REPORT

2021-1 Compiler Term-Project

Implement Lexical Analyzer

과목명 | 컴파일러 01분반

담당교수 | 김효수 교수님

학과 | 소프트웨어학부

학년 | 3학년

학번/이름 | 20194538 이나혁

| 20193418 이하윤

제출일 | 2021년 4월 18일

Contents

I. Tokens

II. Regular expression

III. NFA(Non-deterministic Finite Automata) & DFA(Deterministic Finite Automata)

1. Arithmetic NFA, transition graph, DFA
2. Comparison NFA, transition graph, DFA
3. Identifier NFA, transition graph, DFA
4. Literal String NFA, transition graph, DFA
5. Singed Integer NFA, transition graph, DFA
6. Single Character NFA, transition graph, DFA

IV. Implementation

0. Developing Environment
1. Define Tokens & Symbols
2. Implement DFA
3. Check Double Quote & single Quote
4. Input Stream Preprocessing
5. Tokenize
6. Save Result
7. Entire Process Run
8. '-' Symbol issue

V. Experiment

VI. Appendix

1. NFA & DFA with Transition Table by Hand Write

I. Tokens

Token	Lexeme
VARIABLE TYPE	int, char, boolean, String, void, byte, double, float, long
KEYWORD	abstract, break, case, catch, class, continue, default, do. Else, extends, false, finally, for, if, implements, import, instancedof, interface, native, new, null, package, private, protected, public, return, short, static, super, switch, synchronized, this, throw, throws, true, while, args, main
BOOLEAN STRING	true, false
IDENTIFIER	i, j, k, abc, ab_123, func1, func_, _func_bar__
LITERAL STRING	“Hello World”, “ My student id is 12345678”
SIGNED INTEGER	0, 1, 22, -1, -2, …
SINGLE CHARACTER	‘a’, ‘1’, ‘&’
ARITHETIC OPERATORS	+, -, *, /
ASSIGNMENT OPERATORS	=
COMPARISION OPERATORS	<, >, ==, !=, <=, >=
TERMINATING SYMBOL	;
PAREN	{ , } , (,) , [,]
SEPARATING	,
WHITESPACE	₩t, ₩n, blanks

II. Regular expression

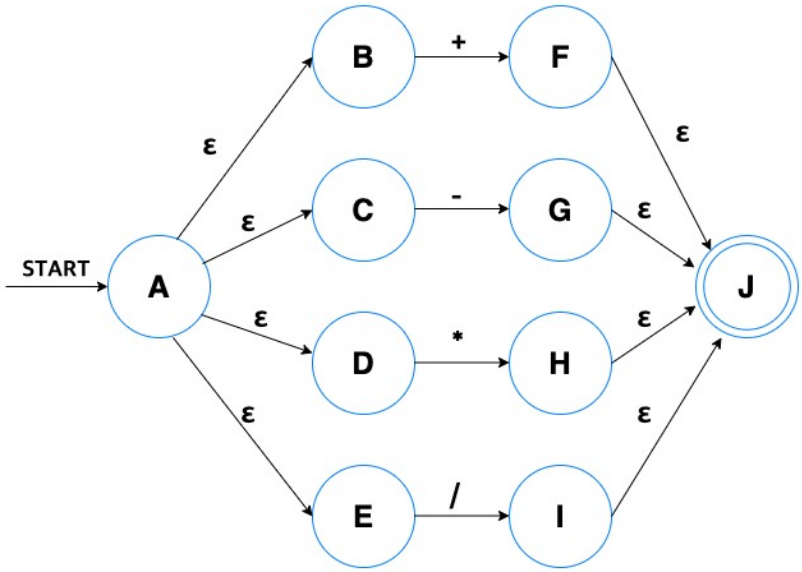
LETTER = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z',
'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z']
ZERO = ['0']
NON_ZERO = ['1', '2', '3', '4', '5', '6', '7', '8', '9']
UNDER_BAR = ['_']
MINUS = ['-']
WHITE_SPACE = [' ', '₩t', '₩n']

Token	Regular Expression
VARIABLE TYPE	int | char | boolean | String | void | byte | double | float | long
KEYWORD	abstract | break | case | catch | class | continue | default | do | else | extends | false | finally | for | if | implements | import | instancedof | interface | native | new | null | package | private | protected | public | return | short | static | super | switch | synchronized | this | throw | throws | true | while | args | main
BOOLEAN STRING	true | false
IDENTIFIER	(letter | under_bar) (zero | non-zero | letter | under_bar)*
LITERAL STRING	“ (zero | non-zero | letter | white_space)* ”
SIGNED INTEGER	zero | ((minus | ε) non-zero (zero | non-zero)*)
SINGLE CHARACTER	‘ (zero | non-zero | letter | white_space) ‘
ARITHETIC OPERATORS	+ | - | * | /
ASSIGNMENT OPERATORS	=
COMPARISION OPERATORS	< | > | == | != | <= | >=
TERMINATING SYMBOL	;
PAREN	{ | } | (|) | [|]
COMMA	,
WHITESPACE	(₩t | | ₩n)*

III. NFA & DFA

1. Arithmetic

- NFA graph



- Draw DFA

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

$\varepsilon - closure(\delta(T_0, +))$ = $\varepsilon - closure(F)$ = { F, J } = T_1

$\varepsilon - closure(\delta(T_0, -))$ = $\varepsilon - closure(G)$ = { G, J } = T_2

$\varepsilon - closure(\delta(T_0, *))$ = $\varepsilon - closure(H)$ = { H, J } = T_3

$\varepsilon - closure(\delta(T_0, /))$ = $\varepsilon - closure(I)$ = { I, J } = T_4

$\varepsilon - closure(\delta(T_1, +)) = \emptyset$

$\varepsilon - closure(\delta(T_1, -)) = \emptyset$

$\varepsilon - closure(\delta(T_1, *)) = \emptyset$

$\varepsilon - closure(\delta(T_1, /)) = \emptyset$

$\varepsilon - closure(\delta(T_2, +)) = \emptyset$

$\varepsilon - closure(\delta(T_2, -)) = \emptyset$

$\varepsilon - closure(\delta(T_2, *)) = \emptyset$

$\varepsilon - closure(\delta(T_2, /)) = \emptyset$

$\varepsilon - closure(\delta(T_3, +)) = \emptyset$

$\varepsilon - closure(\delta(T_3, -)) = \emptyset$

$\varepsilon - closure(\delta(T_3, *)) = \emptyset$

$\varepsilon - closure(\delta(T_3, /)) = \emptyset$

$\varepsilon - closure(\delta(T_4, +)) = \emptyset$

$\varepsilon - closure(\delta(T_4, -)) = \emptyset$

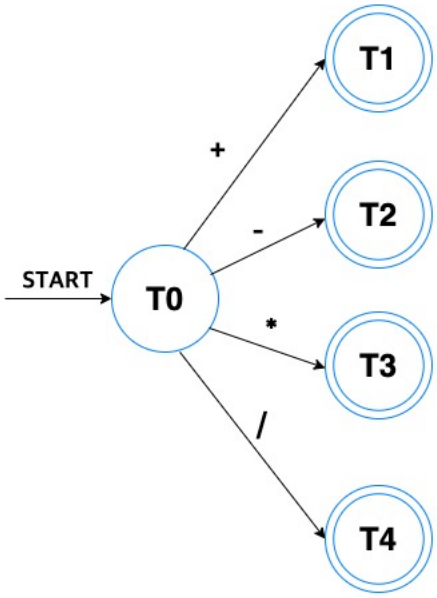
$\varepsilon - closure(\delta(T_4, *)) = \emptyset$

$\varepsilon - closure(\delta(T_4, /)) = \emptyset$

-Transition table

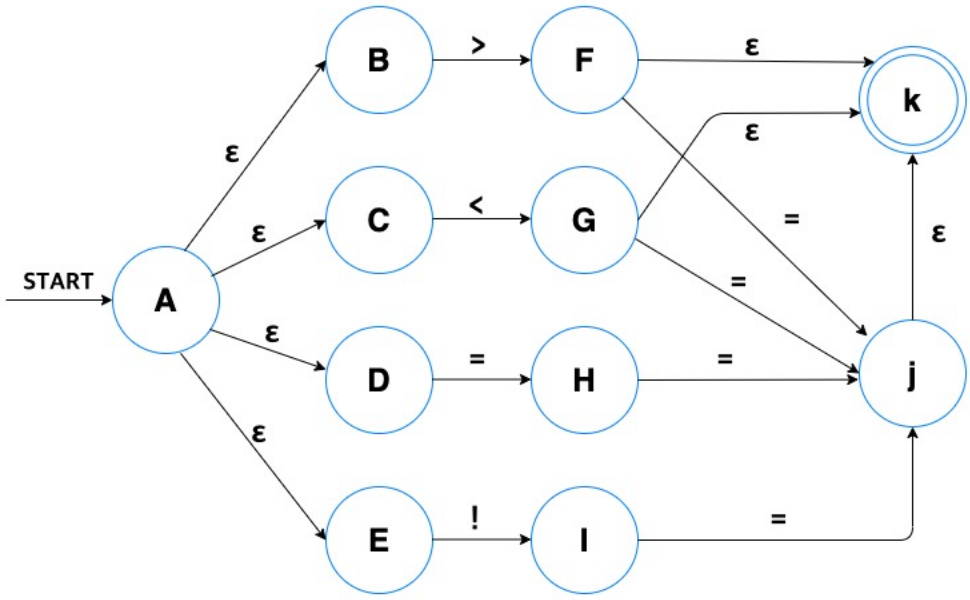
	+	-	*	/
T_0	T_0	T_2	T_3	T_4
T_1	ø	ø	ø	ø
T_2	ø	ø	ø	ø
T_3	ø	ø	ø	ø
T_4	ø	ø	ø	ø

- DFA graph



2. Comparison

- NFA graph



- DRAW DFA

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

$\varepsilon - closure(\delta(T_0, >))$ = $\varepsilon - closure(F)$ = { F, K } = T_1

