**Regular Expression:**

Given below are the regular expressions for multiple lexeme identifier. In the end I combined all these expressions to form single regular expression.

* Relational & Assignment Operators: (=) **|** ( > ( = **|** lambda ) ) **|** ( < ( = **|** - **|** lambda ) ) **|** ( ~ = )
* Comments: ( # ) ( other )\* ( ‘\n’ )
* Character: ( ‘ ) ( Letter ) ( ’ )
* Strings: ( “ ) ( other )\* ( ” )
* Identifier: ( Letter ) ( Letter/Digit )\*
* Numerical Literals: ( + **|** - **|** lambda ) ( Digit )^ ( ( . ( Digit )^ ) **|** lambda ) ( ( ( + **|** - **|** lambda ) E ( Digit )^ ) **|** lambda )
* DFA relational expression: ( Relational & Assignment Operators ) **|** ( Comments ) **|** ( Character ) **|** ( Strings) **|** ( Identifier) **|** ( Numerical Literals)

**Note:** In regular expression the notation I have used are **|** for or. For and there is simply blank space. The last regular expression is for the DFA.

**Not Included in DFA/Regular Expression:**

Following lexemes are not included in both regular expression as well as in DFA:

* Arithematic Operators: **+**, **-**, **\***, **/**, **%**
* Relational Operator: **=**
* Block Start & End: **{**, **}**
* Special Characters: **, # : ;**

**Note:** Although > and < are single character lexeme but DFA will recognize them when it is dealing with <= and >=**.**

**DFA:**

**Starting State:** **q0**

**Finishing States:** **q16**-> where lexeme will be identified at last character. **q17**-> where lexeme will be identified one character after the last character. So we need to go back 1 character.

**Remaining State:**

**q1->** Deal with comments.

**q2->** Deal with strings.

**q4, q5->** Deal with single character.

**q3, q6, q7->** Deal with relational & assignment operator.

**q8->** Deal with identifier.

**q9, q10, q11, q12, q13, q14, q15->** Deal with numeric literals.