2021-1 Compiler class 02 (Prof. Hyo su Kim)

Project #1 Lexical Analyzer

2021.04.18

Student

Software department

Team members : 배인경(20190323) 좌민주(20190912)

Content

#1 LEXICAL SPECIFICATIONS

Variable type
- *int* for a signed integer
- *char* for a single character
- *boolean* for a Boolean string
- *String* for a literal string

Signed integer
- A single zero digit (e.g., 0)
- A non-empty sequence of digits, starting from a non-zero digit
 (e.g., 1, 22, 123, 56, … any non-zero positive integers)
 (e.g., 001 is not allowed)
- A non-empty sequence of digits, starting from a minus sign symbol and a non-zero digit
 (e.g., -1, -22, -123, -56, .. any non-zero negative integers)

Single character
- A single digit, English letter, block starting from and terminating with a symbol ‘
 (e.g., ‘a’, ‘1’, ‘ ‘)

Boolean string : true and false

Literal string
- Any combination of digits, English letters, and blanks, starting from and terminating with a symbol “
 (e.g., “Hello world”, “My student id is 12345678”)

An identifier of variables and functions
- A non-empty sequence of English letters, digits, and underscore symbols, starting from an English letter or a underscore symbol
(e.g., i, j, k, abc, ab_123, func1, func_, __func_bar__)

Keywords for special statements
- *if* for if statement
- *else* for else statement
- *while* for while-loop statement
- *class* for class statement
- *return* for return statement

Arithmetic operators : +, -, *, and /

Assignment operator : =

Comparison operators : <, >, ==, !=, <=, and >=

A terminating symbol of statements : ;

A pair of symbols for defining area/scope of variables and functions : { and }

A pair of symbols for indicating a function/statement : (and)

A pair of symbols for using an array : [and]

A symbol for separating input arguments in functions : ,

Whitespaces : a non-empty sequence of \t, \n, and blanks

#2 TOKEN

Token	Lexeme
TYPE	int, char, boolean, String
INTEGER	-2, -1, 0, 1, 2, 3, 4, 1000
CHARACTER	‘a’, ‘3’, ‘ ‘ ...
BOOLEAN	true, false
LITERAL	“Compiler Term Project”
IDENTIFIER	identifier1, _id, lexical_analyzer__
KEYWORD	if, else, while, class, return
OPERATOR	+, - , *, /
ASSIGN	=
RELOP	<, >, ==, !=, <=, >=
SEMICOLON	;
LBRACE	{
RBRACE	}
LPAREN	(
RPAREN	)
LARRAY	[
RARRAY	]
COMMA	,
WHITESPACE	'\t', ' ', '\n'

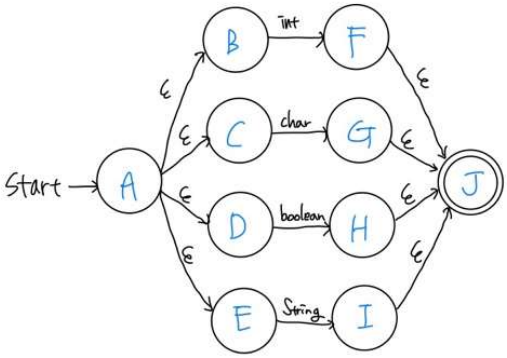
> REGULAR EXPRESSION, NFA, TRANSITION TABLE, DFA for each token (By hand)

Token 1. TYPE

1) Regular Expression

type → int | char | boolean | String

2) NFA



3) Transition Table

T_0 = ε-closure(A) = {A, B, C, D, E}

T_1 = ε-(f(T_0, int)) = ε-(F) = {F, J}

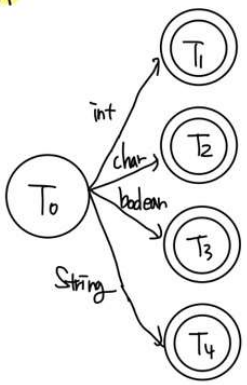
T_2 = ε-(f(T_0, char)) = ε-(G) = {G, J}

T_3 = ε-(f(T_0, boolean)) = ε-(H) = {H, J}

T_4 = ε-(f(T_0, String)) = ε-(I) = {I, J}

	int	char	boolean	String
T_0	T_1	T_2	T_3	T_4
T_1	∅	∅	∅	∅
T_2	∅	∅	∅	∅
T_3	∅	∅	∅	∅
T_4	∅	∅	∅	∅

4) DFA

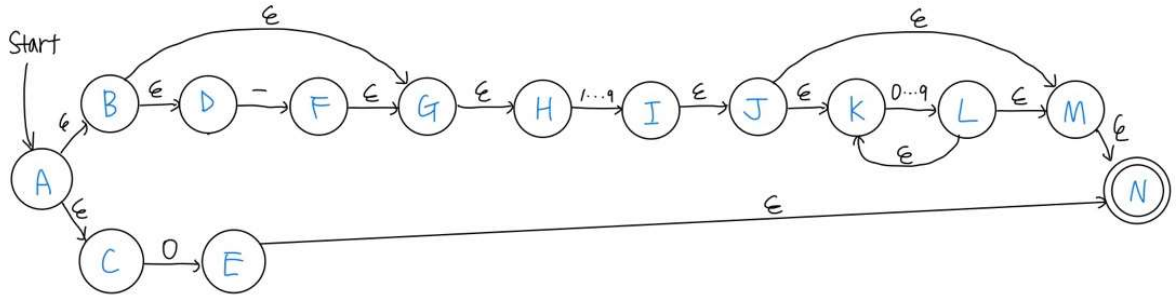


Token 2. INTEGER

1) Regular Expression

integer → ((−|ε)(1|2|3|4|5|6|9|8|9) digit* | 0)

