

# Compiler

## Lexical Analyzer Report

Team 24.  
20176959 김영권

### 1. Definition of tokens and their regex.

| Token name           | Regular expression                                   |
|----------------------|------------------------------------------------------|
| VTYPE                | int   char   boolean   String                        |
| SIGNED_INTEGER       | 0   (-1   ε)(DIGIT 1To9)(DIGIT)*                     |
| SINGLE_CHARACTER     | (‘)(LETTER   DIGIT   WHITESPACE)(‘)                  |
| BOOLEAN_STRING       | true   false                                         |
| LITERAL_STRING       | (“)(LETTER   DIGIT   WHITESPACE)*(“)                 |
| IDENTIFIER           | (_   LETTER)(LETTER   DIGIT   _)*                    |
| KEYWORD              | if   else   while   class   return   public   static |
| ARITHMETIC_OPERATOR  | +   -   *   /                                        |
| ASSIGNMENT_OPERATOR  | =                                                    |
| COMPARISION_OPERATOR | <   >   ==   !=   <=   >=                            |
| TERMINATING_SYMBOL   | ;                                                    |
| LEFT_PAREN           | (                                                    |
| RIGHT_PAREN          | )                                                    |
| LEFT_BRACE           | {                                                    |
| RIGHT_BRACE          | }                                                    |
| LEFT_BRACKET         | [                                                    |
| RIGHT_BRACKET        | ]                                                    |
| COMMA                | ,                                                    |
| WHITESPACE           | (\t   \n   BLANK)*                                   |

## 2. NFA and DFA tables for recognizing regular expressions.

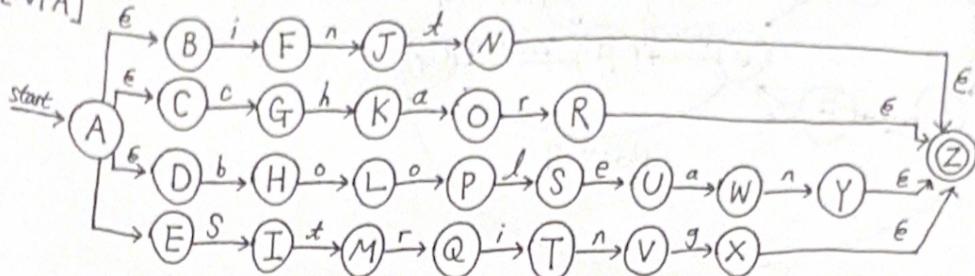
### 1) VTYPE (NFA and process for DFA)

Compiler Team 24. [20176959 김영권].

#### ① VTYPE (Variable type)

Regular Expression : int | char | boolean | String

[NFA]



$$T_0 = \epsilon\text{-closure}(A) = \{A, B, C, D, E\}$$

$$T_1 = \epsilon\text{-closure}(\delta(T_0, i)) = \epsilon\text{-closure}(F) = \{F\}$$

$$T_2 = \epsilon\text{-closure}(\delta(T_0, c)) = \epsilon\text{-closure}(G) = \{G\}$$

$$T_3 = \epsilon\text{-closure}(\delta(T_0, b)) = \epsilon\text{-closure}(H) = \{H\}$$

$$T_4 = \epsilon\text{-closure}(\delta(T_0, s)) = \epsilon\text{-closure}(I) = \{I\}$$

$$T_5 = \epsilon\text{-closure}(\delta(T_1, n)) = \epsilon\text{-closure}(J) = \{J\}$$

$$T_6 = \epsilon\text{-closure}(\delta(T_2, h)) = \epsilon\text{-closure}(K) = \{K\}$$

$$T_7 = \epsilon\text{-closure}(\delta(T_3, o)) = \epsilon\text{-closure}(L) = \{L\}$$

$$T_8 = \epsilon\text{-closure}(\delta(T_4, t)) = \epsilon\text{-closure}(M) = \{M\}$$

$$T_9 = \epsilon\text{-closure}(\delta(T_5, t)) = \epsilon\text{-closure}(N) = \{N\}$$

$$T_{10} = \epsilon\text{-closure}(\delta(T_6, a)) = \epsilon\text{-closure}(O) = \{O\}$$

$$T_{11} = \epsilon\text{-closure}(\delta(T_7, o)) = \epsilon\text{-closure}(P) = \{P\}$$

$$T_{12} = \epsilon\text{-closure}(\delta(T_8, l)) = \epsilon\text{-closure}(Q) = \{Q\}$$

$$T_{13} = \epsilon\text{-closure}(\delta(T_9, r)) = \epsilon\text{-closure}(R) = \{R\}$$

$$T_{14} = \epsilon\text{-closure}(\delta(T_{10}, l)) = \epsilon\text{-closure}(S) = \{S\}$$

$$T_{15} = \epsilon\text{-closure}(\delta(T_{11}, i)) = \epsilon\text{-closure}(T) = \{T\}$$

$$T_{16} = \epsilon\text{-closure}(\delta(T_{12}, e)) = \epsilon\text{-closure}(U) = \{U\}$$

$$T_{17} = \epsilon\text{-closure}(\delta(T_{13}, n)) = \epsilon\text{-closure}(V) = \{V\}$$

$$T_{18} = \epsilon\text{-closure}(\delta(T_{14}, a)) = \epsilon\text{-closure}(W) = \{W\}$$

$$T_{19} = \epsilon\text{-closure}(\delta(T_{15}, g)) = \epsilon\text{-closure}(X) = \{X\}$$

$$T_{20} = \epsilon\text{-closure}(\delta(T_{16}, n)) = \epsilon\text{-closure}(Y) = \{Y\}$$

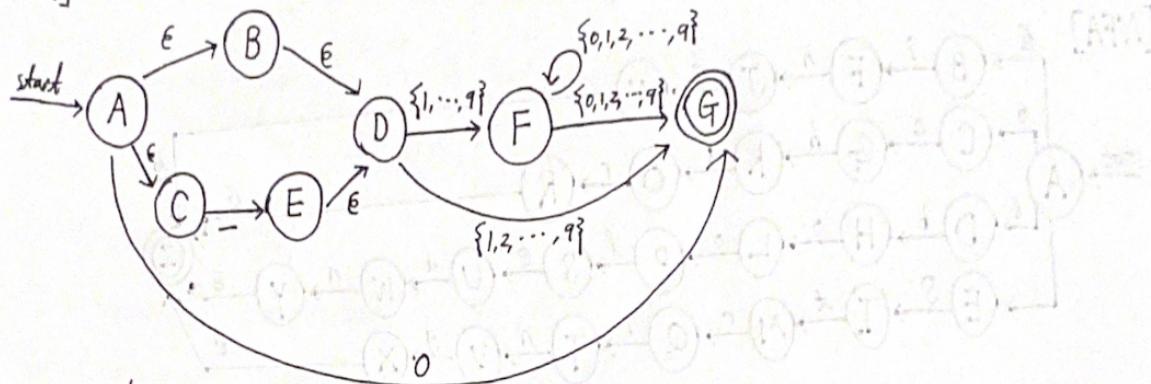
## 1-2) VTYPE DFA table

## 2) SIGNED\_INTEGER (NFA and DFA table)

② Signed Integer.

