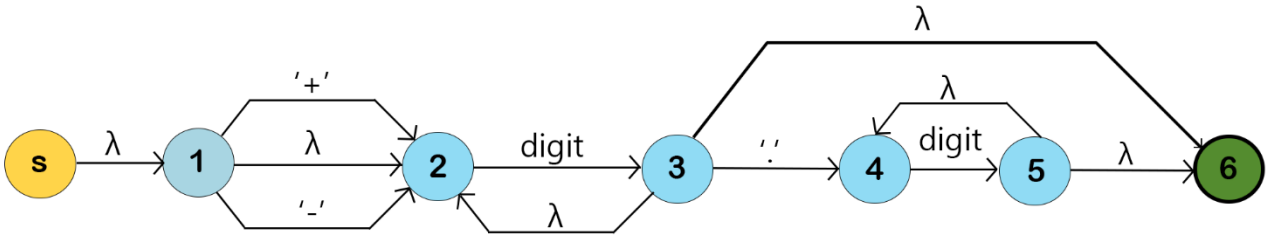
identifier

('_'+letter).(letter+digit)*

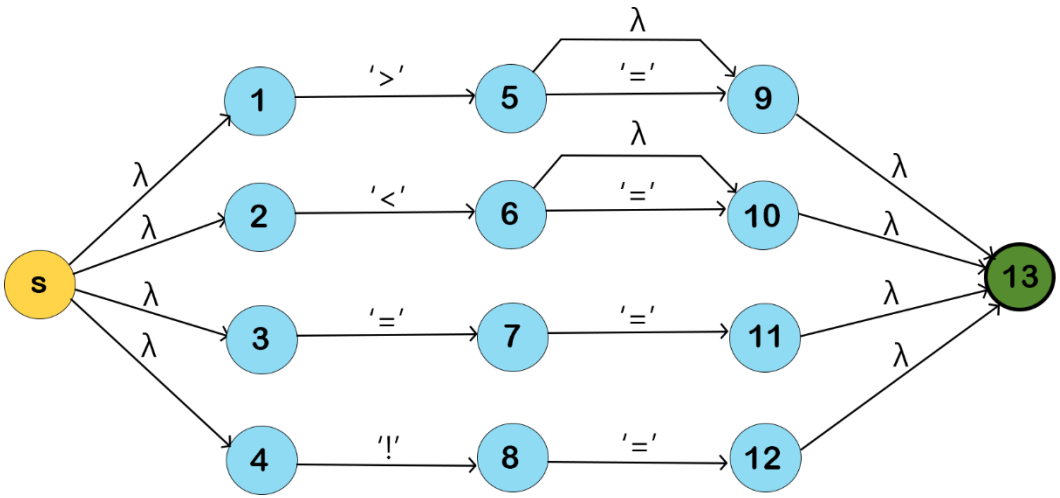


number

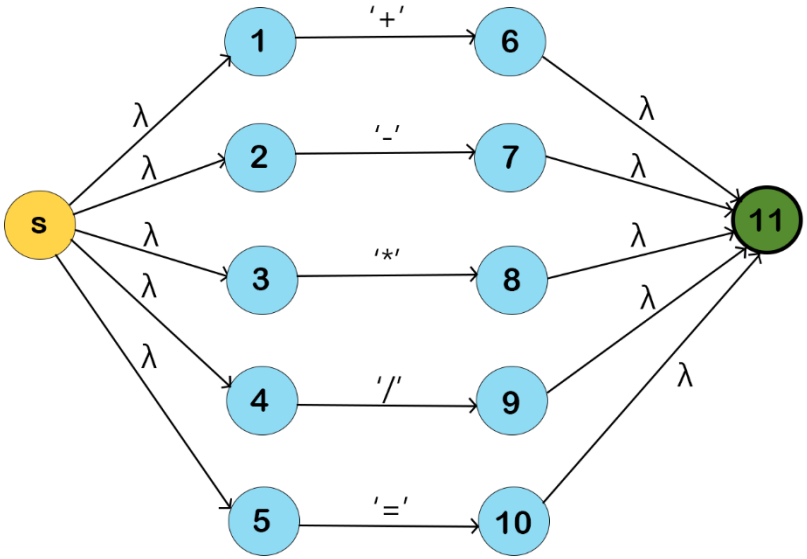
digit⁺('.'.digit⁺ + λ)