2) NFA



3) Transition Table

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

$T_1 = \epsilon\text{-}(\delta(T_0, -)) = \epsilon\text{-}(F) = \{F, G, H\}$

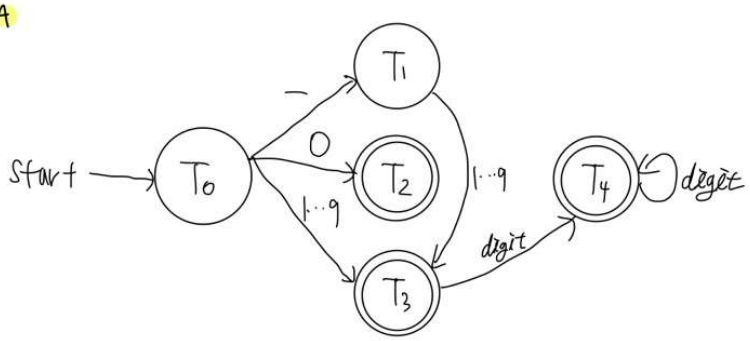
$T_2 = \epsilon\text{-}(\delta(T_0, 0)) = \epsilon\text{-}(E) = \{E, N\}$

$T_3 = \epsilon\text{-}(\delta(T_0, 1\cdots9)) = \epsilon\text{-}(I) = \{I, J, K, M, N\}$

$T_4 = \epsilon\text{-}(\delta(T_3, \text{digit})) = \epsilon\text{-}(L) = \{K, M, N\}$

	—	0	1···9	digit
T_0	T_1	T_2	T_3	∅
T_1	∅	∅	T_3	∅
T_2	∅	∅	∅	∅
T_3	∅	∅	∅	T_4
T_4	∅	∅	∅	T_4

4) DFA

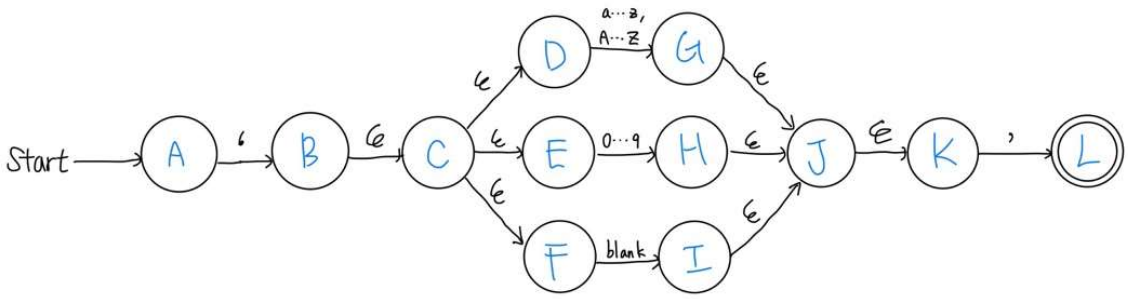


Token3 . CHARACTER

1) Regular Expression

character → ' (letter | digit | blank) '

2) NFA



3) Transition Table

$T_0 = \varepsilon\text{-}(A) = \{A\}$

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

$T_2 = \varepsilon\text{-}(\delta(T_1, \text{letter})) = \varepsilon\text{-}(G) = \{G, J, K\}$

$T_3 = \varepsilon\text{-}(\delta(T_1, \text{digit})) = \varepsilon\text{-}(H) = \{H, J, K\}$

$T_4 = \varepsilon\text{-}(\delta(T_1, \text{blank})) = \varepsilon\text{-}(I) = \{I, J, K\}$

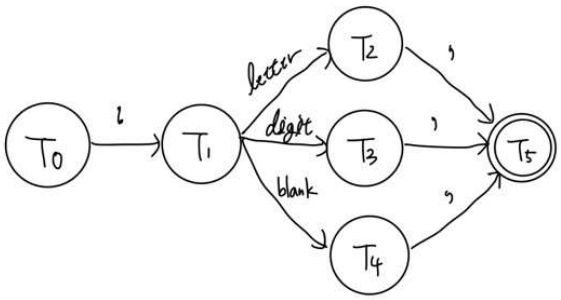
$T_5 = \varepsilon\text{-}(\delta(T_2, ')')) = \varepsilon\text{-}(L) = \{L\}$