• Regular Expression:  $0 \mid (-1e) (1|2|3|4|5|6|7|8|9) (\text{DIGIT})^*$

[NFA]



$$T_0 = \epsilon\text{-closure}(A) = \{A, B, C, D\}$$

$$T_1 = \epsilon\text{-closure}(\delta(T_0, -)) = \epsilon\text{-closure}(E) = \{E, D\}$$

$$T_2 = \epsilon\text{-closure}(\delta(T_0, \{1, 2, \dots, 9\})) = \epsilon\text{-closure}(\{F, G\}) = \{F, G\}$$

$$T_3 = \epsilon\text{-closure}(\delta(T_0, 0)) = \epsilon\text{-closure}(G) = \{G\}$$

$$\epsilon\text{-closure}(\delta(T_1, \{1, 2, \dots, 9\})) = \{F, G\}$$

$$\epsilon\text{-closure}(\delta(T_2, \{0, 1, \dots, 9\})) = \{F, G\}$$

[DFA]

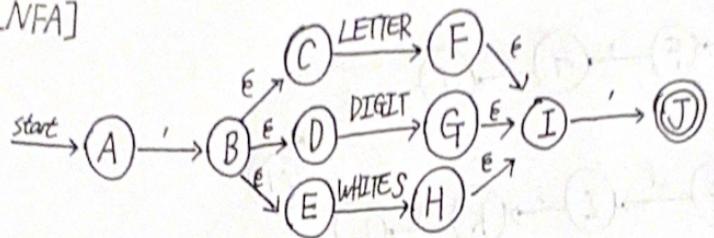
|       | 0     | -     | $\{1, 2, \dots, 9\}$ | $\{0, 1, 2, \dots, 9\}$ |
|-------|-------|-------|----------------------|-------------------------|
| $T_0$ | $T_3$ | $T_1$ | $T_2$                |                         |
| $T_1$ |       |       | $T_2$                |                         |
| $T_2$ |       |       |                      | $T_2$                   |
| $T_3$ |       |       |                      |                         |

### 3) SINGLE\_CHARACTER (NFA and DFA table)

③ Single character.

Regular Expression:  $(') (LETTER \mid DIGIT \mid WHITESPACE) ()$

[NFA]



$$T_0 = \epsilon\text{-closure}(A) = \{A\}$$

$$T_1 = \epsilon\text{-closure}(\delta(T_0, ')) = \epsilon\text{-closure}(B) = \{B, C, D, E\}$$

$$T_2 = \epsilon\text{-closure}(\delta(T_1, LETTER)) = \epsilon\text{-closure}(F) = \{F, I\}$$

$$T_3 = \epsilon\text{-closure}(\delta(T_1, DIGIT)) = \epsilon\text{-closure}(G) = \{G, I\}$$

$$T_4 = \epsilon\text{-closure}(\delta(T_1, WHITESPACE)) = \epsilon\text{-closure}(H) = \{H, I\}$$

$$T_5 = \epsilon\text{-closure}(\delta(T_2, ')) = \epsilon\text{-closure}(J) = \{J\}$$

$$\epsilon\text{-closure}(\delta(T_3, ')) = \dots = \{J\}$$

$$\epsilon\text{-closure}(\delta(T_4, ')) = \dots = \{J\}$$

[DFA]

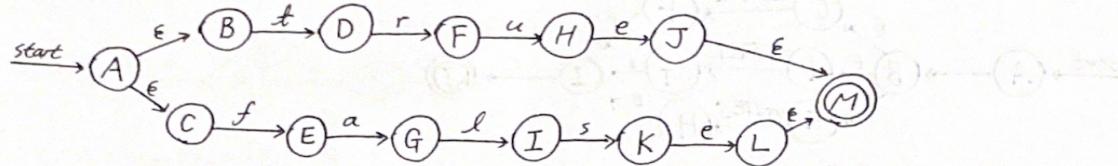
|       | $'$   | LETTER | DIGIT | WHITESPACE |
|-------|-------|--------|-------|------------|
| $T_0$ | $T_1$ |        |       |            |
| $T_1$ |       | $T_2$  | $T_3$ | $T_4$      |
| $T_2$ | $T_5$ |        |       |            |
| $T_3$ | $T_5$ |        |       |            |
| $T_4$ | $T_5$ |        |       |            |
| $T_5$ |       |        |       |            |

#### 4) BOOLEAN\_STRING (NFA and process for DFA)

##### ④ BOOLEAN

• Regular Expression : true | false

[NFA]



$$T_0 = \epsilon\text{-closure}(A) = \{A, B, C\}$$

$$T_1 = \epsilon\text{-closure}(\delta(T_0, t)) = \{D\}$$

$$T_2 = \epsilon\text{-closure}(\delta(T_1, r)) = \{F\}$$

$$T_3 = \epsilon\text{-closure}(\delta(T_2, u)) = \{H\}$$

$$T_4 = \epsilon\text{-closure}(\delta(T_3, e)) = \epsilon\text{-closure}(J) = \{J, M\}$$

$$T_5 = \epsilon\text{-closure}(\delta(T_0, f)) = \emptyset \quad \epsilon\text{-closure}(E) = \{E\}$$

$$T_6 = \epsilon\text{-closure}(\delta(T_5, a)) = \epsilon\text{-closure}(G) = \{G\}$$

$$T_7 = \epsilon\text{-closure}(\delta(T_6, l)) = \epsilon\text{-closure}(I) = \{I\}$$

$$T_8 = \epsilon\text{-closure}(\delta(T_7, s)) = \epsilon\text{-closure}(K) = \{K\}$$

$$T_9 = \epsilon\text{-closure}(\delta(T_8, e)) = \epsilon\text{-closure}(L) = \cancel{\{L, M\}}$$

#### 4-2) BOOLEAN\_STRING (DFA table)

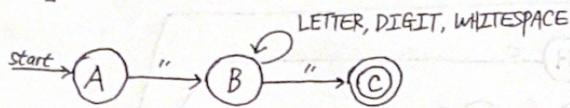
|    | t  | r  | u  | e  | f  | a  | l  | s  |
|----|----|----|----|----|----|----|----|----|
| T0 | T1 |    |    |    | T5 |    |    |    |
| T1 |    | T2 |    |    |    |    |    |    |
| T2 |    |    | T3 |    |    |    |    |    |
| T3 |    |    |    | T4 |    |    |    |    |
| T4 |    |    |    |    |    |    |    |    |
| T5 |    |    |    |    |    | T6 |    |    |
| T6 |    |    |    |    |    |    | T7 |    |
| T7 |    |    |    |    |    |    |    | T8 |
| T8 |    |    |    | T9 |    |    |    |    |
| T9 |    |    |    |    |    |    |    |    |