$\varepsilon - closure(\delta(T_0, <))$ = $\varepsilon - closure(G)$ = { G, K } = T_2

$\varepsilon - closure(\delta(T_0, !))$ = $\varepsilon - closure(H)$ = { H, K } = T_3

$\varepsilon - closure(\delta(T_0, =))$ = $\varepsilon - closure(I)$ = { I, K } = T_4

$\varepsilon - closure(\delta(T_1, >))$ = $\emptyset$

$\varepsilon - closure(\delta(T_1, <))$ = $\emptyset$

$\varepsilon - closure(\delta(T_1, !))$ = $\emptyset$

$\varepsilon - closure(\delta(T_1, =))$ = $\varepsilon - closure(J)$ = { J, K } = T_5

$\varepsilon - closure(\delta(T_2, >))$ = $\emptyset$

$\varepsilon - closure(\delta(T_2, <))$ = $\emptyset$

$\varepsilon - closure(\delta(T_2, !))$ = $\emptyset$

$\varepsilon - closure(\delta(T_2, =))$ = $\varepsilon - closure(J)$ = T_5

$\varepsilon - closure(\delta(T_3, >))$ = $\emptyset$

$\varepsilon - closure(\delta(T_3, <))$ = $\emptyset$

$\varepsilon - closure(\delta(T_3, !))$ = $\emptyset$

$\varepsilon - closure(\delta(T_3, =))$ = $\varepsilon - closure(J)$ = T_5

$\varepsilon - closure(\delta(T_4, >))$ = $\emptyset$

$\varepsilon - closure(\delta(T_4, <))$ = $\emptyset$

$\varepsilon - closure(\delta(T_4, !))$ = $\emptyset$

$\varepsilon - closure(\delta(T_4, =))$ = $\varepsilon - closure(J)$ = T_5

$\varepsilon - closure(\delta(T_5, >))$ = $\emptyset$

$\varepsilon - closure(\delta(T_5, <))$ = $\emptyset$

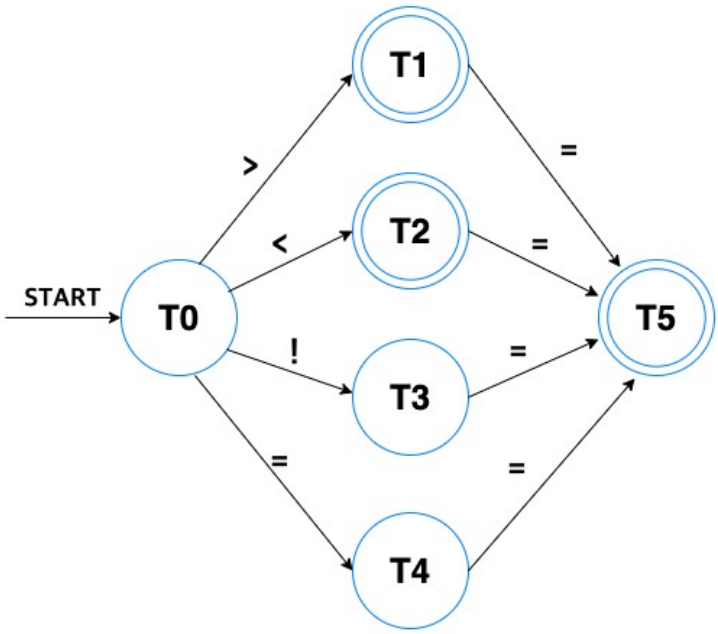
$\varepsilon - closure(\delta(T_5, !))$ = $\emptyset$

$\varepsilon - closure(\delta(T_5, =))$ = $\emptyset$

- TRANSITION TABLE

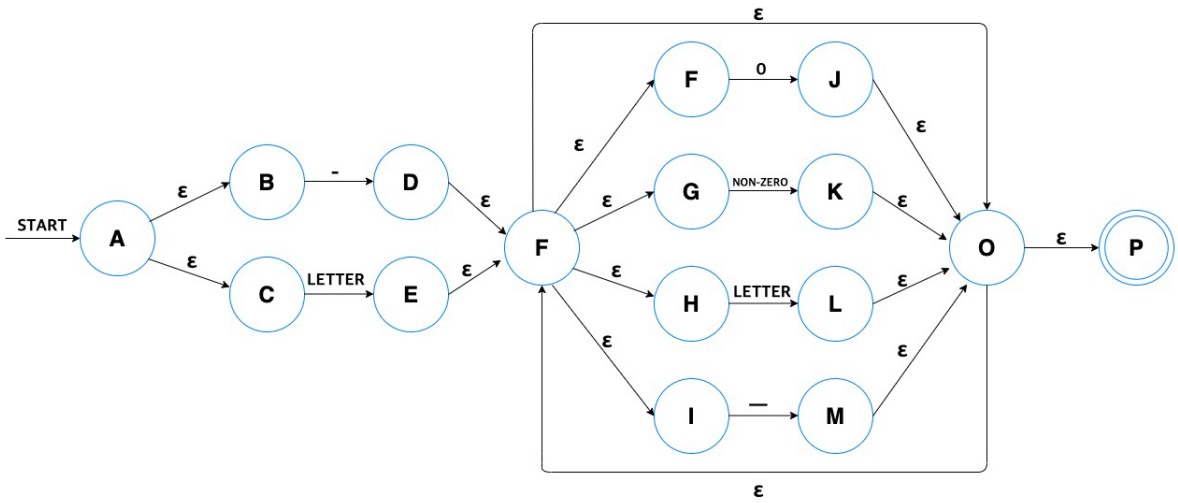
	>	<	!	=
T_0	T_1	T_2	T_3	T_4
T_1	∅	∅	∅	T_5
T_2	∅	∅	∅	T_5
T_3	∅	∅	∅	T_5
T_4	∅	∅	∅	T_5
T_5	∅	∅	∅	T_5
T_6	∅	∅	∅	∅

- DFA GRAPH



3. Identifier

- DFA graph



- DRAW DFA

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

$\varepsilon - closure(\delta(T_0, \_))$ = $\varepsilon - closure(D)$ = { D, F, G, H, I, J, O, P } = T_1

$\varepsilon - closure(\delta(T_0, LETTER))$ = $\varepsilon - closure(E)$ = { E, F, G, H, I, J, O, P } = T_2

$\varepsilon - closure(\delta(T_0, 0))$ = $\emptyset$

$\varepsilon - closure(\delta(T_0, NON - ZERO))$ = $\emptyset$

$\varepsilon - closure(\delta(T_1, 0))$ = $\varepsilon - closure(K)$ = { K, O, F, G, H, I, J, P } = T_3

$\varepsilon - closure(\delta(T_1, NON - ZERO))$ = $\varepsilon - closure(L)$ = { L, O, F, G, H, I, J, P } = T_4

$\varepsilon - closure(\delta(T_1, LETTER))$ = $\varepsilon - closure(M)$ = { M, O, F, G, H, I, J, P } = T_5

$\varepsilon - closure(\delta(T_1, \_))$ = $\varepsilon - closure(N)$ = { N, O, F, G, H, I, J, P } = T_6

$\varepsilon - closure(\delta(T_2, 0))$ = $\varepsilon - closure(K)$ = T_3

$\varepsilon - closure(\delta(T_2, NON - ZERO))$ = $\varepsilon - closure(L)$ = T_4

$\varepsilon - closure(\delta(T_2, LETTER))$ = $\varepsilon - closure(M)$ = T_5

$\varepsilon - closure(\delta(T_2, \_))$ = $\varepsilon - closure(N)$ =T_6

$\varepsilon - closure(\delta(T_3, 0))$ = $\varepsilon - closure(K)$ = T_3

$\varepsilon - closure(\delta(T_3, NON - ZERO))$ = $\varepsilon - closure(L)$ = T_4

$\varepsilon - closure(\delta(T_3, LETTER))$ = $\varepsilon - closure(M)$ = T_5

$\varepsilon - closure(\delta(T_3, \_))$ = $\varepsilon - closure(N)$ =T_6

$\varepsilon - closure(\delta(T_4, 0))$ = $\varepsilon - closure(K)$ = T_3

$\varepsilon - closure(\delta(T_4, NON - ZERO))$ = $\varepsilon - closure(L)$ = T_4

$\varepsilon - closure(\delta(T_4, LETTER))$ = $\varepsilon - closure(M)$ = T_5

$\varepsilon - closure(\delta(T_4, \_))$ = $\varepsilon - closure(N)$ =T_6

$\varepsilon - closure(\delta(T_5, 0))$ = $\varepsilon - closure(K)$ = T_3

$\varepsilon - closure(\delta(T_5, NON - ZERO))$ = $\varepsilon - closure(L)$ = T_4

$\varepsilon - closure(\delta(T_5, LETTER))$ = $\varepsilon - closure(M)$ = T_5

$\varepsilon - closure(\delta(T_5, \_))$ = $\varepsilon - closure(N)$ =T_6

$\varepsilon - closure(\delta(T_6, 0))$ = $\varepsilon - closure(K)$ = T_3

$\varepsilon - closure(\delta(T_6, NON - ZERO))$ = $\varepsilon - closure(L)$ = T_4