operator logical

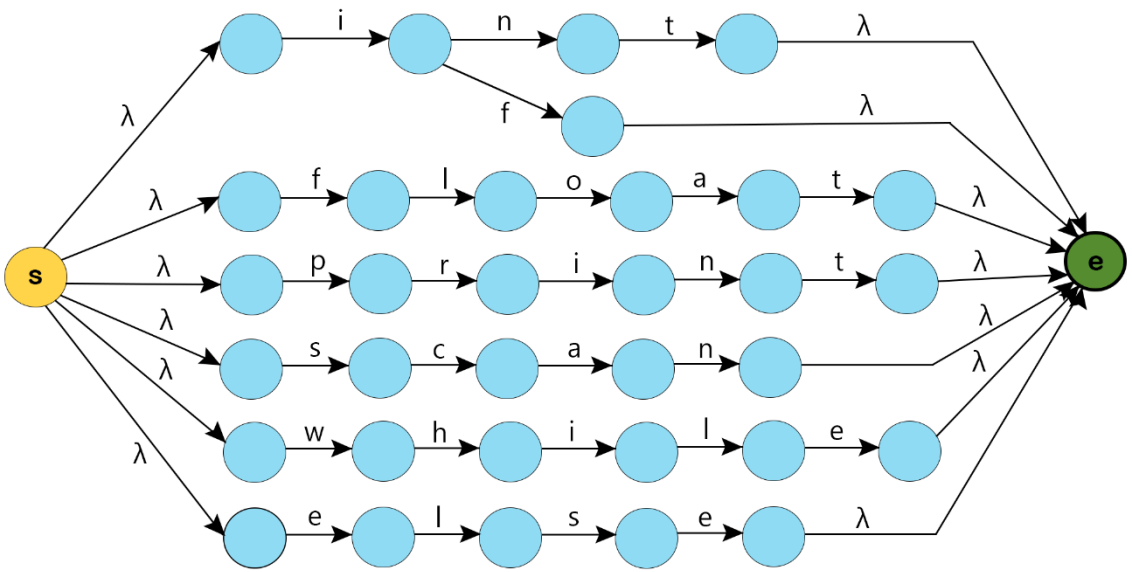


operator mathematical



keywords

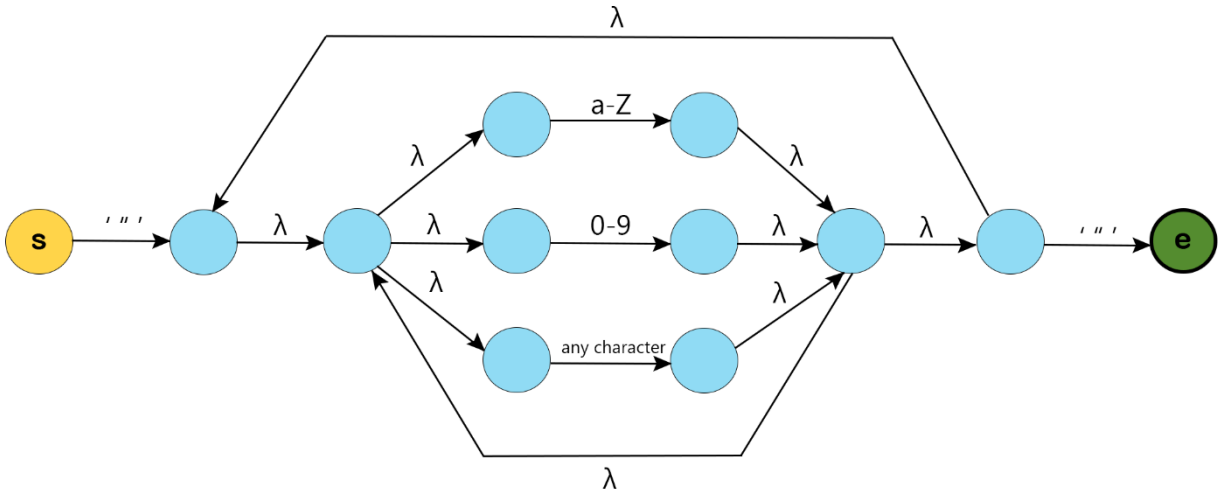
{int,float,scan,print,if,else,while}



We wont include the Keyword NFA in the Main NFA for the reason mentioned below :
The program identifies all the keywords as Identifiers then, if the lexeme is a keyword, It would return it as a keyword, if not it would be returned as an identifier.