## 5) LITERAL\_STRING (NFA and DFA table)

### ⑤ LITERAL STRING

• Regular Expression :  $(")(\langle \text{LETTER} \rangle | \langle \text{DIGIT} \rangle | \langle \text{WHITE SPACE} \rangle)^* (" )$

[NFA]



$$T_0 = \epsilon\text{-closure}(A) = \{A\}$$

$$T_1 = \epsilon\text{-closure}(\delta(T_0, '\")) = \epsilon\text{-closure}(B) = \{B\}$$

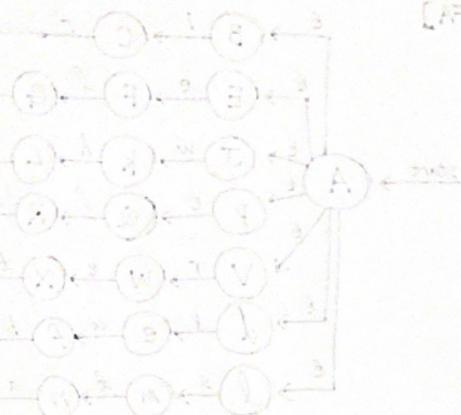
$$\boxed{T_2} = \epsilon\text{-closure}(\delta(T_1, \text{LETTER})) = \epsilon\text{-closure}(B) = \{B\}$$

$$= \epsilon\text{-closure}(\delta(T_1, \text{DIGIT})) = \epsilon\text{-closure}(B) = \{B\}$$

$$= \epsilon\text{-closure}(\delta(T_1, \text{WHITE SPACE})) = \epsilon\text{-closure}(B) = \{B\}$$

$$\boxed{T_2} \neq \epsilon\text{-closure}(\delta(T_1, '\")) = \epsilon\text{-closure}(C) = \{C\}$$

[DFA]



[DFA]

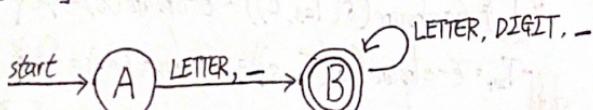
| '"            | LETTER | DIGIT | WHITE SPACE |
|---------------|--------|-------|-------------|
| $T_0$         | $T_1$  |       |             |
| $T_1$         | $T_2$  | $T_1$ | $T_1$       |
| $\boxed{T_2}$ |        |       |             |

## 6) IDENTIFIER (NFA and DFA table)

### ⑥ IDENTIFIER

• Regular Expression :  $(\_ | \text{LETTER})(\text{LETTER} | \text{DIGIT} | \_)^*$

[NFA]



$$T_0 = \epsilon\text{-closure}(A) = \{A\}$$

$$\boxed{T_1} = \epsilon\text{-closure}(\delta(T_0, \_-)) = \{B\}$$

$$= \epsilon\text{-closure}(\delta(T_0, \text{LETTER})) = \{B\}$$

$$= \epsilon\text{-closure}(\delta(T_1, \text{LETTER})) = \{B\}$$

$$= \epsilon\text{-closure}(\delta(T_1, \text{DIGIT})) = \{B\}$$

$$= \epsilon\text{-closure}(\delta(T_1, \_-)) = \{B\}$$

[DFA]

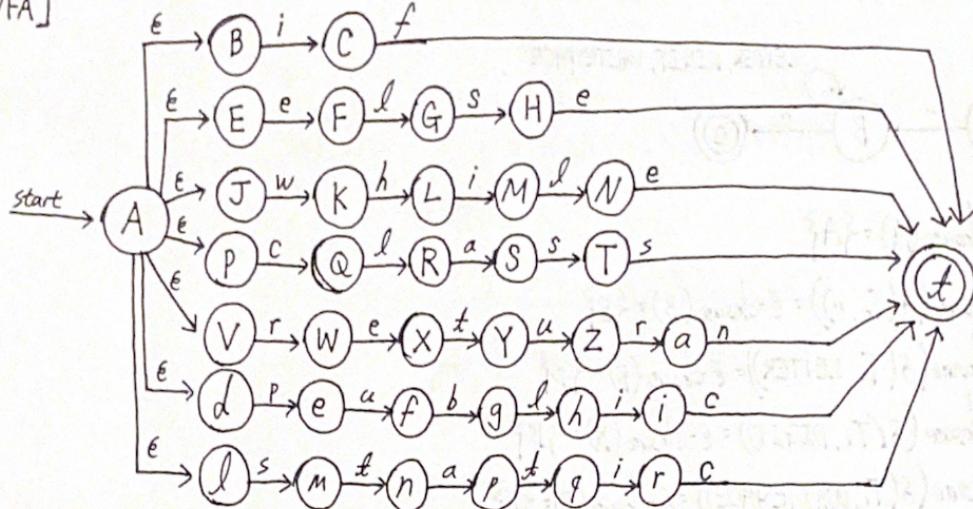
|               | LETTER | DIGIT | -     |
|---------------|--------|-------|-------|
| $T_0$         | $T_1$  |       | $T_1$ |
| $\boxed{T_1}$ | $T_1$  | $T_1$ | $T_1$ |

## 7) KEYWORD (NFA and process for DFA)

### ② KEYWORD

Regular Expression : if | else | while | class | return | public | static

[NFA]



$$T_0 = \epsilon\text{-closure}(A) = \{A, B, E, J, P, V, d, l, f\}$$

$$T_1 = \epsilon\text{-closure}(\delta(T_0, i)) = \epsilon\text{-closure}(C) = \{C\}$$

$$T_2 = \epsilon\text{-closure}(\delta(T_1, f)) = \epsilon\text{-closure}(t) = \{t\}$$

$$T_3 = \epsilon\text{-closure}(\delta(T_0, e)) = \epsilon\text{-closure}(F) = \{F\}$$

$$T_4 = \epsilon\text{-closure}(\delta(T_3, l)) = \epsilon\text{-closure}(G) = \{G\}$$

$$T_5 = \epsilon\text{-closure}(\delta(T_4, s)) = \epsilon\text{-closure}(H) = \{H\}$$

$$T_6 = \epsilon\text{-closure}(\delta(T_5, e)) = \epsilon\text{-closure}(t) = \{t\}$$

$$T_7 = \epsilon\text{-closure}(\delta(T_0, w)) = \epsilon\text{-closure}(K) = \{K\}$$

$$T_8 = \epsilon\text{-closure}(\delta(T_6, h)) = \epsilon\text{-closure}(L) = \{L\}$$

$$T_9 = \epsilon\text{-closure}(\delta(T_7, i)) = \epsilon\text{-closure}(M) = \{M\}$$

$$T_{10} = \epsilon\text{-closure}(\delta(T_8, l)) = \epsilon\text{-closure}(N) = \{N\}$$