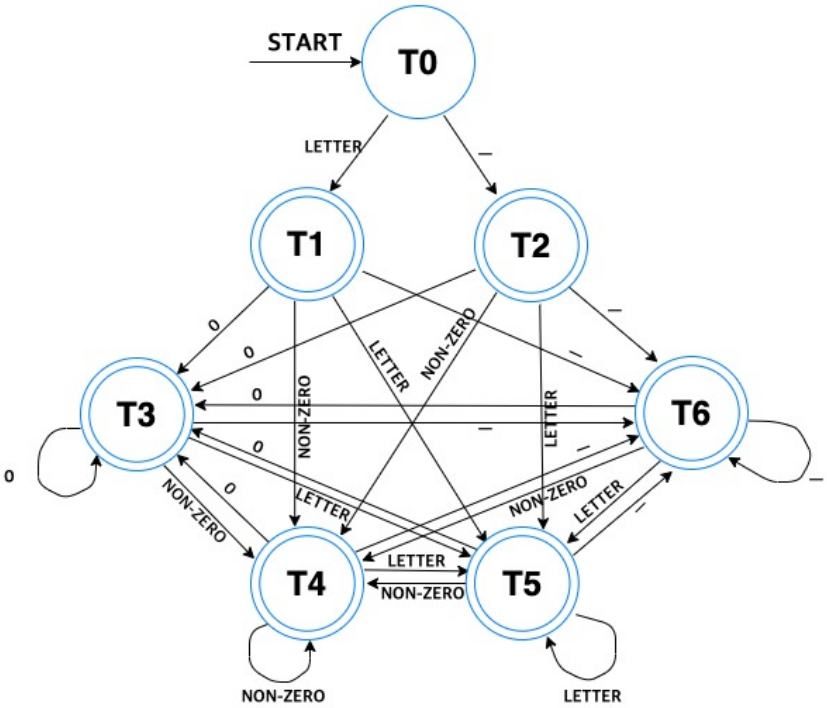
$\varepsilon - closure(\delta(T_6, LETTER))$ = $\varepsilon - closure(M)$ = T_5

$\varepsilon - closure(\delta(T_6, \_))$ = $\varepsilon - closure(N)$ =T_6

- TRANSITION TABLE

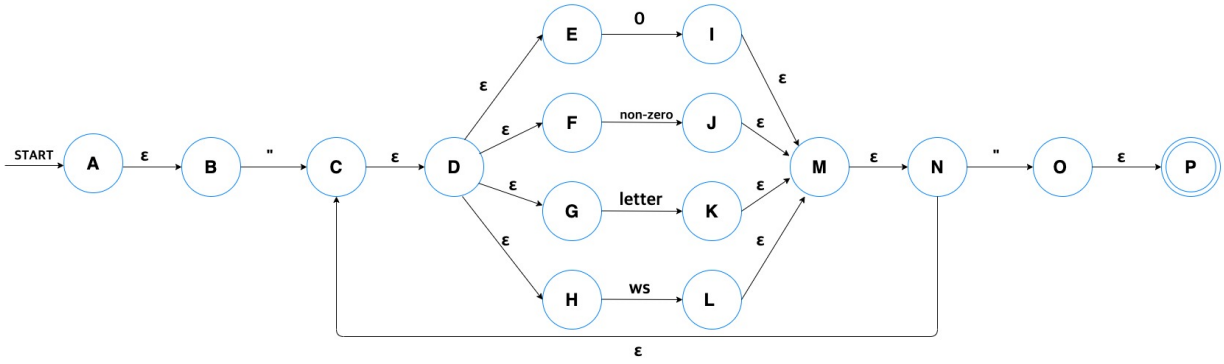
	0	NON-ZERO	LETTER	_
T_0	ø	ø	T_2	T_1
T_1	T_3	T_4	T_5	T_6
T_2	T_3	T_4	T_5	T_6
T_3	T_3	T_4	T_5	T_6
T_4	T_3	T_4	T_5	T_6
T_5	T_3	T_4	T_5	T_6
T_6	T_3	T_4	T_5	T_6

- DFA GRAPH



4. Literal String

- NFA GRAPH



- DRAW DFA

$\varepsilon - closure(A) = \{ A, B \} = T_0$

$\varepsilon - closure(\delta(T_0, 0)) = \emptyset$

$\varepsilon - closure(\delta(T_0, NON - ZERO)) = \emptyset$

$\varepsilon - closure(\delta(T_0, LETTER)) = \emptyset$

$\varepsilon - closure(\delta(T_0, WHITE\_SPACE)) = \emptyset$

$\varepsilon - closure(\delta(T_0, "")) = \varepsilon - closure(C) = \{ C, D, E, F, G, H \} = T_1$

$\varepsilon - closure(\delta(T_1, 0)) = \varepsilon - closure(I) = \{ I, M, N, C, D, E, F, G, H \} = T_2$

$\varepsilon - closure(\delta(T_1, NON - ZERO)) = \varepsilon - closure(J) = \{ J, M, N, C, D, E, F, G, H \} = T_3$

$\varepsilon - closure(\delta(T_1, LETTER)) = \varepsilon - closure(K) = \{ K, M, N, C, D, E, F, G, H \} = T_4$

$\varepsilon - closure(\delta(T_1, WHITE\_SPACE)) = \varepsilon - closure(L) = \{ L, M, N, C, D, E, F, G, H \} = T_5$

$\varepsilon - closure(\delta(T_1, "")) = \emptyset$

$\varepsilon - closure(\delta(T_2, 0)) = \varepsilon - closure(I) = T_2$

$\varepsilon - closure(\delta(T_2, NON - ZERO)) = \varepsilon - closure(J) = T_3$

$\varepsilon - closure(\delta(T_2, LETTER)) = \varepsilon - closure(K) = T_4$

$\varepsilon - closure(\delta(T_2, WHITE\_SPACE)) = \varepsilon - closure(L) = T_5$

$\varepsilon - closure(\delta(T_2, "")) = \varepsilon - closure(O) = \{ O, P \} = T_6$

$\varepsilon - closure(\delta(T_3, 0)) = \varepsilon - closure(I) = T_2$

$\varepsilon - closure(\delta(T_3, NON - ZERO)) = \varepsilon - closure(J) = T_3$

$\varepsilon - closure(\delta(T_3, LETTER)) = \varepsilon - closure(K) = T_4$

$\varepsilon - closure(\delta(T_3, WHITE\_SPACE)) = \varepsilon - closure(L) = T_5$

$\varepsilon - closure(\delta(T_3, "")) = \varepsilon - closure(O) = \{ O, P \} = T_6$

$\varepsilon - closure(\delta(T_4, 0)) = \varepsilon - closure(I) = T_2$

$\varepsilon - closure(\delta(T_4, NON - ZERO)) = \varepsilon - closure(J) = T_3$

$\varepsilon - closure(\delta(T_4, LETTER)) = \varepsilon - closure(K) = T_4$

$\varepsilon - closure(\delta(T_4, WHITE\_SPACE)) = \varepsilon - closure(L) = T_5$

$\varepsilon - closure(\delta(T_4, "")) = \varepsilon - closure(O) = \{ O, P \} = T_6$

$\varepsilon - closure(\delta(T_5, 0)) = \varepsilon - closure(I) = T_2$

$\varepsilon - closure(\delta(T_5, NON - ZERO)) = \varepsilon - closure(J) = T_3$

$\varepsilon - closure(\delta(T_5, LETTER)) = \varepsilon - closure(K) = T_4$

$\varepsilon - closure(\delta(T_5, WHITE\_SPACE)) = \varepsilon - closure(L) = T_5$

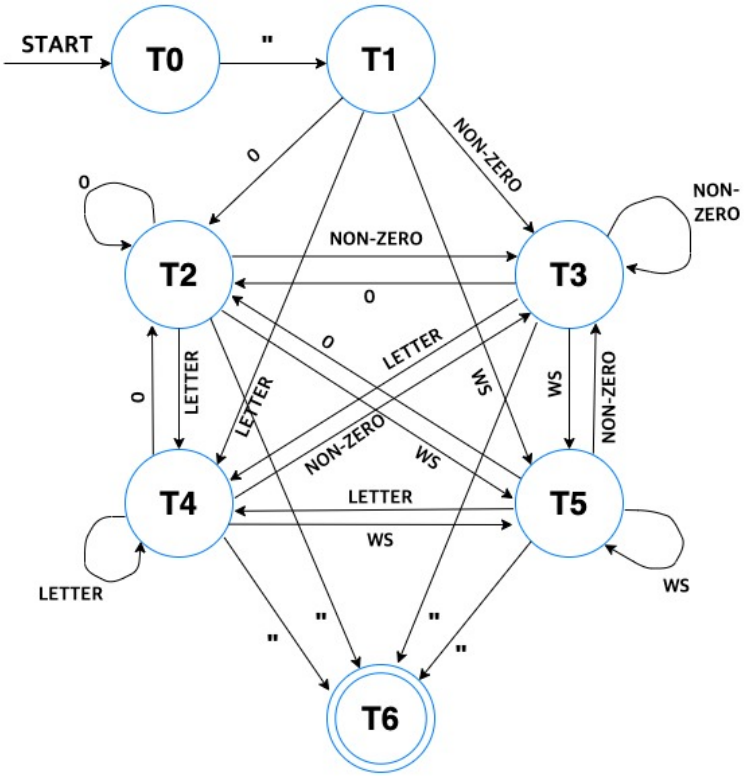
$\varepsilon - closure(\delta(T_5, "")) = \varepsilon - closure(O) = \{ O, P \} = T_6$

$\varepsilon - closure\big(\delta(T_6, 0)\big) = \emptyset$
$\varepsilon - closure\big(\delta(T_6, NON - ZERO)\big) = \emptyset$
$\varepsilon - closure\big(\delta(T_6, LETTER)\big) = \emptyset$
$\varepsilon - closure\big(\delta(T_6, WHITE\_SPACE)\big) = \emptyset$
$\varepsilon - closure\big(\delta(T_6, ")\big) = \emptyset$