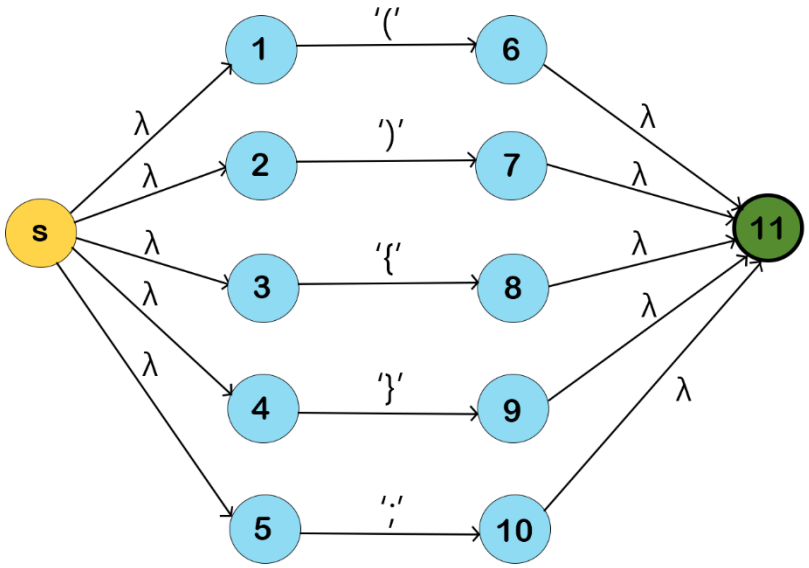
string

‘ “ ’ . (letter+digit + anycharacter)* . ‘ ” ’