$$T_{11} = \epsilon\text{-closure}(\delta(T_9, e)) = \epsilon\text{-closure}(t) = \{t\}$$

$$T_{12} = \epsilon\text{-closure}(\delta(T_{10}, c)) = \epsilon\text{-closure}(Q) = \{Q\}$$

$$T_{13} = \epsilon\text{-closure}(\delta(T_{11}, l)) = \epsilon\text{-closure}(R) = \{R\}$$

$$T_{14} = \epsilon\text{-closure}(\delta(T_{12}, a)) = \epsilon\text{-closure}(S) = \{S\}$$

$$T_{15} = \epsilon\text{-closure}(\delta(T_{13}, s)) = \epsilon\text{-closure}(T) = \{T\}$$

$$T_{16} = \epsilon\text{-closure}(\delta(T_{14}, f)) = \epsilon\text{-closure}(t) = \{t\}$$

$$T_{17} = \epsilon\text{-closure}(\delta(T_{15}, e)) = \epsilon\text{-closure}(X) = \{X\}$$

$$T_{18} = \epsilon\text{-closure}(\delta(T_{16}, h)) = \epsilon\text{-closure}(Y) = \{Y\}$$

$$T_{19} = \epsilon\text{-closure}(\delta(T_{17}, i)) = \epsilon\text{-closure}(Z) = \{Z\}$$

$$T_{20} = \epsilon\text{-closure}(\delta(T_{18}, u)) = \epsilon\text{-closure}(a) = \{a\}$$

$$T_{21} = \epsilon\text{-closure}(\delta(T_{19}, r)) = \epsilon\text{-closure}(t) = \{t\}$$

$$T_{22} = \epsilon\text{-closure}(\delta(T_{20}, l)) = \epsilon\text{-closure}(f) = \{f\}$$

$$T_{23} = \epsilon\text{-closure}(\delta(T_{21}, b)) = \epsilon\text{-closure}(g) = \{g\}$$

$$T_{24} = \epsilon\text{-closure}(\delta(T_{22}, i)) = \epsilon\text{-closure}(h) = \{h\}$$

$$T_{25} = \epsilon\text{-closure}(\delta(T_{23}, e)) = \epsilon\text{-closure}(j) = \{j\}$$

$$T_{26} = \epsilon\text{-closure}(\delta(T_{24}, c)) = \epsilon\text{-closure}(p) = \{p\}$$

$$T_{27} = \epsilon\text{-closure}(\delta(T_{25}, t)) = \epsilon\text{-closure}(q) = \{q\}$$

$$T_{28} = \epsilon\text{-closure}(\delta(T_{26}, f)) = \epsilon\text{-closure}(r) = \{r\}$$

$$T_{29} = \epsilon\text{-closure}(\delta(T_{27}, s)) = \epsilon\text{-closure}(t) = \{t\}$$

$$T_{14} = \epsilon\text{-closure}(\delta(T_0, r)) = \epsilon\text{-closure}(W) = \{W\}$$

$$T_{15} = \epsilon\text{-closure}(\delta(T_{14}, e)) = \epsilon\text{-closure}(X) = \{X\}$$

$$T_{16} = \epsilon\text{-closure}(\delta(T_{15}, t)) = \epsilon\text{-closure}(Y) = \{Y\}$$

$$T_{17} = \epsilon\text{-closure}(\delta(T_{16}, u)) = \epsilon\text{-closure}(Z) = \{Z\}$$

$$T_{18} = \epsilon\text{-closure}(\delta(T_{17}, r)) = \epsilon\text{-closure}(a) = \{a\}$$

$$T_{19} = \epsilon\text{-closure}(\delta(T_{18}, n)) = \epsilon\text{-closure}(t) = \{t\}$$

$$T_{20} = \epsilon\text{-closure}(\delta(T_0, p)) = \epsilon\text{-closure}(e) = \{e\}$$

$$T_{21} = \epsilon\text{-closure}(\delta(T_{19}, u)) = \epsilon\text{-closure}(f) = \{f\}$$

$$T_{22} = \epsilon\text{-closure}(\delta(T_{20}, b)) = \epsilon\text{-closure}(g) = \{g\}$$

$$T_{23} = \epsilon\text{-closure}(\delta(T_{21}, l)) = \epsilon\text{-closure}(h) = \{h\}$$

$$T_{24} = \epsilon\text{-closure}(\delta(T_{22}, i)) = \epsilon\text{-closure}(j) = \{j\}$$

$$T_{25} = \epsilon\text{-closure}(\delta(T_{23}, e)) = \epsilon\text{-closure}(t) = \{t\}$$

$$T_{26} = \epsilon\text{-closure}(\delta(T_{24}, c)) = \epsilon\text{-closure}(f) = \{f\}$$

$$T_{27} = \epsilon\text{-closure}(\delta(T_{25}, t)) = \epsilon\text{-closure}(q) = \{q\}$$

$$T_{28} = \epsilon\text{-closure}(\delta(T_{26}, f)) = \epsilon\text{-closure}(r) = \{r\}$$

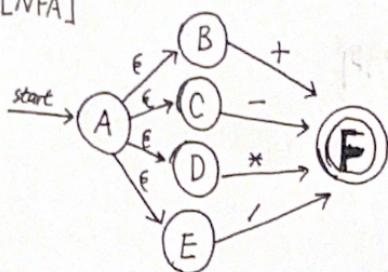
$$T_{29} = \epsilon\text{-closure}(\delta(T_{27}, s)) = \epsilon\text{-closure}(t) = \{t\}$$

## 8) ARITHMETIC\_OPERATOR (NFA and DFA table)

### ⑧ ARITHMETIC OPERATOR

• Regular Expression :  $+ | - | * | /$

[NFA]



$$T_0 = \epsilon\text{-closure}(A) = \{A, B, C, D, E\}$$

$$\begin{aligned} T_1 &= \epsilon\text{-closure}(\delta(T_0, +)) = \{F\} \\ &= \epsilon\text{-closure}(\delta(T_0, -)) = \{F\} \\ &= \epsilon\text{-closure}(\delta(T_0, *)) = \{F\} \\ &= \epsilon\text{-closure}(\delta(T_0, /)) = \{F\} \end{aligned}$$

[DFA]

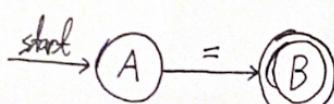
|                | +              | -              | *              | /              |
|----------------|----------------|----------------|----------------|----------------|
| T <sub>0</sub> | T <sub>1</sub> | T <sub>1</sub> | T <sub>1</sub> | T <sub>1</sub> |
| T <sub>1</sub> |                |                |                |                |

## 9) ASSIGNMENT OPERATOR (NFA and DFA table)

### ⑨ Assignment operator.

• Regular Expression :  $=$

[NFA]



$$T_0 = \epsilon\text{-closure}(A) = \{A\}$$

$$T_1 = \epsilon\text{-closure}(\delta(T_0, =)) = \{B\}$$

[DFA]