- TRANSITION TABLE

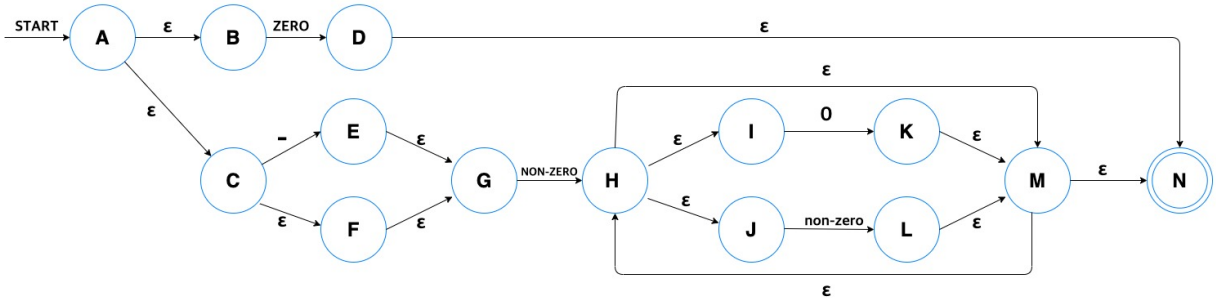
	0	NON-ZERO	LETTER	WHITE_SPACE	“ (DOUBLE_ QUOTE)
T_0	∅	∅	∅	∅	T_1
T_1	T_2	T_3	T_4	T_5	∅
T_2	T_2	T_3	T_4	T_5	T_6
T_3	T_2	T_3	T_4	T_5	T_6
T_4	T_2	T_3	T_4	T_5	T_6
T_5	T_2	T_3	T_4	T_5	T_6
T_6	∅	∅	∅	∅	∅

- DFA GRAPH



5. Signed Integer

- NFA GRAPH



- DFA DRAW

$\varepsilon - closure(A)$ = { A, B, C, F, G } = T_0

$\varepsilon - closure(\delta(T_0, 0))$ = $\varepsilon - closure(D)$ = { D, N }= T_1

$\varepsilon - closure(\delta(T_0, MINUS))$ = $\varepsilon - closure(E)$ = { E, G } = T_2

$\varepsilon - closure(\delta(T_0, NON - ZERO))$ = $\varepsilon - closure(H)$ = { H, I, J, M, N } = T_3

$\varepsilon - closure(\delta(T_1, 0))$ = $\emptyset$

$\varepsilon - closure(\delta(T_1, MINUS))$ = $\emptyset$

$\varepsilon - closure(\delta(T_1, NON - ZERO))$ = $\emptyset$

$\varepsilon - closure(\delta(T_2, 0))$ = $\emptyset$

$\varepsilon - closure(\delta(T_2, MINUS))$ = $\emptyset$

$\varepsilon - closure(\delta(T_2, NON - ZERO))$ = $\varepsilon - closure(H)$ = T_3

$\varepsilon - closure(\delta(T_3, 0))$ = $\varepsilon - closure(K)$ = { K, M, N, H, I, J } = T_4

$\varepsilon - closure(\delta(T_3, MINUS))$ = $\emptyset$

$\varepsilon - closure(\delta(T_3, NON - ZERO))$ = $\varepsilon - closure(L)$ = { L, M, N, H, I, J } = T_5

$\varepsilon - closure(\delta(T_4, 0))$ = $\varepsilon - closure(K)$ = T_4

$\varepsilon - closure(\delta(T_4, MINUS))$ = $\emptyset$

$\varepsilon - closure(\delta(T_4, NON - ZERO))$ = $\varepsilon - closure(L)$ = T_5

$\varepsilon - closure(\delta(T_5, 0))$ = $\varepsilon - closure(K)$ = T_4

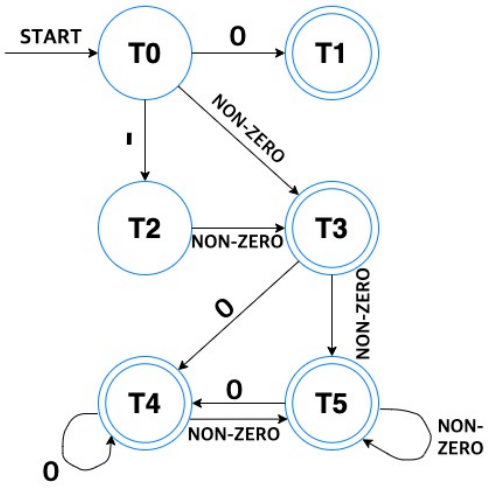
$\varepsilon - closure(\delta(T_5, MINUS))$ = $\emptyset$

$\varepsilon - closure(\delta(T_5, NON - ZERO))$ = $\varepsilon - closure(L)$ = T_5

- TRANSITION TABLE

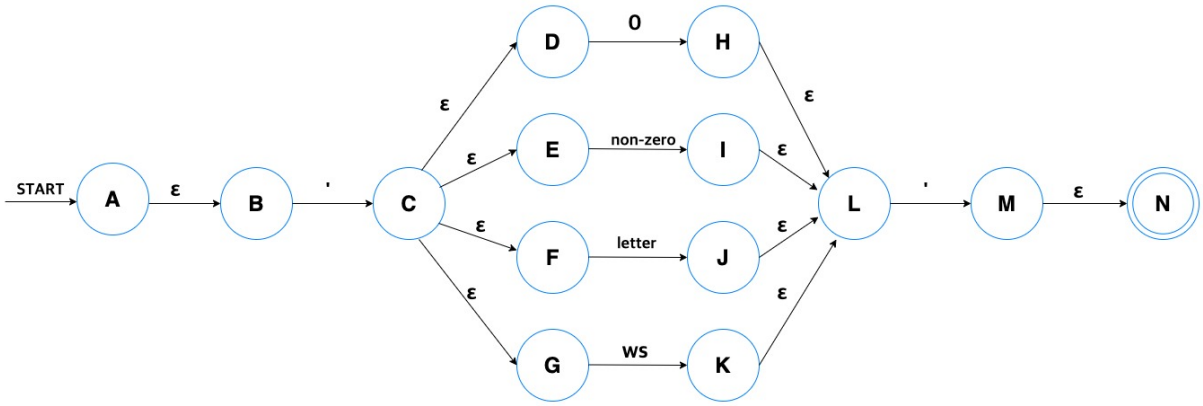
	0	MINUS	NON-ZERO
T_0	T_1		
T_1	∅	∅	∅
T_2	∅	∅	T_3
T_3	T_4	∅	T_5
T_4	T_4	∅	T_5
T_5	T_4	∅	T_5

- DFA GRAPH



6. Single Character

- NFA GRAPH



- DRAW DFA

$\varepsilon - closure(A)$ = { A, B } = T_0

$\varepsilon - closure(\delta(T_0, 0))$ = $\emptyset$

$\varepsilon - closure(\delta(T_0, NON - ZERO))$ = $\emptyset$

$\varepsilon - closure(\delta(T_0, LETTER))$ = $\emptyset$

$\varepsilon - closure(\delta(T_0, WHITE\_SPACE))$ = $\emptyset$

$\varepsilon - closure(\delta(T_0, '))$ = $\varepsilon - closure(C)$ = { C, D, E, F, G, H } = T_1

$\varepsilon - closure(\delta(T_1, 0))$ = $\varepsilon - closure(H)$ = { H, L } = T_2

$\varepsilon - closure(\delta(T_1, NON - ZERO))$ = $\varepsilon - closure(I)$ = { I, L } = T_3

$\varepsilon - closure(\delta(T_1, LETTER))$ = $\varepsilon - closure(J)$ = { J, L } = T_4

$\varepsilon - closure(\delta(T_1, WHITE\_SPACE))$ = $\varepsilon - closure(K)$ = { K, L } = T_5

$\varepsilon - closure(\delta(T_1, '))$ = $\emptyset$

$\varepsilon - closure(\delta(T_2, 0))$ = $\emptyset$

$\varepsilon - closure(\delta(T_2, NON - ZERO))$ = $\emptyset$

$\varepsilon - closure(\delta(T_2, LETTER))$ = $\emptyset$

$\varepsilon - closure(\delta(T_2, WHITE\_SPACE))$ = $\emptyset$

$\varepsilon - closure(\delta(T_2, '))$ = $\varepsilon - closure(M)$ = { M, N } = T_6

$\varepsilon - closure(\delta(T_3, 0))$ = $\emptyset$

$\varepsilon - closure(\delta(T_3, NON - ZERO))$ = $\emptyset$

$\varepsilon - closure(\delta(T_3, LETTER))$ = $\emptyset$

$\varepsilon - closure(\delta(T_3, WHITE\_SPACE))$ = $\emptyset$

$\varepsilon - closure(\delta(T_3, '))$ = $\varepsilon - closure(M)$ = T_6

$\varepsilon - closure(\delta(T_4, 0))$ = $\emptyset$

$\varepsilon - closure(\delta(T_4, NON - ZERO))$ = $\emptyset$

$\varepsilon - closure(\delta(T_4, LETTER))$ = $\emptyset$

$\varepsilon - closure(\delta(T_4, WHITE\_SPACE))$ = $\emptyset$

$\varepsilon - closure(\delta(T_4, '))$ = $\varepsilon - closure(M)$ = T_6

$\varepsilon - closure(\delta(T_5, 0))$ = $\emptyset$

$\varepsilon - closure(\delta(T_5, NON - ZERO))$ = $\emptyset$

$\varepsilon - closure(\delta(T_5, LETTER))$ = $\emptyset$

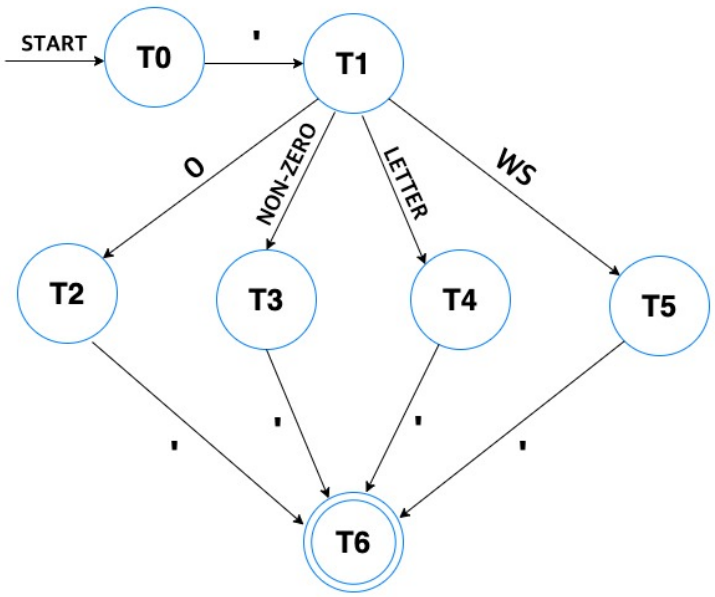
$\varepsilon - closure(\delta(T_5, WHITE\_SPACE))$ = $\emptyset$