$\varepsilon\text{-}(\delta(T_3, ')')) = \varepsilon\text{-}(L) = \{L\} = T_5$

$\varepsilon\text{-}(\delta(T_4, ')')) = \varepsilon\text{-}(L) = \{L\} = T_5$

	'	letter	digit	blank
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) DFA

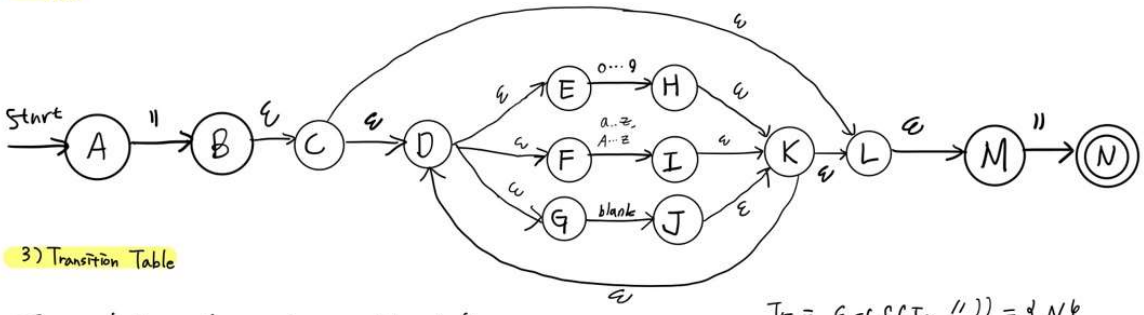


Token4. LITERAL

1) Regular Expression

literal → " (digit | letter | blank)* "

2) NFA



3) Transition Table

$\Sigma = \{$ " , digit, letter, blank $\}$

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

$T_1 = \epsilon\text{-}(\delta(T_0, \text{"})) = \{B, C, D, E, F, G, L, M\}$

$T_2 = \epsilon\text{-}(\delta(T_1, \text{digit})) = \{H, K, L, M, D, E, F, G\}$

$T_3 = \epsilon\text{-}(\delta(T_1, \text{letter})) = \{I, K, L, M, D, E, F, G\}$

$T_4 = \epsilon\text{-}(\delta(T_1, \text{blank})) = \{J, K, L, M, D, E, F, G\}$

$\epsilon\text{-}(\delta(T_2, \text{digit})) = \{H, K, L, M, D, E, F, G\} = T_2$

$\epsilon\text{-}(\delta(T_2, \text{letter})) = T_3$

$\epsilon\text{-}(\delta(T_2, \text{blank})) = T_4$

$T_5 = \epsilon\text{-}(\delta(T_2, \text{"})) = \{N\}$

$\epsilon\text{-}(\delta(T_3, \text{"})) = T_5$

$\epsilon\text{-}(\delta(T_3, \text{digit})) = T_2$

$\epsilon\text{-}(\delta(T_3, \text{letter})) = T_3$

$\epsilon\text{-}(\delta(T_3, \text{blank})) = T_4$

$\epsilon\text{-}(\delta(T_3, \text{"})) = \{N\}$

$\epsilon\text{-}(\delta(T_4, \text{"})) = \{N\}$

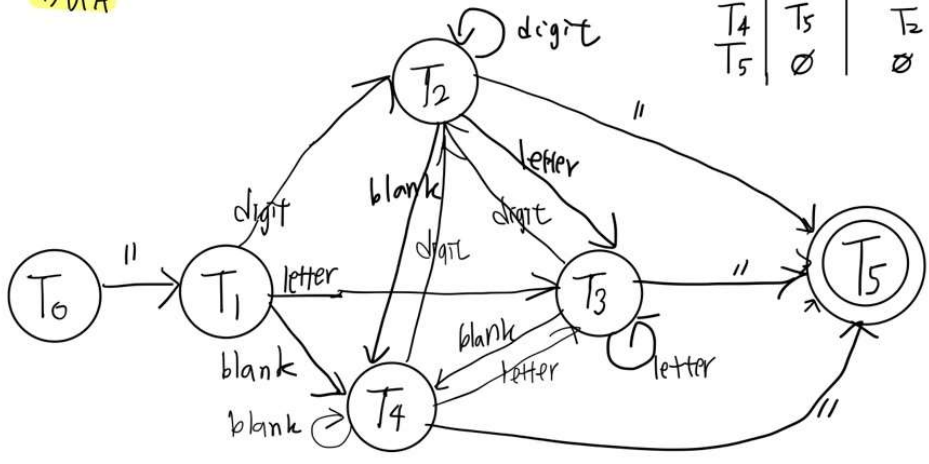
$\epsilon\text{-}(\delta(T_4, \text{digit})) = T_2$

$\epsilon\text{-}(\delta(T_4, \text{letter})) = T_3$

$\epsilon\text{-}(\delta(T_4, \text{blank})) = T_4$

	"	digit	letter	blank
T_0	T_1	$\emptyset$	$\emptyset$	$\emptyset$
T_1	T_5	T_2	T_3	T_4
T_2	T_5	T_2	T_3	T_4
T_3	T_5	T_2	T_3	T_4
T_4	T_5	T_2	T_3	T_4
T_5	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