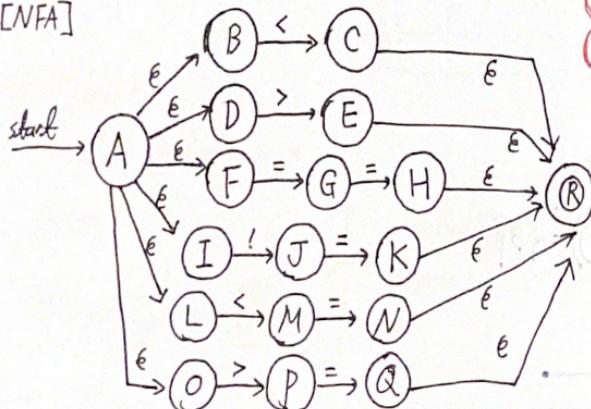
|                | =              |
|----------------|----------------|
| T <sub>0</sub> | T <sub>1</sub> |
| T <sub>1</sub> |                |

## 10) COMPARISON\_OPERATOR (NFA and process for DFA)

### ⑩ Comparison Operators.

Regular Expression : < | > | == | != | <= | >=

[NFA]



$$\begin{aligned}
 T_0 &= \epsilon\text{-closure}(A) = \{A, B, D, F, I, L, O\} \\
 T_1 &= \epsilon\text{-closure}(\delta(T_0, <)) = \epsilon\text{-closure}(\{C, M\}) = \{C, M, R\} \\
 T_2 &= \epsilon\text{-closure}(\delta(T_0, >)) = \epsilon\text{-closure}(\{E, P\}) = \{E, P, R\} \\
 T_3 &= \epsilon\text{-closure}(\delta(T_0, =)) = \epsilon\text{-closure}(\{G\}) = \{G\} \\
 T_4 &= \epsilon\text{-closure}(\delta(T_0, !=)) = \epsilon\text{-closure}(\{J\}) = \{J\} \\
 T_5 &= \epsilon\text{-closure}(\delta(T_1, =)) = \epsilon\text{-closure}(\{N\}) = \{N, R\} \\
 T_6 &= \epsilon\text{-closure}(\delta(T_2, =)) = \epsilon\text{-closure}(\{Q\}) = \{Q, R\} \\
 T_7 &= \epsilon\text{-closure}(\delta(T_3, =)) = \epsilon\text{-closure}(\{H\}) = \{H, R\} \\
 T_8 &= \epsilon\text{-closure}(\delta(T_4, =)) = \epsilon\text{-closure}(\{K\}) = \{K, R\}
 \end{aligned}$$

### 10-2) COMPARISON\_OPERATOR (DFA table)

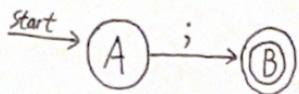
|    | <  | >  | =  | != |
|----|----|----|----|----|
| T0 | T1 | T2 | T3 | T4 |
| T1 |    |    | T5 |    |
| T2 |    |    | T6 |    |
| T3 |    |    | T7 |    |
| T4 |    |    | T8 |    |
| T5 |    |    |    |    |
| T6 |    |    |    |    |
| T7 |    |    |    |    |
| T8 |    |    |    |    |

## 11) TERMINATING SYMBOL (NFA and DFA table)

### ⑪ TERMINATING SYMBOL

• Regular Expression : ;

[NFA]

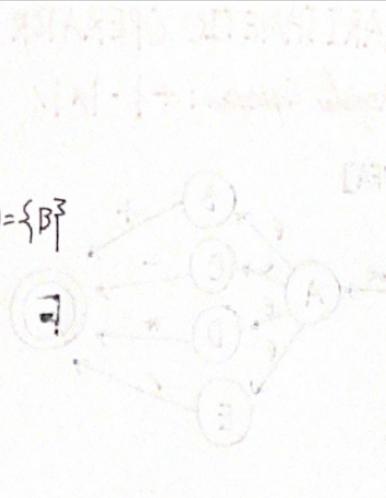


$$T_0 = \epsilon\text{-closure}(A) = \{A\}$$

$$T_1 = \epsilon\text{-closure}(\delta(T_0, ;)) = \epsilon\text{-closure}(B) = \{B\}$$

[DFA]

|                | ; |  |
|----------------|---|--|
| T <sub>0</sub> |   |  |
| T <sub>1</sub> |   |  |

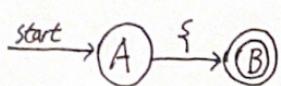


## 12,13) LEFT and RIGHT BRACE (NFA and DFA table)

### ⑫ LEFT BRACE

• Regular Expression : {

[NFA]



$$T_0 = \epsilon\text{-closure}(A) = \{A\}$$

$$T_1 = \epsilon\text{-closure}(\delta(T_0, \{)) = \epsilon\text{-closure}(B) = \{B\}$$

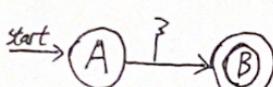
[DFA]

|                | { |  |
|----------------|---|--|
| T <sub>0</sub> |   |  |
| T <sub>1</sub> |   |  |

### ⑬ RIGHT BRACE

• Regular Expression : }

[NFA]



$$T_0 = \epsilon\text{-closure}(A) = \{A\}$$

$$T_1 = \epsilon\text{-closure}(\delta(T_0, }) = \epsilon\text{-closure}(B) = \{B\}$$

[DFA]

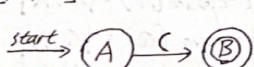
|                | } |  |
|----------------|---|--|
| T <sub>0</sub> |   |  |
| T <sub>1</sub> |   |  |

## 14,15) LEFT and RIGHT PAREN (NFA and DFA table)

### ⑯ LEFT PAREN

• Regular Expression : (

[NFA]



$$T_0 = \epsilon\text{-closure}(A) = \{A\}$$

$$T_1 = \epsilon\text{-closure}(\delta(T_0, ())) = \{B\}$$

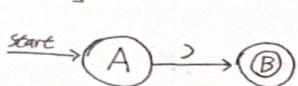
[DFA]

|       |   |
|-------|---|
|       | C |
| $T_0$ |   |
| $T_1$ |   |

### ⑰ RIGHT PAREN

• Regular Expression : )

[NFA]



$$T_0 = \epsilon\text{-closure}(A) = \{A\}$$

$$T_1 = \epsilon\text{-closure}(\delta(T_0, })) = \{B\}$$

[DFA]

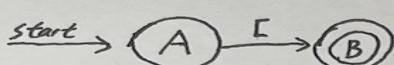
|       |   |
|-------|---|
|       | ) |
| $T_0$ |   |
| $T_1$ |   |