$\varepsilon - closure\big(\delta(T_5, ')\big) = \varepsilon - closure(M) = T_6$
$\varepsilon - closure\big(\delta(T_6, 0)\big) = \emptyset$
$\varepsilon - closure\big(\delta(T_6, NON - ZERO)\big) = \emptyset$
$\varepsilon - closure\big(\delta(T_6, LETTER)\big) = \emptyset$
$\varepsilon - closure\big(\delta(T_6, WHITE\_SPACE)\big) = \emptyset$
$\varepsilon - closure\big(\delta(T_6, ')\big) = \emptyset$

- TRANSITION TABLE

	0	NON-ZERO	LETTER	WHITE_SPACE	' (QUOTE)
T_0	∅	∅	∅	∅	T_1
T_1	T_2	T_3	T_4	T_5	∅
T_2	∅	∅	∅	∅	T_6
T_3	∅	∅	∅	∅	T_6
T_4	∅	∅	∅	∅	T_6
T_5	∅	∅	∅	∅	T_6
T_6	∅	∅	∅	∅	∅

- DFA GRAPH



IV. IMPLEMENTATION

0. Developing Environmemt

Name	OS	Language	IDE
이나혁	Mac OS 11 Big Sur (Intel Processor)	Python3.6	PyCharm & Jupyter Notebook
이하윤	Mac OS 11 Big Sur (M1 Silicon)	Python3.8	PyCharm & VS Code

1. Define Tokens & Symbols

```
# define token & symbol
ALPHABET = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y',
'z', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z']
ZERO = ['0']
MINUS = ['-']
NON_ZERO = ['1', '2', '3', '4', '5', '6', '7', '8', '9']
UNDER_BAR = ['_']
ARITHMETIC = ['+', '-', '*', '/']
COMPARISON = ['<', '>', '!', '=']
ASSIGN = ['=']
QUOTE = ["'"]
DOUBLE_QUOTE = ['"']
SYMBOL = ['~', '@', '#', '$', '%', '^', '&', '?', '!', '|' , '₩₩', '.', ':']
COMMA = [',']
PARENS = ['(', ')', '[', ']', '{', '}']
SEMI = [';']
WHITE_SPACE = [' ', '₩t', '₩n']
LETTER = ALPHABET + SYMBOL
OTHERS = SYMBOL + COMMA + PARENS + WHITE_SPACE + SEMI + ASSIGN
VTYPE = ['int', 'char', 'boolean', 'String', 'void', 'byte', 'double', 'float',
         'long']
KEYWORD = ['abstract', 'break', 'case', 'catch', 'class', 'continue', 'default',
           'do', 'else', 'extends', 'false', 'finally', 'for', 'if', 'implements',
           'import', 'instanceof', 'interface', 'native', 'new', 'null', 'package',
           'private', 'protected', 'public', 'return', 'short', 'static', 'super',
           'switch', 'synchronized', 'this', 'throw', 'throws', 'true', 'try', 'while', 'args', 'main']
```

2. Implement DFA

앞서 그린 DFA를 구현한다. state를 정의하고 현재 state를 지정한다. 입력받은 token을 한 글자씩 split해 순차적으로 DFA에 입력한다. 순서대로 입력된 input을 끝까지 읽었을 때, 현재 state가 final state에 있으면 True를 반환(accept)하고 그렇지 않을 경우 False(reject)을 반환한다.

```
def isArithmetic(token):

    state = ['T0', 'T1', 'T2', 'T3', 'T4']
    locate = state[0]

    for value in token:
        if locate == state[0]:
            if value in ARITHMETIC[0]:
                locate = state[1]
            elif value in ARITHMETIC[1]:
                locate = state[2]
            elif value in ARITHMETIC[2]:
                locate = state[3]
            elif value in ARITHMETIC[3]:
                locate = state[4]
            else:
                return False
        else:
            return False

    if locate == state[1] or locate == state[2] or locate == state[3] or locate == state[4]:
        return True
    else:
        return False
```

위와 같은 알고리즘으로 arithmetic, comparison, identifier, literal string, signed integer, single character, others(그 외)를 판단하는 함수 isArithmetic(token), isComparison(token), isID(token), isLiteralString(token), isSignedINT(token), isSingleCharacter(token), isOthers(token)을 정의한다.

3. Check Double Quote(“) / Check Single Quote(‘)

Double Quote(“)로 둘러싸인 것과 Single Quote(‘)로 둘러싸인 것은 각각 String과 Char로 인식하여 하나의 Lexeme을 구성해야한다. 따라서 사전 전처리를 통해 이를 묶어줄 수 있도록 한다. 각각의 묶인 구성이 Literal String을 만족시키는지, 그리고 Character를 만족시키는지 확인하는 DFA 검사의 과정은 위에 정의한 함수들로 이루어지므로 이 과정에서 추가할 필요는 없다.

이에 따라 checkDoubleQuote(list), checkSingleQuote(list)의 두 함수를 구현하여 Input Stream을 list형으로 받아 전처리된 list를 반환하는 함수를 구성하였다. 두 함수는 거의 동일한 형태를 가지기 때문에 아래에 이 중 하나의 함수만 첨부하였다.

```
### Double Quote(")를 체크하여 List 내 Char 를 결합해주는 함수
def checkDoubleQuote(string_list: list):
  result = []
  temp = []
  checkOn = False
  for i in string_list:
```

```
    if i == '"':
      if checkOn == True:
        temp.append('"')
        result.append(''.join(temp))
        temp = []
        checkOn = False
      elif checkOn == False:
        checkOn = True
        temp.append('"')
    else:
      if checkOn == False:
        result.append(i)
      elif checkOn == True:
        temp.append(i)
  # List 반환
  return result
```

4. Input Stream Preprocessing

입력 Stream을 받아 이를 Char 단위로 쪼개어 List로 반환한다. 이 때, 위(3.)에서 정의한 두 함수를 이용하여 Quote에 대한 전처리도 함께 진행한다. 이 함수가 적용된 예시는 아래와 같다.

INPUT (String) ❯ 'cau 123 "good"'
OUTPUT (List) ❯ ['c', 'a', u', ' ', '1', '2', '3', ' ', '"good"']

```
### 입력 Stream 을 받아 이를 Char 단위로 쪼개어 List 반환 ("", '' 처리 포함)
def preProcessing(input_str: str):
  # 전역 변수 사용
  global initial_input
  # 결과 출력을 위한 Stream 복사
  initial_input = input_str
  # String to List
  test = list(input_str)
  # "", '' 처리 수행
  test = checkDoubleQuote(checkSingleQuote(test))
  # List 반환
  return test
```

5. Tokenize

구현의 가장 핵심적인 부분이다. 위 전처리 과정을 통해 String을 List로 변환하였고, 이 List가 곧 이 함수의 Input이 된다. 이 함수는 재귀(Recursion) 방식으로 구현된다. List를 입력받은 이 함수는 첫 원소부터 순차적으로 탐색해가며 Lexeme이 완성되면 그 값과 Token을 저장한다. 그리고 기존 List에서 해당 Lexeme에 대한 Char 원소들을 제거한 후 제거된 새 List를 Input으로 다시 Tokenize 함수를 실행한다. 이러한 과정이 재귀로 진행되며, Input Stream에 대한 List가 모든 Lexeme이 저장되어 최종적으로 비게 되었을 때 return하게 된다.

아래는 구현에 대한 세부적인 설명이다.

1) Tokenize 함수의 Return (재귀 종료)

위에서 설명했듯, 이 함수는 재귀 방식으로 동작하며 Lexeme을 추출하고 난 나머지 Char List를 새로운 Input으로 한다고 하였다. 그렇다면 언제 이 함수는 Return하느냐에 대한 부분이 이 부분이다. 분석해야할 모든 String, 즉 Code(String)에 존재하는 각각의 모든 Char가 분석되어 List가 비게 되면 이 함수는 return하게 된다.

```
# 더 이상 분석할 String 이 존재하지 않으면 tokenize 함수 반환
if len(input_string) == 0:

  return
```

2) 검사 대상 리스트 구현

순차적으로 Char를 읽으며 DFA 검사를 수행하고 Lexeme이 완성되었을 때 Token를 저장할 수 있어야 한다. 예를 들어 'if123' 라는 ID에 대한 검사가 수행이 된다면 최소한 6번('i', 'if', 'if1', 'if12', 'if123', 그리고 Reject)에 대한 검사가 반드시 진행되어야 한다. 이 단계는 이러한 검사 대상 리스트를 새롭게 제작하는 단계이다. 이 부분에 대한 예시는 아래와 같다.