4) DFA

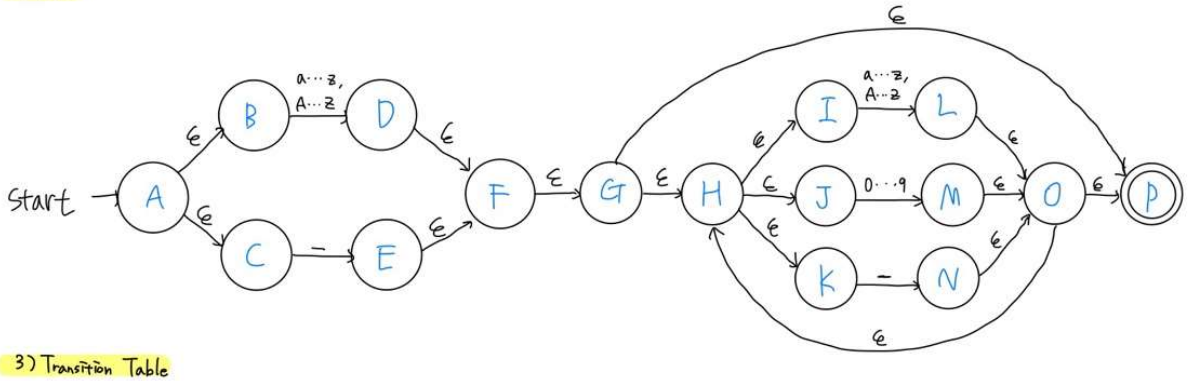


Token 5. IDENTIFIER

1) Regular Expression

identifier → (letter | _) (letter | digit | _)*

2) NFA



3) Transition Table

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

$T_1 = \varepsilon\text{-}(\delta(T_0, letter)) = \varepsilon\text{-}(D) = \{D, F, G, H, I, J, K, P\}$

$T_2 = \varepsilon\text{-}(\delta(T_0, \_)) = \varepsilon\text{-}(E) = \{E, F, G, H, I, J, K, P\}$

$T_3 = \varepsilon\text{-}(\delta(T_1, letter)) = \varepsilon\text{-}(L) = \{H, I, J, K, L, O, P\}$

$T_4 = \varepsilon\text{-}(\delta(T_1, digit)) = \varepsilon\text{-}(M) = \{H, I, J, K, M, O, P\}$

$T_5 = \varepsilon\text{-}(\delta(T_1, \_)) = \varepsilon\text{-}(N) = \{H, I, J, K, N, O, P\}$

$\varepsilon\text{-}(\delta(T_2, letter)) = \varepsilon\text{-}(L) = T_3$

$\varepsilon\text{-}(\delta(T_2, digit)) = \varepsilon\text{-}(M) = T_4$

$\varepsilon\text{-}(\delta(T_2, \_)) = \varepsilon\text{-}(N) = T_5$

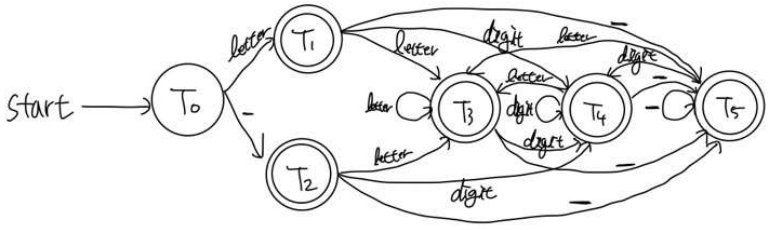
$\varepsilon\text{-}(\delta(T_3, letter)) = T_3 \;/\; \varepsilon\text{-}(\delta(T_3, digit)) = T_4 \;/\; \varepsilon\text{-}(\delta(T_3, \_)) = T_5$

$\varepsilon\text{-}(\delta(T_4, letter)) = T_3 \;/\; \varepsilon\text{-}(\delta(T_4, digit)) = T_4 \;/\; \varepsilon\text{-}(\delta(T_4, \_)) = T_5$

$\varepsilon\text{-}(\delta(T_5, letter)) = T_3 \;/\; \varepsilon\text{-}(\delta(T_5, digit)) = T_4 \;/\; \varepsilon\text{-}(\delta(T_5, \_)) = T_5$

	letter	digit	_
T_0	T_1	$\emptyset$	T_2
T_1	T_3	T_4	T_5
T_2	T_3	T_4	T_5
T_3	T_3	T_4	T_5
T_4	T_3	T_4	T_5
T_5	T_3	T_4	T_5