## 16,17) LEFT and RIGHT BRACKET (NFA and DFA table)

### ⑯ LEFT BRACKET

• Regular Expression : [

[NFA]



$$T_0 = \epsilon\text{-closure}(A) = \{A\}$$

$$T_1 = \epsilon\text{-closure}(\delta(T_0, [)) = \{B\}$$

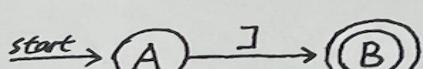
[DFA]

|       |   |
|-------|---|
|       | [ |
| $T_0$ |   |
| $T_1$ |   |

### ⑰ RIGHT BRACKET

• Regular Expression : ]

[NFA]



$$T_0 = \epsilon\text{-closure}(A) = \{A\}$$

$$T_1 = \epsilon\text{-closure}(\delta(T_0, ]) = \{B\}$$

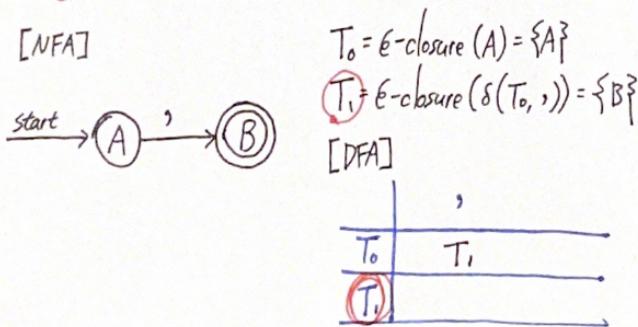
[DFA]

|       |   |
|-------|---|
|       | ] |
| $T_0$ |   |
| $T_1$ |   |

## 18) COMMA (NFA and DFA table)

### (18) COMMA

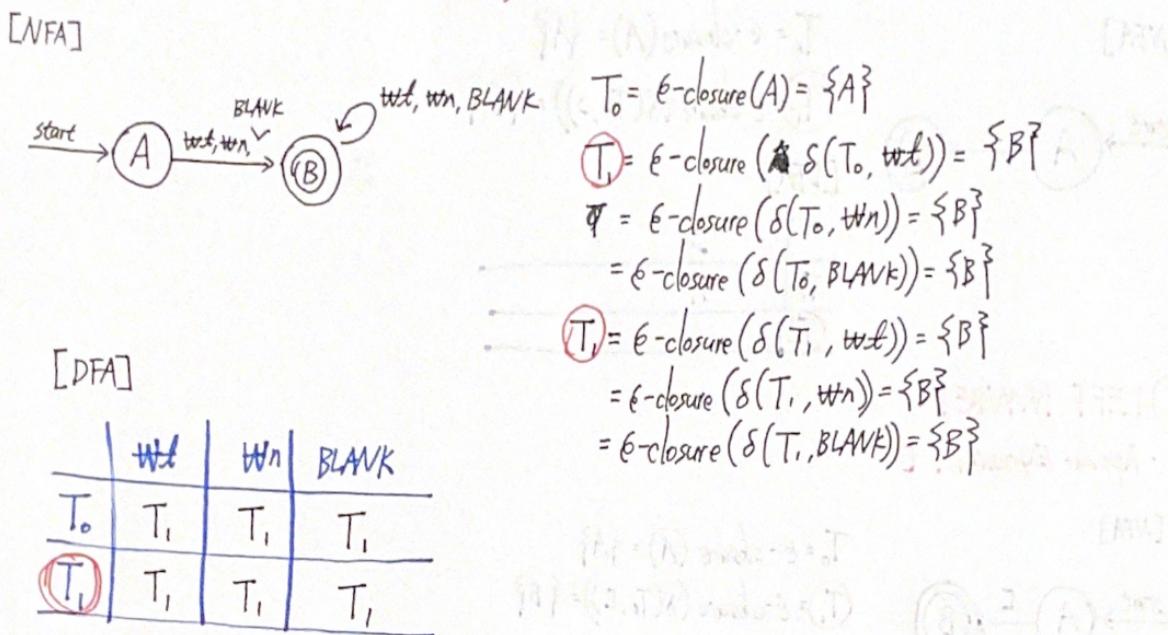
• Regular Expression : ,



## 19) WHITESPACE (NFA and DFA table)

### (19) WHITE SPACE.

• Regular Expression : ( $\text{wt} \mid \text{wn} \mid \text{BLANK}$ )<sup>+</sup>



### 3. All about how the lexical analyzer works.

#### <구현과정 및 소스코드 설명>

첫 번째로, **dfa\_table.py**라는 파일에 위에서 구한 DFA Table을 코드로 나타내었습니다. Input에 따른 DFA state 변화를 효율적으로 관리하기 위해 Python에서 제공하는 자료형인 딕셔너리를 활용하였습니다. 가장 먼저, 다음과 같이 **dfa\_table**이라는, 각각의 DFA에 대한 state table을 value로 가지는 딕셔너리를 생성하였습니다.

```
dfa_table = {
    "SIGNED_INTEGER": {
        "T0": {"0": "T3", "-": "T1", DIGIT_1T09: "T2", "FINAL": False},
        "T1": {DIGIT_1T09: "T2", "FINAL": False},
        "T2": {DIGIT: "T2", "FINAL": True},
        "T3": {"FINAL": True}
    },
    "SINGLE_CHARACTER": {
        "T0": {"'': "T1", "FINAL": False},
        "T1": {SYMBOL_FOR_CHAR: "T2", DIGIT: "T3", " ': "T4", "FINAL": False},
        "T2": {"'': "T5", "FINAL": False},
        "T3": {"'': "T5", "FINAL": False},
        "T4": {"'': "T5", "FINAL": False},
        "T5": {"FINAL": True}
    },
    "BOOLEAN_STRING": {
        "T0": {"t": "T1", f: "T5", "FINAL": False},
        "T1": {"r": "T2", "FINAL": False},
        "T2": {"u": "T3", "FINAL": False},
        "T3": {"e": "T4", "FINAL": False},
        "T4": {"FINAL": True},
        "T5": {"a": "T6", "FINAL": False},
        "T6": {"l": "T7", "FINAL": False},
        "T7": {"s": "T8", "FINAL": False},
        "T8": {"e": "T9", "FINAL": False},
        "T9": {"FINAL": True}
    }
},
```

**dfa\_table**의 각각의 value를 역시 딕셔너리 형태로 각자의 DFA state change를 나타내고 있으며, 이들은 현재 state에서 어떤 입력을 받았을 때, 어떤 state로 이동하는지 나타내며, 더하여 해당 state가 Final state인지를 True or False로 나타내었습니다.

위의 코드에서 예를 들면, SIGNED\_INTEGER DFA는 T0 state에서 0을 입력받으면 table에서 look up하여 T3 state로 이동하며, 이때 T0 state는 Final state가 아닙니다.

두번째로 **dfa.py** 파일입니다.

**dfa.py** 파일에서는 **lexical\_analyzer.py**에서 사용할 DFA class를 정의하였습니다. DFA class는 필드 값으로 DFA에서 처리할 token name과, 현재 상태인 current\_state, 지금까지 성공적으로 처리해온 input들을 보관하는 input record를 갖습니다.