BEGIN > ['a', 'b', 'c', '1']
END > [['a'], ['a', 'b'], ['a', 'b', 'c'], ['a', 'b', 'c', '1']]

```
# 반복문을 통해 검사할 Char 리스트를 재구성
# ex. 'abc1' -> [['a'], ['a','b'], ['a','b','c'], ['a','b','c','1']]
while True:
  test_iter = []
  for i in range(len(input_string)):
    test_iter.append(input_string[0:i + 1])
  break
```

이후, 이 List에 대한 반복 검사가 수행된다.

```
# 재귀 방식의 Char List 반복 검사
for i in test_iter:
  # 검사 횟수 Count
  count = count + 1
```

3) Lexeme 완성 시

먼저 DFA 검사를 하였을 때 각 검사에 대해 모두 Reject인 경우 바로 직전의 검사까지의 Char들로 Lexeme을 구성할 수 있다. Lexeme 구성을 마치면 새로운 Input Stream을 구성하게 되며 Loop를 탈출하게 된다.

```
# 모든 DFA 검사에서 Reject 되는 경우 반복문 break 후 다음 Char List 검사
if ((isArithmetic(temp_str) == False)
  and (isComparison(temp_str) == False)
  and (isID(temp_str) == False)
```

```
    and (isLiteralString(temp_str) == False)
    and (isSignedINT(temp_str) == False)
    and (isSingleCharacter(temp_str) == False)
    and (isOthers(temp_str) == False)):

    # Reject 직전의 Char 까지 하여 Lexeme 구성
    token = ''.join(i[0:-1])

    # 다음 Input Stream 구성을 위해 Count - 1 수행
    count = count - 1

    break
```

4) DFA 검사

Arithmetic, Comparison, ID, LiteralString, SignedINT, SingleCharacter, Others에 대한 DFA 검사를 수행한다. 그리고 각 검사 종류 별로 Lexeme이 완성되기 전까지 Accept된 횟수를 저장한다. 이는 (-) Symbol이 INT로써 사용될지 Operator로써 사용될지 구분하는 용, 혹은 다른 애매한 상황을 구분하는 용으로 사용될 수 있기 때문이다.

Arithmetic과 Comparison에 대한 검사는 간단하게 구현된다.

```
# Arithmetic DFA 검사
if (isArithmetic(temp_str) == True):
    count_Arith = count_Arith + 1
    token_type = 'ARITHMETIC'

# Comparision DFA 검사
if (isComparison(temp_str) == True):
    count_Compa = count_Compa + 1
    token_type = 'COMPARISON'
```

Identifier의 경우 ID로 일단 분류 되었지만 Java 언어에서 사용되는 예약어(Keyword) 혹은 데이터 타입을 명시한 VType일 수 있다. 따라서 추가 조치를 수행해주어야 한다.

```
# Identifier DFA 검사
if (isID(temp_str) == True):
    count_ID = count_ID + 1

    # 사전에 지정된 예약어(Keyword)인 경우
    if temp_str in KEYWORD:
        token_type = 'KEYWORD'

    # 데이터 타입(Value Type)인 경우
    elif temp_str in VTYPE:
        token_type = 'VTYPE'

    # 그 외의 경우 ID
    else:
        token_type = 'ID'
```

Literal String, Signed Integer, Single Character에 대한 검사는 간단하게 구현된다.

```
# Literal String DFA 검사
if (isLiteralString(temp_str) == True):
    count_Liter = count_Liter + 1
    token_type = 'LITERAL'

# Signed Integer DFA 검사
if (isSignedINT(temp_str) == True):
    count_SiINT = count_SiINT + 1
    token_type = 'INT'

# Single Character DFA 검사
if (isSingleCharacter(temp_str) == True):
    count_SiChr = count_SiChr + 1
    token_type = 'CHAR'
```

DFA 검사를 통해 Others로 분류된 Lexeme은 해당 value에 따라 무슨 Token인지 추가로 분류해줄 필요가 있다. 예를 들면 기호(Symbol), 콤마(Comma), 괄호(Parens, ex. []{}()), White Space, 세미콜론(Semi), 할당 연산자(=)이 있다.

```
# 그 외의 경우(Others)에 대한 DFA 검사
if (isOthers(temp_str) == True):
    count_Others = count_Other + 1
    token_type = 'OTHERS'

    # Others는 기호(Symbol), 콤마(Comma), 괄호(Parens, ex.[]{}()), White Space, 세미콜론(Semi), 할당 연산자(Assign,=)으로 구성
    OTHERS = SYMBOL + COMMA + PARENS + WHITE_SPACE + SEMI + ASSIGN

    # 사전에 지정된 기호(Symbol)인 경우
    if temp_str in SYMBOL:
        token_type = 'SYMBOR'

    # 사전에 지정된 콤마(Comma)인 경우
    elif temp_str in COMMA:
        token_type = 'COMMA'

    # 사전에 지정된 괄호 종류(Parens)인 경우
    elif temp_str in PARENS:
        token_type = 'PARENS'

    # White Space(' ', '₩n', '₩t')의 경우
    elif temp_str in WHITE_SPACE:
        token_type = 'WHITE_SPACE'

    # 세미콜론(;)의 경우
    elif temp_str in SEMI:
        token_type = 'SEMI'
```

```
  # 할당 연산자(=)의 경우
  elif temp_str in ASSIGN:
    token_type = 'ASSIGN'
```

5) 마지막 Lexeme 검사인 경우

마지막 Lexeme 검사인 경우 Reject이 발생되지 않을 수 있다. 이 경우 정상적으로 Accept된 DFA를 확인하여 Token의 타입을 확인한 후 저장하게 된다.

```
# Char List 의 마지막 검사의 경우
if (test_iter.index(i) == len(test_iter) - 1) and (True in [isArithmetic(temp_str),
isComparison(temp_str), isID(temp_str), isLiteralString(temp_str), isSignedINT(temp_str),
isSingleCharacter(temp_str), isOthers(temp_str)]):

  try:
    # (-) Symbol 처리, 연산자가 될 것인지, Integer 처리할 것인지 결정
    if (count_Arith >= 1) and (count_SiINT >= 1) and (token_key[-1] in ['INT', 'ID']):
      # -- Arithmatic DFA Accept 횟수와 Signed Interger DFA Accept 횟수가 모두 1 이상이고,
      # -- 바로 직전 Token 이 Int 혹은 ID 일 경우 연산자 처리
      token = temp_str[0]
      token_type = 'ARITHMETIC'
    else:
      token = temp_str

  # (-) Symbol 이 Input 의 맨 앞에 나온 경우 Out of List range 예외 처리
  except:
    token = temp_str
```

6) 모든 검사에 대해 Lexeme이 결정되지 않을 경우

모든 Char List, 그리고 각각의 DFA 검사에 대해 Lexeme이 결정되지 않을 수 있다. 이 경우 No Token 처리한다.

```
# 모든 DFA 검사에서 Reject 되어 Lexeme 분류가 이루어지지 않은 경우
if token == "":
  token = "NO TOKEN"
```

7) Lexeme 구성 시 새로운 Input List를 생성, 재귀 호출

Lexeme이 구성되면 Input List에서 해당 Lexeme을 구성하는데 사용되었던 Char들을 제거한 후 새로운 Input List를 생성한다. 그리고 이를 다시 tokenize() 함수에 넣는 재귀 호출을 진행하게 된다. 예를 들면 아래와 같다.

BEFORE > ['1', '2', '3', 'a', 'b', 'c', ';']
Store <INT, 123>
AFTER > ['a', 'b', 'c', ';'] => **tokenize(Recursion)!**

```
# 재귀 함수의 Input 이 될 새로운 Input String List 를 결정
try:
```

```
    if (count_Arith >= 1) and (count_SilNT >= 1) and (token_key[-1] in ['INT', 'ID']):
        token = temp_str[0]
        token_type = 'ARITHMETIC'

        new_test = input_string[1:]

    else:
        new_test = input_string[count:]

# (-) Symbol 이 Input 의 맨 앞에 나온 경우 Out of List range 예외 처리
except:
    new_test = input_string[count:]

# print("FINDME", input_string)

# 결정된 Lexeme 을 전역 변수로 선언된 List 에 삽입
token_value.append(token)

# 결정된 Token Type 을 전역 변수로 선언된 List 에 삽입
token_key.append(token_type)

# 재귀 방식으로 나머지 Input String 에 대한 Tokenize 계속 실행
tokenize(new_test)
```

6. Save Result

추출된 Token 정보를 저장하여 Output File을 생성한다. 이 때 저장되는 파일명은 <input_file_name>_output.txt'이다.

```
### 결과를 새로운 파일에 출력 및 저장하는 함수
def save_result():
    global token_value
    global token_key
    # Output File 에 대한 이름은 '<input_file_name>_output.txt'로 저장됨
    save = open(file_path + '_output.txt', 'w')
    for i, j in zip(token_value, token_key):
        # White Space 를 제외하고 저장
        if j != 'WHITE_SPACE':
            save.write("<" + j + ", " + i + ">₩n")
            # print("<"+j+",", i+">")
    print("Successfully token list saved")
    save.close
```

7. 전체 Process 실행

Input file은 Command에서 Input File명을 받아와 읽게 된다. 함수로 구성된 각 단계를 거친 후 전체 Lexical Analyzer는 종료된다.

```
# Command 내 인자 개수 확인
if len(sys.argv) != 2:
    print("Insufficient arguments")
    sys.exit()

# input File 열기
file_path = sys.argv[1]
f = open(file_path, 'r')

### 전체 Lexical Analyzer Flow 에 대한 Process 실행
def process(input_str: str):
    global token_value
    global token_key
    token_value = []
    token_key = []
    initial_input = ''
    # PreProcessing
    test = preProcessing(input_str)
    # Tokenize
    tokenize(test)
    # Save Result
    save_result()


# Command 에서 받은 Code 에 대한 Lexical Analize Process 호출
process(f.read())
```

8. (-) Symbol에 대한 Issue

- 기호는 INT Token의 한 Char로 간주될 수도 있고, 혹은 Operator로 간주될 수 있다. 상황에 따라 다르기 때문에 별도의 과정을 거쳤다. 아래는 전체 코드의 일부분이다. 조건문의 조건을 확인하면 count_Arith, count_SiINT가 모두 1 이상이고, 해당 Lexeme 추출 직전에 추출된 Lexeme의 종류가 Integer 혹은 Identifier인지 확인하고 있다.