4) DFA

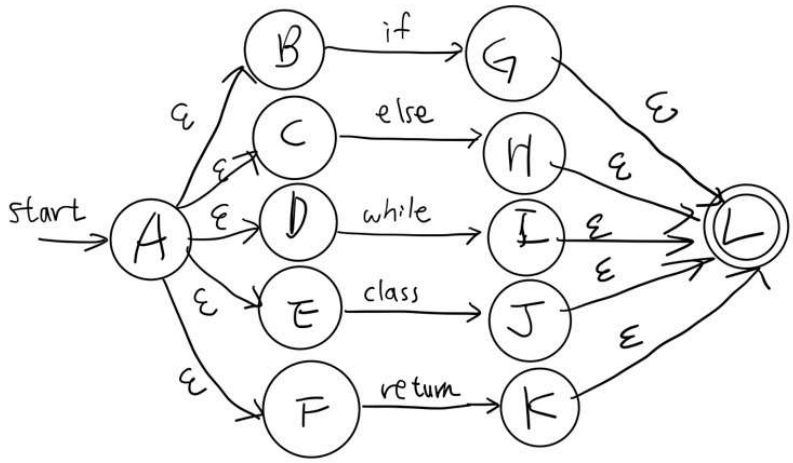


Token 6. KEYWORD

1) Regular Expression

keyword → if | else | while | class | return

2) NFA



3) Transition Table

Σ = { if, else, while, class, return }

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

$T_1 = \epsilon\text{-}(T_0, \text{if}) = \{G, L\}$

$T_2 = \epsilon\text{-}(T_0, \text{else}) = \{H, L\}$

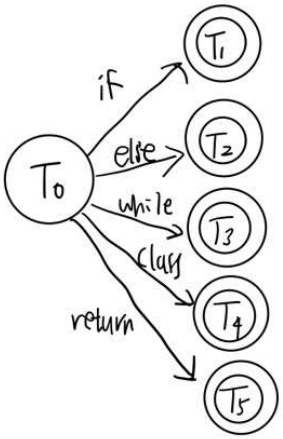
$T_3 = \epsilon\text{-}(T_0, \text{while}) = \{I, L\}$

$T_4 = \epsilon\text{-}(T_0, \text{class}) = \{J, L\}$

$T_5 = \epsilon\text{-}(T_0, \text{return}) = \{K, L\}$

	if	else	while	class	return
T_0	T_1	T_2	T_3	T_4	T_5
T_1	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
T_2	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
T_3	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
T_4	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
T_5	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

4) DFA

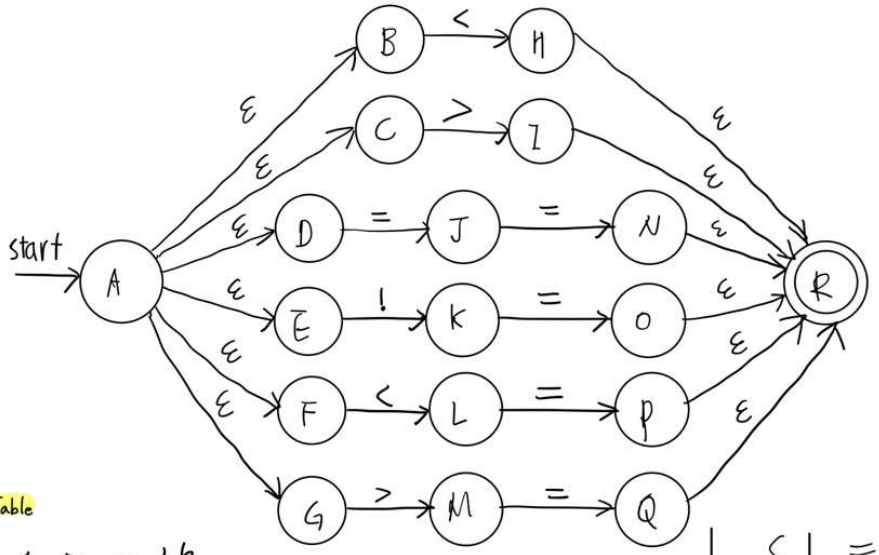


Token 7. RELOP

1) Regular Expression

relop → < | > | == | != | <= | >=

2) NFA



3) Transition Table

Σ = { <, >, =, ! }

T_0 = ε-(A) = { A, B, C, D, E, F, G }

T_1 = ε-(δ(T_0, <)) = { H, I, L, R }

T_2 = ε-(δ(T_0, =)) = { J }

T_3 = ε-(δ(T_0, !)) = { K }

T_4 = ε-(δ(T_0, >)) = { I, M, R }

T_5 = ε-(δ(T_1, =)) = { P, R }

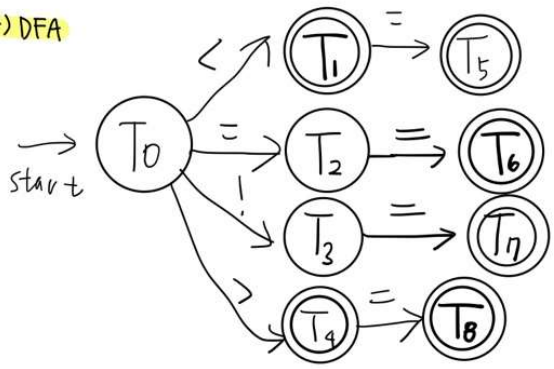
T_6 = ε-(δ(T_2, =)) = { N, R }

T_7 = ε-(δ(T_3, =)) = { O, R }

T_8 = ε-(δ(T_4, =)) = { Q, R }

	<	=	!	>
T_0	T_1	T_2	T_3	T_4
T_1	∅	T_5	∅	∅
T_2	∅	T_6	∅	∅
T_3	∅	T_7	∅	∅
T_4	∅	T_8	∅	∅
T_5	∅	∅	∅	∅
T_6	∅	∅	∅	∅
T_7	∅	∅	∅	∅
T_8	∅	∅	∅	∅