또한 `method`로는 가장 먼저 `initialize`를 갖는데, 이는 DFA object의 `current_state`와 `input_record`를 각각 초기 값(`T0`과 "")으로 초기화합니다.

다음으로 `is_final_state`라는 메서드를 갖는데, 이는 DFA object의 `current_state`가 해당 `token`의 Final state인지 확인하는 메서드입니다.

다음으로 `get_record`메서드는, `input_record` 필드 값을 반환하는 메서드로, 후에 `output_file`에 `token value`를 기록하기 위해 정의되었습니다.

마지막으로 `run` 메서드는, DFA에서 새로운 Input을 처리하는 메서드입니다.

새로운 Input 처리를 시도하며, 처리할 수 없는 input이라서 `KeyError`가 발생하였는지 여부와, `KeyError`가 발생하였다면, `current_state`가 Final인지 여부에 따라 다음과 같이 `return` 합니다.

- 처리 불가능한 input이며, 마지막 state가 Final state가 아님 : ("fail", "mess")를 반환
- 처리 불가능한 input이며, 마지막 state가 Final state임 : ("end", self.input\_record) 반환
- 처리 가능한 input이라면 ("going", self.input\_record) 반환

마지막으로 `lexical_analyzer.py` 파일입니다.

이는 `input_file`을 implemented lexical analyzer가 처리하는 전반적인 로직이 담긴 파일입니다.

우선 기본 로직은 다음과 같습니다.

1. `Input file`로 부터, 한 글자씩 읽으며, 이 값을 `input_char`라는 변수에 저장한다.
2. `input_char`를 각각의 DFA.run(`input_char`) method에 대입하여 수행한다.
3. 이때, 모든 DFA에서 `run` method를 호출하는 것이 아니라, `success_DFA`라는, 지금까지의 `input`들을 처리 할 수 있었던 DFA List에 포함되는 DFA에 대해서만 수행한다.
4. 결과에 따라 다음과 같이 처리한다.
  - `result[0] == "fail"`이라면, 해당 DFA를 `success_DFA`에서 `remove`, `remove`한 후, `success_DFA`에 남아있는 DFA가 존재하지 않는다면, 모든 DFA가 처리할 수 없는 input이므로 Error 메시지 출력
  - `result[0] == "end"`라면, `TokenName`과 `value`를 `output_file`에 기록. 이때 여러가지 고려할 사항들은 if를 통해 처리하였으며, 다음과 같이 처리하였다.
    1. '='만 입력받아도, `ASSIGNMENT_OPER`는 Final state에 도달한다. 하지만 다음 `input`이 다시 한번 '='이라면, `ASSIGNMENT_OPER`가 "end"를 `return`하더라도 이는, `COMPARISON_OPER`이다. 따라서 `end`를 `return`한 DFA가 `ARITHMETIC`이라면, `input`을 확인하여 '='이 아닐 경우에만 `ASSIGNMENT_OPER`로 기록, '='이라면 `COMPARISON_OPER`로 기록한다.
    2. ','를 입력받으면, `COMMA DFA`는 Final state에 도달한다. 하지만 교수님께서 주신 Requirement에는 단순 `COMMA`는 언급이 없으며, `COMMA`가 Function의 `parameters`를 separating하는 경우를 언급하고 있다. 따라서 ','를 입력받아서, `COMMA DFA`가 "end"를 반환한다면, 이전 입력들을 하나하나 확인하며, '('이 먼저 나오는지, ')'이 먼저 나오는지 확인한다.

만약 ‘(‘이 먼저 나온다면, 이는 ARGS\_SEPERATING이며, ‘)’가 먼저 나오거나, 둘다 나오지 않는다면, 이는 단순 COMMA이므로, output에 기록하지 않고, Error 메시지를 기록하여 프로그램을 종료한다.

3. 마지막으로, ‘-’를 입력받으면, 이는 ARITHMETIC\_OPER와 SIGNED\_INTEGER중에 어느것을 위해 사용되었는지 까다로운 고민이 필요하였다.

첫번째로 시도한 방법은, ‘-’ 직전의 input(Whitespace 제외)이 DIGIT에 해당할 경우에는 ‘-’를 ARITHMETIC\_OPER로 처리해보았다.  
하지만 이 경우에는, 문자열 연산 등에서 ‘-’가 사용될 경우, ‘-’를 ARITHMETIC\_OPER로 처리하지 못하는 문제가 발생하였다.

두번째 시도에서는, 반대로 SIGNED\_INTEGER에서 쓰이는 ‘-’의 조건을 생각해 보았다. 만약 ‘-’이 SIGNED\_INTEGER의 symbol로 사용된다면 ‘-’ symbol 직전의 input(Whitespace 제외)은 다음 중 하나일 것이다.

+, -, \*, /, =, '\n', ',', '(', ')', '{', '}'

만약 위의 문자가 아닌, 숫자 등이 ‘-’ 직전의 input이라면, 이는 ARITHMETIC이라고 판단하였으며, 직전의 input이 위의 항목 중 하나에 포함된다면, 이는 SIGNED\_INTEGER에 쓰이는 symbol이라고 판단하였다.

- result[0] == “going”라면, input을 처리할 수 있다는 의미이므로, 다음 input들을 계속 받으면 되기에 별다른 처리가 필요하지 않다.  
하지만 예외적으로, 해당 input이 input\_file의 마지막 글자였다면, going을 return한 DFA가 Final state에 도달했는지 확인 한 후, Final state라면 token\_name과 지금까지의 input\_record를 output\_file에 기록하고 프로그램을 종료하며,  
만약 Final state가 아닌데 더이상 처리할 input이 없다면, 이는 Error로 처리한다.
5. 만약 result[0] == “end”여서, TokenName과 Value를 output에 기록하였다면, 각각의 DFA와 success\_DFA를 initializing 한 후, 초기 상태에서 다음 input들을 읽어 나가며 위의 과정을 반복한다.

<실행과정 예시>

e.g. input : age =25;

1. a를 읽어서, successDFA에 포함되는 DFA의 run method를 수행한다.  
이때, success\_DFA의 초기 값은 모든 DFA를 포함하므로, 결국 모든 DFA에 대해 run한다.
2. g를 읽어서, successDFA에 포함되는 DFA의 run method를 수행한다.  
이전 input인 a를 처리하였을 때, 성공적으로 처리 가능했던 DFA는 IDENTIFIER뿐이므로,

현재 success\_DFA에는 IDENTIFIER만 존재한다.

따라서 IDENTIFIER DFA에 대해서만 ‘g’에 대해 run method를 수행한다.