delimiter

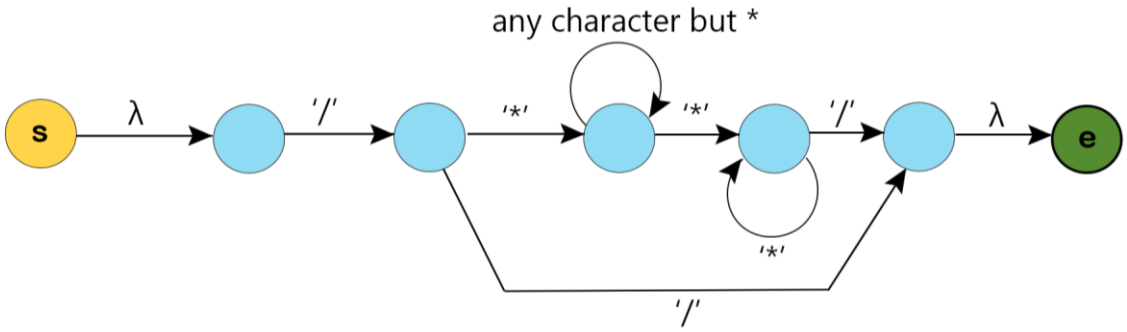
{ ‘(’ , ‘)’ , ‘{’ , ‘}’ ‘;’ }



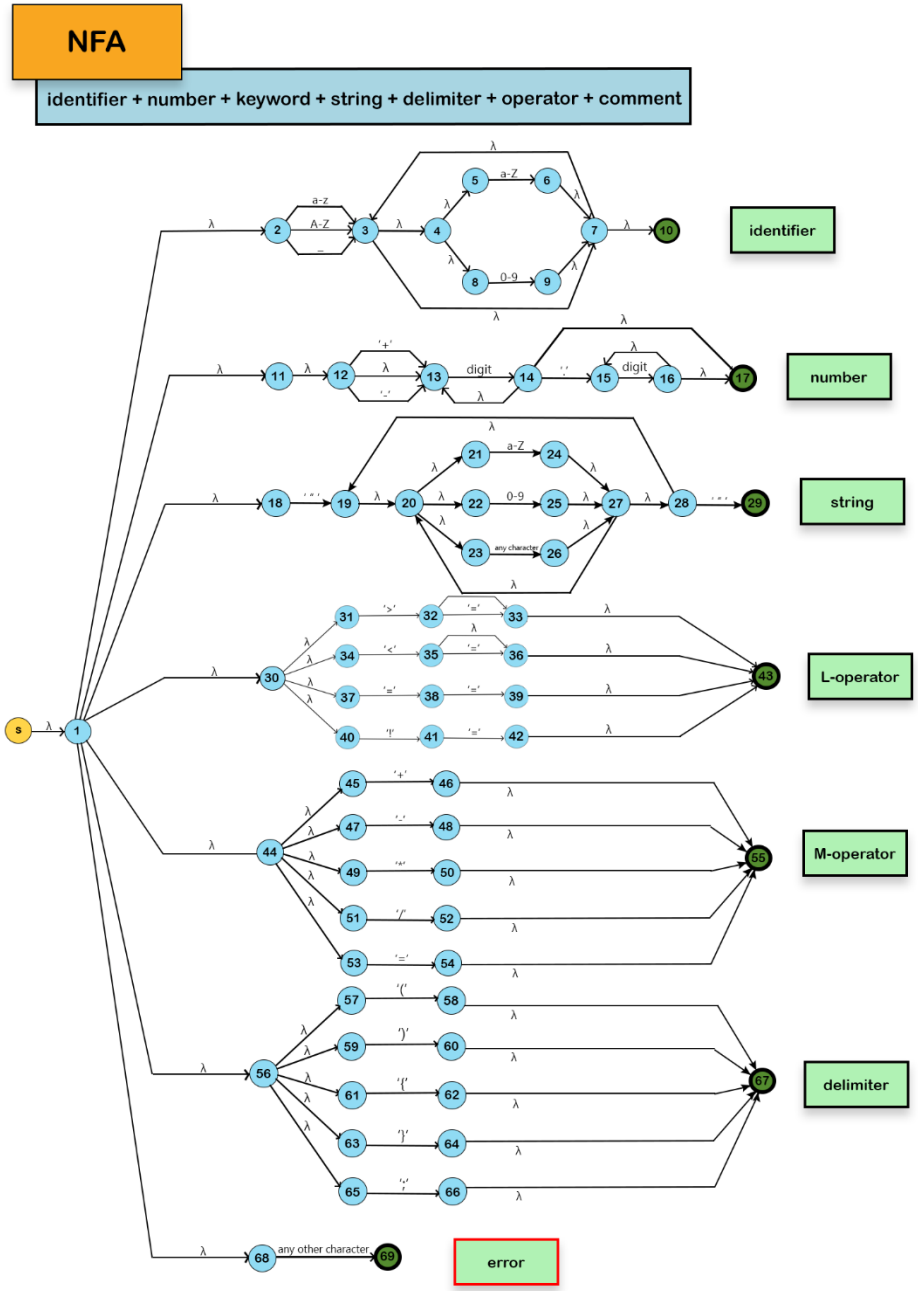
comment

// for one line, *…*\ for n line

To reduce the lookahead of the main NFA or DFA comments are excluded from the main NFA or DFA.



Now that we created an NFA for each token, we combine them to create the Main NFA of the scanner :



Note that,
Keyword and comment are not present in the Main NFA, the reasoning behind this decision Is mentioned above where their standalone NFA were created.

Now that we have a complete NFA to work with It is more efficient to Convert it to a DFA Before starting it's programming
Therefore we will use an algorithm knows as The epsilon-closure to create a DFA with the main NFA.

λ-closure

converting the NFA to a DFA using epsilon-closure algorithm by calaculating which states can go to which states without any valid input, in other words using only epsilon(lambda):

({1})

A

{ 1 , 2 , 11 ,12 , 13 , 18 , 30 , 31 , 34 , 37 , 40 , 44 ,45 , 47 , 49 , 51 , 53 , 56 , 57 , 59 , 61 , 63 , 65 , 68 }

move(A,letter) = {3} move(A,!) = {41} move(A,)) = {60,67}=S F
move(A,digit) = {14} move(A,+) = {46,55}=N F move(A,{) = {62,67}=T F
move(A,") = {19} move(A,-) = {48,55}=O F move(A,}) = {64,67}=U F
move(A,>} = {32} move(A,*} = {50,55}=P F move(A,;} = {66,67}=V F
move(A,<} = {35} move(A,/} = {52,55} =Q F move(A,any other character} = {69} F
move(A,=} = {38,54} move(A,(} = {58,67}=R F move(A,_) = {3}

({3})

B

{ 3 , 4 , 5 , 7 , 8 , 10 }

move(B,letter) = {3,6,7,9,10} FINAL
move(B,digit) = {3,7,9,10} FINAL
move(B,_) = {3,4,5,7,8,9,10} FINAL

({14})

C

{ 13 , 14 , 17 }

move(C,digit) = C FINAL
move(C,.) = {15}
move({15},digit) = {15,16,17} = L FINAL

({19})

D

{ 19 , 20 , 21 , 22 , 23 }

move(D,digit) = {19 , 25 , 27 , 28 }
move(D,letter) = {19 , 24, 27 , 28 }
move(D,any other character) = {19 , 24, 26 , 28 }

({19, 25, 27, 28}) | ({19, 25, 27, 28}) | ({19, 25, 27, 28})

E

{ 19 , 20 , 21 , 22 , 23 , 25 , 27, 28 }

move(E,letter) = {24,27,28,19,20}
move(E,digit) = {25,27,28,19,20}
move(E,any other character) = {26,27,28,19,20}
move(F,") = {29} FINAL
move(G,") = {29} FINAL
move(H,") = {29} FINAL