4) DFA

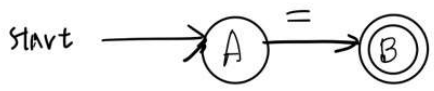


Token 8 ASSIGN

1) Regular Expression

assign → =

2) NFA



3) Transition Table

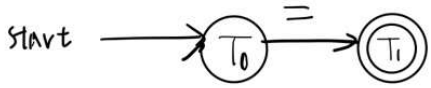
$\Sigma = \{=\}$

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

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

	=
T_0	T_1
T_1	$\emptyset$

4) DFA

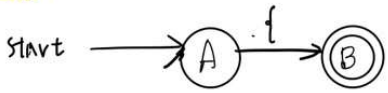


Token 9. LBRACE

1) Regular Expression

lbrace → {

2) NFA



3) Transition Table

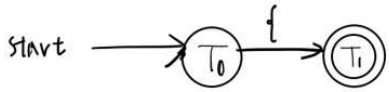
$\Sigma = \{\{\}$

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

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

	{
T_0	T_1
T_1	$\emptyset$

4) DFA

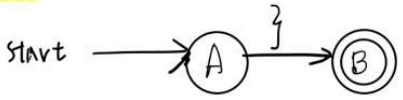


Token 10 R BRACE

1) Regular Expression

rbrace → }

2) NFA



3) Transition Table

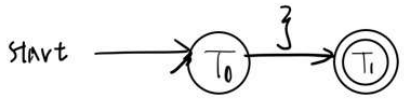
$\Sigma = \{\}\}$

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

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

	}
T_0	T_1
T_1	$\emptyset$

4) DFA

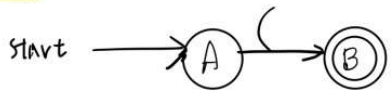


Token 11 LPAREN

1) Regular Expression

lparen → (

2) NFA



3) Transition Table

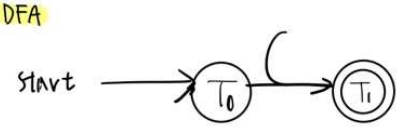
$\Sigma = \{ (\}$

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

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

	(
T_0	T_1
T_1	$\emptyset$

4) DFA

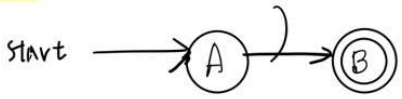


Token 12. RPAREN

1) Regular Expression

rparen →)

2) NFA



3) Transition Table

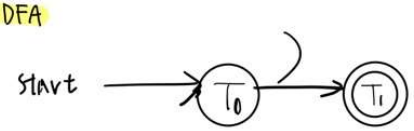
$\Sigma = \{) \}$

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

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

	)
T_0	T_1
T_1	$\emptyset$

4) DFA

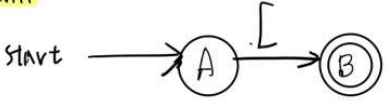


Token 13. LARRAY

1) Regular Expression

larray → [

2) NFA



3) Transition Table

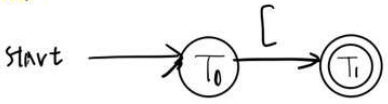
$\Sigma = \{ [\}$

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

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

	[
T_0	T_1
T_1	$\emptyset$

4) DFA

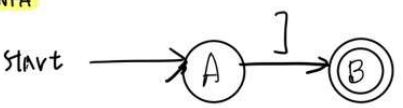


Token 14 RARRAY

1) Regular Expression

rarray →]

2) NFA



3) Transition Table

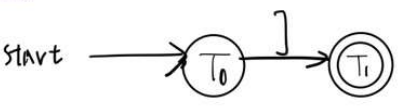
$\Sigma = \{] \}$

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

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

	]
T_0	T_1
T_1	$\emptyset$

4) DFA

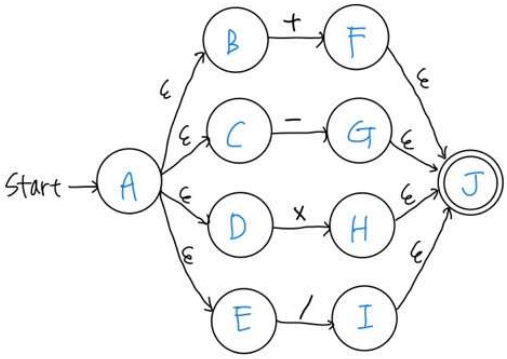


Token 15. OPERATOR

1) Regular Expression

operator → + | − | * | /

2) NFA



3) Transition Table

T_0 = ε-closure(A) = {A, B, C, D, E}

T_1 = ε-(δ(T_0, +)) = ε-(F) = {F, J}

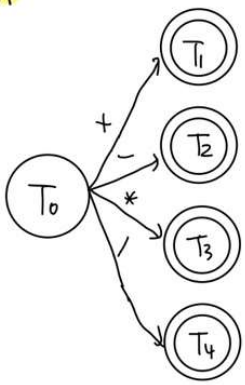
T_2 = ε-(δ(T_0, −)) = ε-(G) = {G, J}

T_3 = ε-(δ(T_0, *)) = ε-(H) = {H, J}

T_4 = ε-(δ(T_0, /)) = ε-(I) = {I, J}

	+	−	*	/
T_0	T_1	T_2	T_3	T_4
T_1	∅	∅	∅	∅
T_2	∅	∅	∅	∅
T_3	∅	∅	∅	∅
T_4	∅	∅	∅	∅

4) DFA

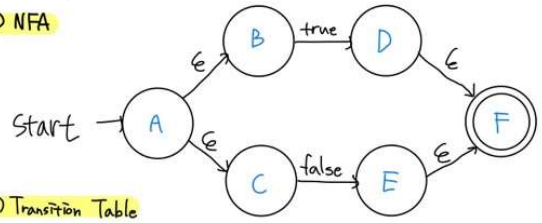


Token16. BOOLEAN

1) Regular Expression

boolean → true | false

2) NFA



3) Transition Table

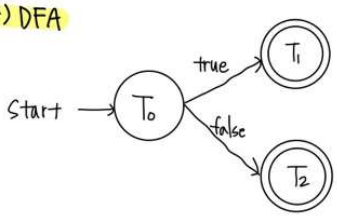
$T_0 = \epsilon\text{-closure}(A) = \{A, B, C\}$