3. e를 읽어서, successDFA에 포함되는 DFA의 run method를 수행한다.  
이전 input인 a와 g를 순차적으로 처리하였을 때, 모두 처리 가능했던 DFA는 IDENTIFIER뿐이므로, 현재 success\_DFA에는 IDENTIFIER만 존재한다.  
따라서 IDENTIFIER DFA에 대해서만 Input e를 처리한다.
4. ‘ ’를 읽어서, successDFA에 포함되는 DFA의 run method를 수행한다.  
이전 input인 a, g, e를 순차적으로 처리하였을 때, 모두 처리 가능했던 DFA는 IDENTIFIER뿐이므로, 현재 success\_DFA에는 IDENTIFIER만 존재한다.  
따라서 IDENTIFIER DFA에 대해서만 Input ‘ ’를 처리한다.  
하지만 IDENTIFIER DFA는 ‘ ’를 처리할 수 없으며, current\_state는 Final이므로 run method는 (“end”, “age”)를 return한다.  
따라서 output\_file에 <IDENTIFIER> age를 기록, 모든 DFA와 success\_DFA를 초기화 한다.
5. 더 이상 ‘ ’를 처리 가능한 DFA가 없어서, e까지만 처리한 결과를 기록하였으므로, index를 조정하여 다시 ‘ ’를 읽어온다.  
이후 success\_DFA에 해당하는 DFA 각각의 run method를 수행한다.  
이때, 5번 과정에서 Token을 output\_file에 기록한 후, success\_DFA를 initializing 하였으므로 결국 모든 DFA에 대해 ‘ ’처리를 시도한다.
6. ‘ ’를 처리 가능한 DFA는 WHITESPACE DFA 뿐이므로, 6번 과정 이후, success\_DFA는 WHITESPACE만 갖게 된다.  
따라서 새로운 input인 '='를 WHITESPACE DFA에서 처리하려고 시도하지만, WHITESPACE DFA는 이를 처리할 수 없다. 즉 (“end”, ‘ ’)를 return할 것이다.  
하지만 Project Requirement에서 Whitespace Token은 기록하지 말라고 언급되어 있기에, 별도의 기록 과정을 생략한 후, 각각의 DFA와 success\_DFA를 initializing 한다.
7. '='를 읽었을 때, 이를 처리할 수 없었으므로, 마찬가지로 index를 조정하여 '=' 처리를 다시 시도한다. 마찬가지로 success\_DFA는 7번 과정 마지막에서 초기화 되었으므로, 결국 모든 DFA에 대해 '='처리를 시도한다.
8. 더 이상 ‘ ’를 처리 가능한 DFA가 없어서, e까지만 처리한 결과를 기록하였으므로, index를 조정하여 다시 ‘ ’를 읽어온다.  
이후 success\_DFA에 해당하는 DFA 각각의 run method를 수행한다.  
이때, 5번 과정에서 Token을 output\_file에 기록한 후, success\_DFA를 initializing 하였으므로 결국 모든 DFA에 대해 ‘ ’처리를 시도한다.
9. '='를 처리 가능했던 DFA는 ASSIGNMENT\_DFA와 COMPARISON\_DFA 뿐이므로, success\_DFA는 9번 과정 이후 이 둘만을 가지고 있을 것이다.  
따라서 새로운 input인 2에 대하여 두 DFA에 대해 run method를 수행한다.  
이때 이 둘은 모두 2라는 input을 처리할 수 없으며,  
ASSIGNMENT\_DFA는 current\_state가 Final이므로 (“end”, '=')를,  
COMPARISON\_DFA는 current\_state가 Final이 아니므로 (“fail”, “mess”)를 반환한다.  
따라서 output\_file에 <ASSIGNMENT\_OPER> = 를 기록한 후,  
각각의 DFA와 success\_DFA를 Initializing 한다.
10. ‘2’를 처리하지 못하여 결과를 기록하였으므로, index를 조정하여 다시 ‘2’를 읽어온다.

이후 success\_DFA에 해당하는 DFA 각각의 run method를 수행한다.

이때, 10번 과정에서 Token을 output\_file에 기록한 후, success\_DFA를 initializing 하였으므로 결국 모든 DFA에 대해 ‘2’ 처리를 시도한다.

이를 처리 가능한 DFA는 SIGNED\_INTEGER 뿐이므로, success\_DFA에는 SIGNED\_INTEGER만 남게 된다.

11. 새로운 input ‘5’를 success\_DFA에 남아있는 유일한 DFA인 SIGNED\_INTEGER의 run method에서 처리한다.

SIGNED\_INTEGER는 5를 처리할 수 있으므로, run method는 (“going”, “25”)를 반환한다.

12. input ‘;’를 success\_DFA에 유일하게 남아있는 SIGNED\_INTEGER DFA에서 처리하려고 시도한다. 하지만, SIGNED\_INTEGER는 ‘;’를 처리할 수 없으며, 이미 Final state에 도달하였으므로 (“end”, “25”)를 반환한다.

따라서 output\_file에 <SIGNED\_INTEGER> 25를 기록한 후, 각각의 DFA와 success\_DFA를 Initializing 한다.

13. ‘;’를 처리하지 못하여 결과를 기록하였으므로, index를 조정하여 다시 ‘;’를 읽어온다.

이후 success\_DFA에 해당하는 DFA 각각의 run method를 수행한다.

이때, 13번 과정에서 Token을 output\_file에 기록한 후, success\_DFA를 initializing 하였으므로 결국 모든 DFA에 대해 ‘;’ 처리를 시도한다.

이를 처리 가능한 DFA는 TERMINATING\_SYMBOL 뿐이므로, success\_DFA에는 TERMINATING\_SYMBOL만 남게 된다.

또한 TERMINATING\_SYMBOL DFA의 run method의 반환값은 (“going”, ‘;’)인데, ‘;’가 input\_file의 마지막 글자이므로 TERMINATING\_SYMBOL DFA가 current\_state에 도달했는지 확인한다.

TERMINATING\_SYMBOL DFA는 ‘;’를 읽은 후, Final state가 되었으므로, output\_file에 <TERMINATING\_SYMBOL> ;를 기록한 후, 프로그램을 종료한다.

<test.java & test.java\_output.txt>

[test.java]

```
public class test {

    public static int main(String[] args, int args2) {
        char character = 'b' + 'c'-'b'-'a';
        int num = 1 - -1 - -2 - -3;
        int num2 = -1 - 8(-3) + -2;
        boolean case = true;

        while(case) {
            case = false != false;
        }
        String d_12 = "Hello Universe!";
        index = 0;
        if(index >= 0){
            index += 1;
            print("This is" + d_12);
        }
        print("index:");
        print(index);
        return 0;
    }

}
```

[test.java\_output.txt]

길이가 너무 길어서 부득이하게 아래에 링크로 기재하였습니다.

[https://github.com/youngkwon02/Simplified\\_Java\\_Compiler/blob/main/test\\_code/test.java\\_output.txt](https://github.com/youngkwon02/Simplified_Java_Compiler/blob/main/test_code/test.java_output.txt)