여기서 count_Arith는 DFA에 대한 검사를 진행했을 때 Lexeme에 대한 Token을 결정짓기 전까지의 누적 Arith_DFA 검사에 대한 Accept 횟수이다. 마찬가지로 count_SiINT Ehgks Token 결정 전까지의 누적 Singed Int DFA 검사에 대한 Accept 횟수이다. - 심볼이 연산자로 분류되기 위해선 누적 횟수가 두 count에 대해 모두 1 이상이어야 한다는 실험 결과가 있었다.

```
try:
    # (-) Symbol 처리, 연산자가 될 것인지, Integer 처리할 것인지 결정
    if (count_Arith >= 1) and (count_SiINT >= 1) and (token_key[-1] in ['INT', 'ID']):
        # -- Arithmatic DFA Accept 횟수와 Signed Interger DFA Accept 횟수가 모두 1 이상이고,
        # -- 바로 직전 Token 이 Int 혹은 ID 일 경우 연산자 처리
```

```
      token = temp_str[0]
      token_type = 'ARITHMETIC'
    else:
      token = temp_str

# (-) Symbol 이 Input 의 맨 앞에 나온 경우 Out of List range 예외 처리
except:
    token = temp_str
```

V. Experiment

- Case 1 :

input	output
-0 0abc0 123if 123if0 ' ' a-1	<ARITHMETIC, -> <INT, 0> <INT, 0> <ID, abc0> <INT, 123> <KEYWORD, if> <INT, 123> <ID, if0> <CHAR, ' '> <ID, a> <ARITHMETIC, -> <INT, 1>

- Case 2:

input	output
001 0010 0010a0010 0010-10 0010--10	<INT, 0> <INT, 0> <INT, 1> <INT, 0> <INT, 0> <INT, 10> <INT, 0> <INT, 0> <INT, 10> <ID, a0010> <INT, 0> <INT, 0> <INT, 10> <ARITHMETIC, -> <INT, 10> <INT, 0> <INT, 0> <INT, 10> <ARITHMETIC, -> <INT, -10>

- Case 3:

input	output
int main(){char if123='1';int 0a=a+-1;return -0;}	<VTYPE, int> <KEYWORD, main> <PARENS, (> <PARENS,)> <PARENS, {> <VTYPE, char> <ID, if123> <ASSIGN, => <CHAR, '1'> <SEMI, ;> <VTYPE, int> <INT, 0> <ID, a> <ASSIGN, => <ID, a> <ARITHMETIC, +> <INT, -1> <SEMI, ;> <KEYWORD, return> <ARITHMETIC, -> <INT, 0> <SEMI, ;> <PARENS, }>

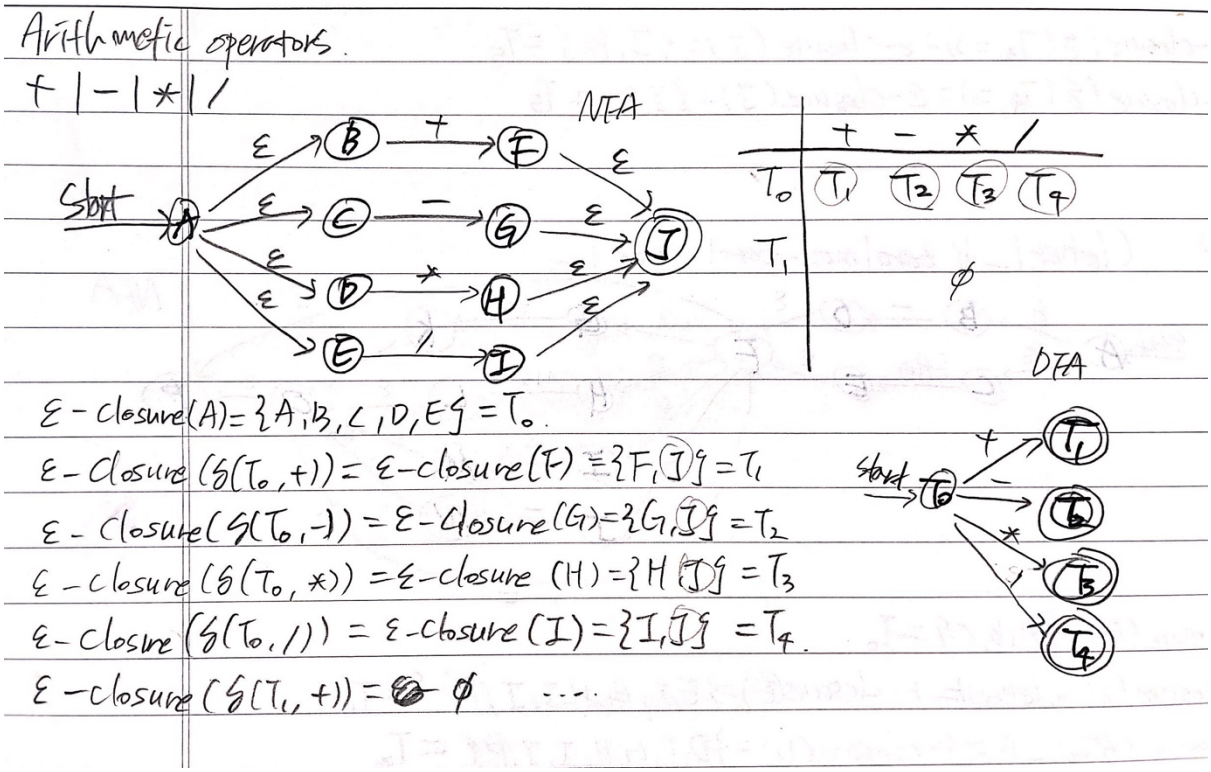
- Case 4:

input	output

package test;
public abstract class Test{
int a;
public Test(){
a=-10;
}
}
	<KEYWORD, package>
	<ID, test>
	<SEMI, ;>
	<KEYWORD, public>
	<KEYWORD, abstract>
	<KEYWORD, class>
	<ID, Test>
	<PARENS, {>
	<VTYPE, int>
	<ID, a>
	<SEMI, ;>
	<KEYWORD, public>
	<ID, Test>
	<PARENS, (>
	<PARENS,)>
	<PARENS, {>
	<ID, a>
	<ASSIGN, =>
	<INT, -10>
	<SEMI, ;>
	<PARENS, }>
	<PARENS, }>

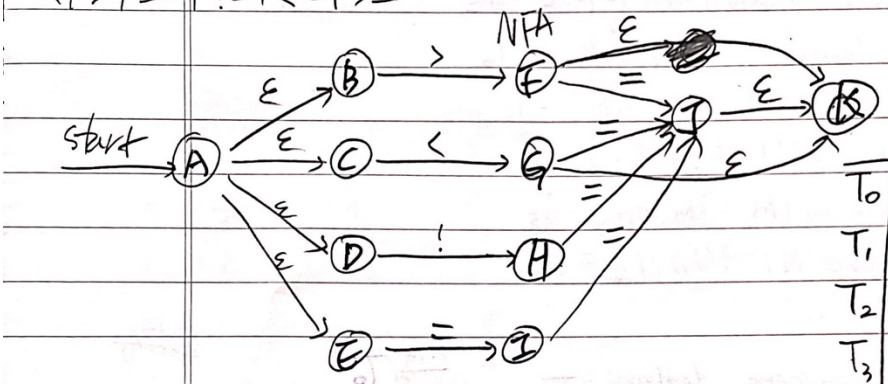
VI. Appendix

- NFA and DFA with transition table write down by hand



Comparasion operator

<|>|==|!=|<=|>=



	<	>	!	=
T_0	T_2	T_1	T_3	T_4
T_1	∅	∅	∅	T_5
T_2	∅	∅	∅	T_5
T_3	∅	∅	∅	T_5
T_4	∅	∅	∅	T_5
T_5	∅	∅	∅	∅

ε-closure(A) = {A, B, C, D, E} = T_0

ε-closure(δ(T_0, <)) = ε-closure({G}) = {G, K} = T_2

ε-closure(δ(T_0, >)) = ε-closure(F) = {F, K} = T_1

ε-closure(δ(T_0, !)) = ε-closure(H) = T_3

ε-closure(δ(T_0, =)) = ε-closure(I) = T_4

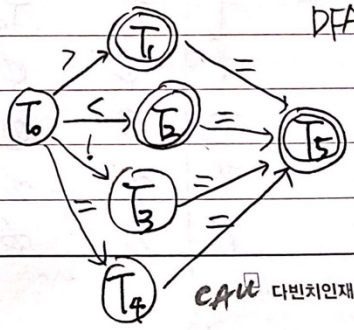
ε-closure(δ(T_1, =)) = ε-closure(J) = {J, K} = T_5

ε-closure(δ(T_2, =)) = ε-closure(J) = {J, K} = T_5

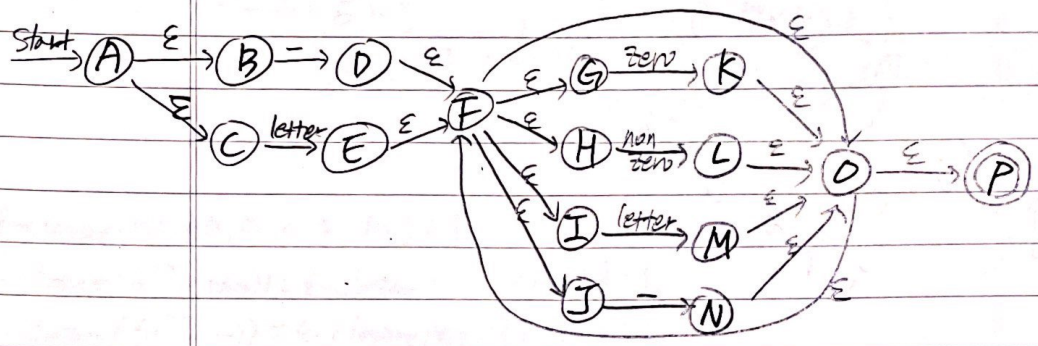
ε-closure(δ(T_3, =)) = ε-closure(J) = T_5