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

$T_2 = \epsilon\text{-closure}(\delta(T_0, \text{false})) = \epsilon\text{-closure}(E) = \{E, F\}$

	true	false
T_0	T_1	T_2
T_1	$\emptyset$	$\emptyset$
T_2	$\emptyset$	$\emptyset$

4) DFA

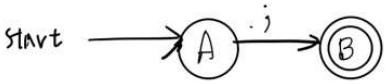


Token17. SEMICOLON

1) Regular Expression

Semicolon → ;

2) NFA



3) Transition Table

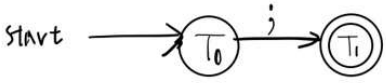
$\Sigma = \{ ; \}$

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

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

	;
T_0	T_1
T_1	$\emptyset$

4) DFA

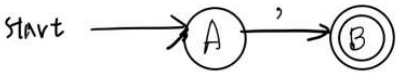


Token18. COMMA

1) Regular Expression

comma → ,

2) NFA



3) Transition Table

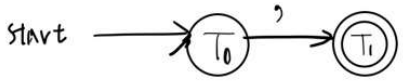
$\Sigma = \{ , \}$

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

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

	,
T_0	T_1
T_1	$\emptyset$

4) DFA



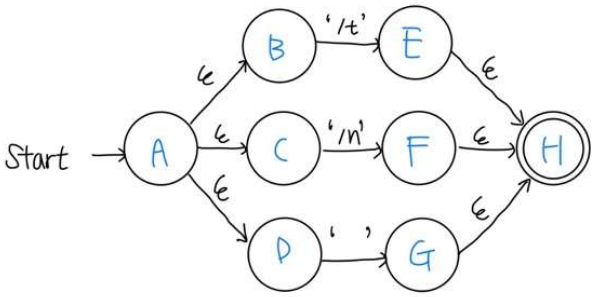
> WHITESPACE 는 우리가 정의한 Token에는 있지만 .out 파일에는 조건에 따라 Token으로 출력하지 않음.

Token 19. WHITESPACE

1) Regular Expression

whitespace → '/t' | '/n' | ' '

2) NFA



3) Transition Table

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

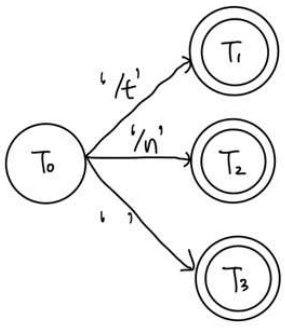
$T_1 = \epsilon\text{-closure}(\delta(T_0, \text{'/t'})) = \epsilon\text{-closure}(E) = \{E, H\}$

$T_2 = \epsilon\text{-closure}(\delta(T_0, \text{'/n'})) = \epsilon\text{-closure}(F) = \{F, H\}$

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

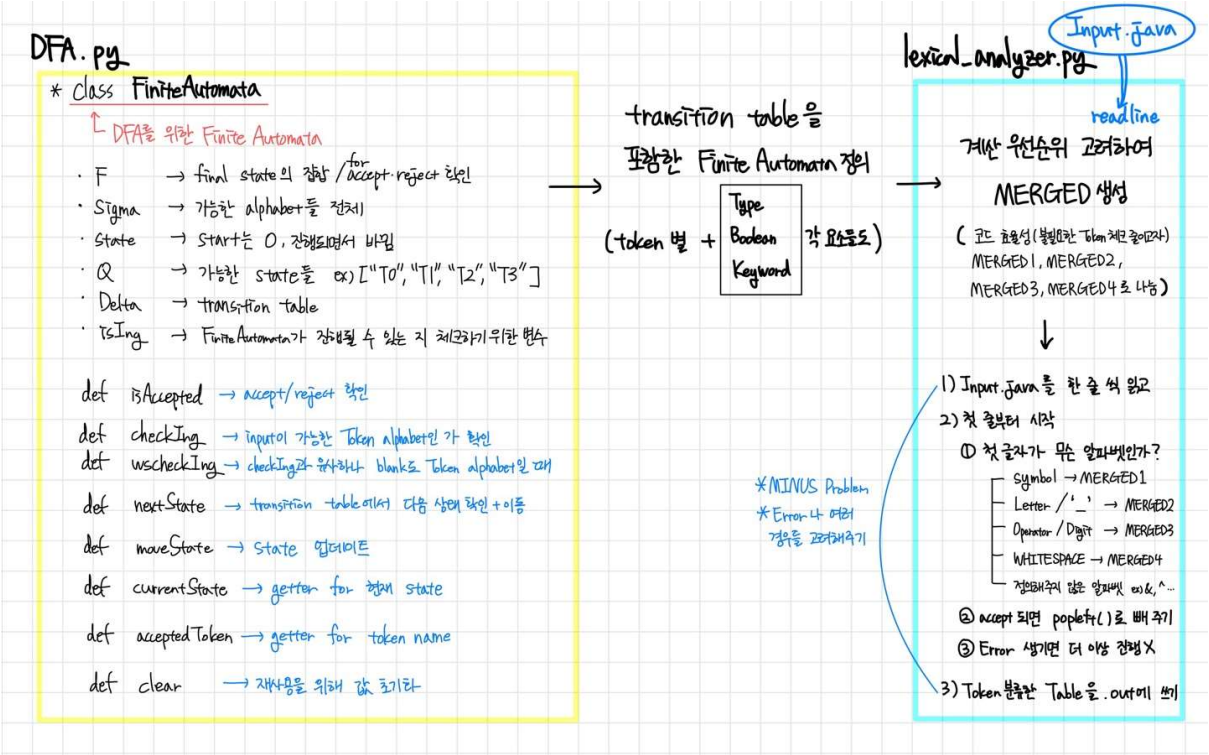
	'/t'	'/n'	' '
T_0	T_1	T_2	T_3
T_1	∅	∅	∅
T_2	∅	∅	∅
T_3	∅	∅	∅