({32})

F

{ 32 , 33 , 43 }

move(F,=) = {33,43} = G FINAL

({35})

J

{ 35 , 36 , 43 } FINAL

move(J,=) = {36,43} = H FINAL

({38,54})

K

{ 38 , 54 , 55 } = FINAL

move(K,=) = {39 , 43 } = I FINAL

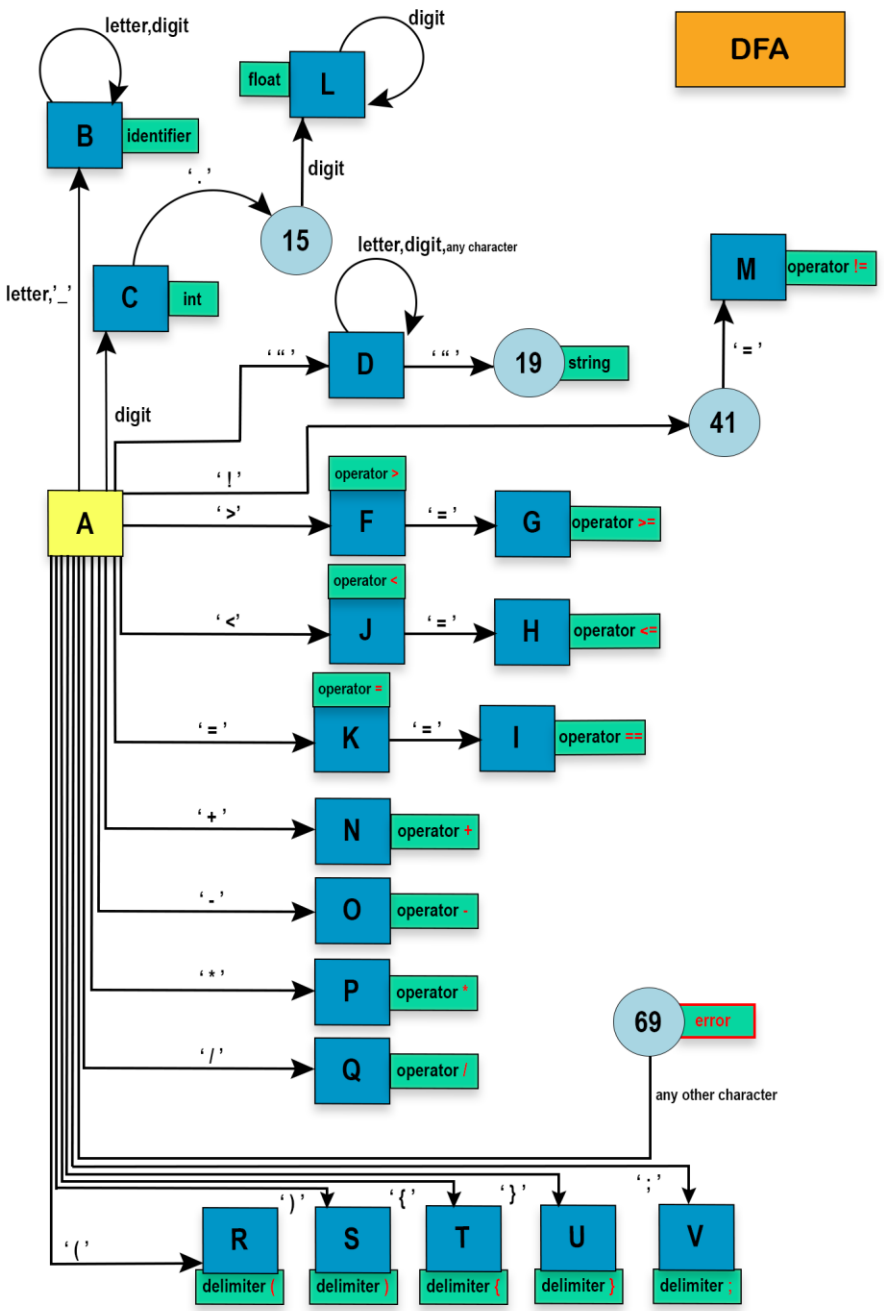
({41})

{41}

{ 41 }

move(41,=) = {42,43} = M FINAL

Finally we have enough information to create the main DFA :



As you see in the DFA on the left;
Each square is a combination of several states That where generated through the epsilon-closure process. You can determine which states are in these combinations by looking up the epsilon closure process in the previous page.
For example the state A in this DFA contains {1,2,11,12,13,18,30,31,34,37,40,44,45,47,49,51, 53,56,57,59,61,63,65,58 } from the main NFA.