ε-closure(δ(T_4, =)) = ε-closure(J) = T_5

DFA



sID (letter|_)(zero | non-zero | letter | _ |)*



ε-closure(A) = {A, B, C} = T_0

ε-closure(δ(T_0, _)) = ε-closure(D) = {D, F, G, H, I, J, O, P} = T_1

ε-closure(δ(T_0, letter)) = ε-closure(E) = {E, F, G, H, I, J, O, P} = T_2

ε-closure(δ(T_1, zero)) = ε-closure(K) = {K, O, F, G, H, I, J, P} = T_3

ε-closure(δ(T_1, nz)) = ε-closure(L) = {L, O, F, G, H, I, J, P} = T_4

ε-closure(δ(T_1, letter)) = ε-closure(M) = {M, O, F, G, H, I, J, P} = T_5

ε-closure(δ(T_1, _)) = ε-closure(N) = {N, O, F, G, H, I, J, P} = T_6

ε-closure(δ(T_2, zero)) = ε-closure(K) = T_3

ε-closure(δ(T_2, nz)) = ε-closure(L) = T_4

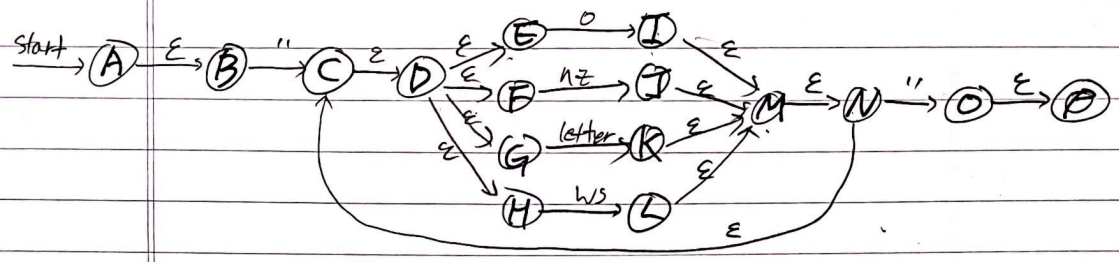
ε-closure(δ(T_2, letter)) = ε-closure(M) = T_5

ε-closure(δ(T_2, _)) = ε-closure(N) = T_6

T_3, T_4, T_5, T_6 동일 경우.

	letter	_	zero	non-zero
T_0	T_2	T_1	$\emptyset$	$\emptyset$
T_1	T_5	T_6	T_3	T_4
T_2	T_5	T_6	T_3	T_4
T_3	T_5	T_6	T_3	T_4
T_4	T_5	T_6	T_3	T_4
T_5	T_5	T_6	T_3	T_4
T_6	T_5	T_6	T_3	T_4

Literal String "(zero|non-zero|letter|WS)*"



ε-closure(A) = {A, B} = T_0

ε-closure(δ(T_0, ")) = {C, D, E, F, G, H} = T_1

ε-closure(δ(T_1, zero)) = {I, M, N, C, D, E, F, G, H} = T_2 ε-closure(I)

ε-closure(δ(T_1, nz)) = {J, M, N, C, D, E, F, G, H} = T_3 ε-closure(J)

ε-closure(δ(T_1, letter)) = {K, M, N, C, D, E, F, G, H} = T_4 ε-closure(K)

ε-closure(δ(T_1, WS)) = {L, M, N, C, D, E, F, G, H} = T_5 ε-closure(L)

ε-closure(δ(T_2, zero)) = ε-closure(I) = T_2

ε-closure(δ(T_2, nz)) = ε-closure(J) = T_3

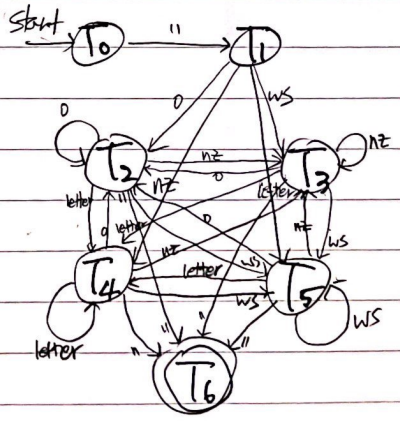
ε-closure(δ(T_2, letter)) = ε-closure(K) = T_4

ε-closure(δ(T_2, WS)) = ε-closure(L) = T_5

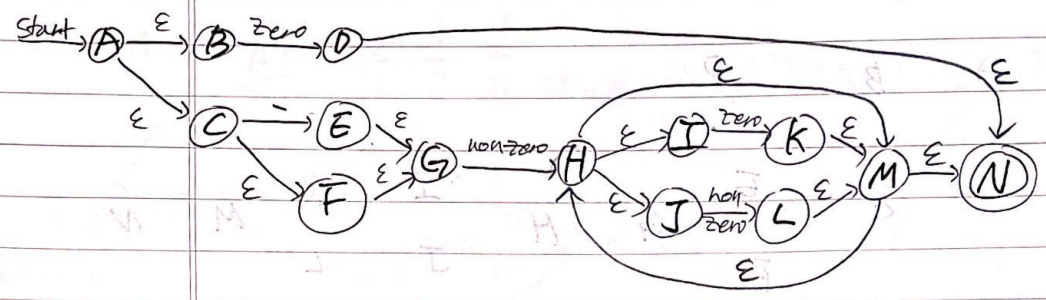
ε-closure(δ(T_2, ")) = ε-closure(O) = {O, P} = T_6

T_2, T_3, T_4, T_5 는 같은 case

	zero	non-zero	letter	WS	"
T_0	ϕ	ϕ	ϕ	ϕ	T_1
T_1	T_2	T_3	T_4	T_5	ϕ
T_2	T_2	T_3	T_4	T_5	T_6
T_3	T_2	T_3	T_4	T_5	T_6
T_4	T_2	T_3	T_4	T_5	T_6
T_5	T_2	T_3	T_4	T_5	T_6
T_6	ϕ	ϕ	ϕ	ϕ	ϕ



Signed Integer zero | ((- | ε) non-zero (zero | non-zero)*)



ε-closure(A) = {A, B, C, F, G} = T_0

ε-closure(δ(T_0, zero)) = ε-closure(D) = {D, N} = T_1

ε-closure(δ(T_0, -)) = ε-closure(E) = {E, G} = T_2

ε-closure(δ(T_0, non-zero)) = ε-closure(H) = {H, I, J, M, N} = T_3

ε-closure(δ(T_2, non-zero)) = ε-closure(H) = T_3

ε-closure(δ(T_3, zero)) = ε-closure(K) = {K, M, N, H, I, J} = T_4

ε-closure(δ(T_3, non-zero)) = ε-closure(L) = {L, M, N, H, I, J} = T_5

ε-closure(δ(T_4, zero)) = ε-closure(K) = T_4

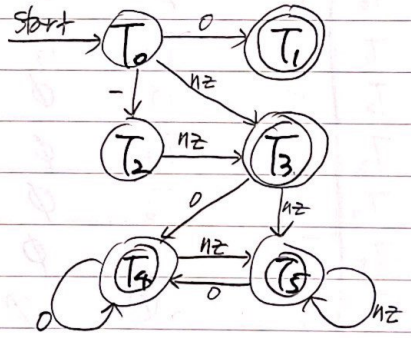
ε-closure(δ(T_4, non-zero)) = ε-closure(L) = T_5

ε-closure(δ(T_5, zero)) = ε-closure(K) = T_4

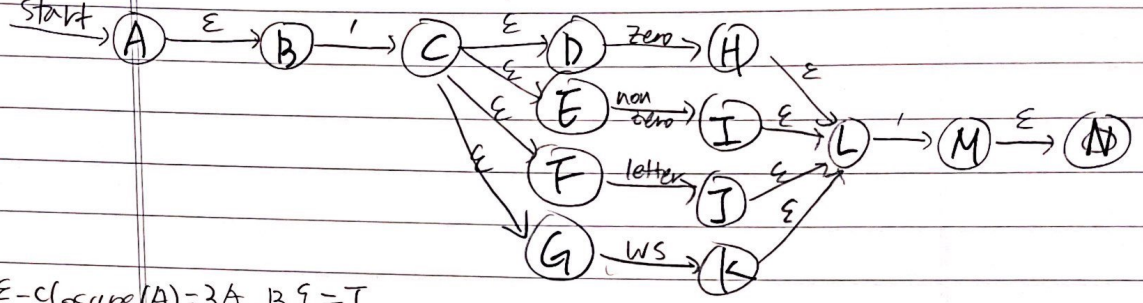
ε-closure(δ(T_5, non-zero)) = ε-closure(L) = T_5

	zero	non-zero	-(minus)
T_0	T_1	T_3	T_2
T_1	∅	∅	∅
T_2	∅	T_3	∅
T_3	T_4	T_5	∅
T_4	T_4	T_5	∅
T_5	T_4	T_5	∅

DFA



Single character `(zero | non-zero | letter | ws)'



ε-closure(A) = {A, B} = T_0

ε-closure(δ(T_0, ')) = ε-closure(C) = {C, D, E, F, G} = T_1

ε-closure(δ(T_1, zero)) = ε-closure(H) = {H, L} = T_2

ε-closure(δ(T_1, nz)) = ε-closure(I) = {I, L} = T_3

ε-closure(δ(T_1, letter)) = ε-closure(J) = {J, L} = T_4

ε-closure(δ(T_1, ws)) = ε-closure(K) = {K, L} = T_5

ε-closure(δ(T_2, ')) = ε-closure(M) = {M, N} = T_6

ε-closure(δ(T_3, ')) = ε-closure(M) = T_6

ε-closure(δ(T_4, ')) = ε-closure(M) = T_6

ε-closure(δ(T_5, ')) = ε-closure(M) = T_6

	zero	non-zero	letter	ws	'
T_0	ϕ	ϕ	ϕ	ϕ	T_1
T_1	T_2	T_3	T_4	T_5	ϕ
T_2	ϕ	ϕ	ϕ	ϕ	T_6
T_3	ϕ	ϕ	ϕ	ϕ	T_6
T_4	ϕ	ϕ	ϕ	ϕ	T_6
T_5	ϕ	ϕ	ϕ	ϕ	T_6
T_6	ϕ	ϕ	ϕ	ϕ	ϕ