4) DFA



#3 IMPLEMENTATION

1) OVERALL Procedures



2) Detail (DFA.py, lexical_analyzer.py)

DFA.py

 # 우리가 분류할 token은 아니지만 쓰이는 Alphabet들 정의

```
15  # Alphabet Definition
16  LETTER = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U',
17            'V', 'W', 'X', 'Y', 'Z',
18            'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u',
19            'v', 'w', 'x', 'y', 'z']
20  DIGIT = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
21  POS_DIGIT = DIGIT[1:]
22  BLANK = [' ']
```

 # class FiniteAutomata() : DFA를 위한 FiniteAutoma 클래스

1) 클래스 변수 종류

```
24  class FiniteAutomata:  # Finite Automata for DFA
25      tokenName = ""
26      F = {}      # a set of final(or accepting) states
27      Sigma = []  # a finite set of input symbols
28      state = 0  # a start state q0
29      Q = []  # a finite set of states
30      Delta = {}  # a set of state transition functions
31      isIng = 1  # a flag that saves FA is working
32
33      def __init__(self, token, states, alphabet, final, table):
34          self.tokenName = token
35          self.Q = states
36          self.Sigma = alphabet
37          self.state = "T0"
38          self.F = final
39          self.Delta = self.transition(table)
```

: tokenName => token이름
: F => final states
: Sigma => 각 토큰에서 가능한 input alphabet들 집합
: state => 현재상태
: Q => DFA에서 나오는 state들 종류
: Delta => transition functions
: isIng => FA가 계속 진행되는 지 확인하기 위한 변수

2) 클래스 함수 종류
: isAccepted => accept / reject 확인

```
41      def isAccepted(self):  # return true when current lexeme is matched with this token
42          if self.state in self.F:
43              return True
44          else:
45              return False
```

: checkIng & wscheckIng => 인자로 들어온 input이 self.Sigma에 있는 지 확인
=> 공백이 self.Sigma에 있으면 wscheckIng

```
47      def checkIng(self, input):...
70
71      def wscheckIng(self, input):...
```

: nextState => transition table을 확인해서 다음 상태로 이동할 수 있는 지 확인하고 이동시킴

```
92      def nextState(self, input):  # look transition table and move to nest state
93          items = self.Delta[self.state].items()
94          posNext = dict()
95          for key, value in items:
96              if value != "":
97                  posNext[key] = value
98          keys = posNext.keys()
99          newKey = ""
100         inputStr = ""
101
102         for i in keys:
103             if len(i) > 1:
104                 i = i.strip('][').replace("'", "").split(', ')
105             if input in i:
106                 newKey = str(i)
107
108         if newKey in posNext:
109             nextState = posNext[newKey]
110         else:
111             self.isIng = 0
112             return self.isIng
113         self.state = nextState
114         return self.isIng
```

: currentState, acceptedToken => getter 역할

```
119     def currentState(self, next_state):  # get currentState
120         return self.state
121
122     def acceptedToken(self):  # get token name
123         if self.state in self.F:
124             return self.tokenName
125         else:
126             return "Not Accepted"
```

: clear => FA 재사용을 위한 클래스 변수 초기화

```
128     def clear(self):  # clear to reuse the automata
129         self.state = "T0"
130         self.isIng = 1
131         return
```

 # 객체 선언
각 토큰 별로 FiniteAutomata 객체 선언 모두 해 준다.
이 때, TYPE & BOOLEAN & KEYWORD에 해당하는 char, int, boolean, String, true, false, if, else, while, class, return에 대한 FiniteAutomata 객체도 만들어 준다.

ex) Integer

```
145  # Finite Automata Definitions with transition table
146  Integer = FiniteAutomata(
147      "INTEGER",  # matched token name
148      ["T0", "T1", "T2", "T3", "T4"],  # state
149      ["-", "0", str(POS_DIGIT), str(DIGIT)],  # input stream
150      ["T2", "T3", "T4"],  # accepted state
151      {  # nfa to dfa transition table
152          "T0": ["T1", "T2", "T3", ""],
153          "T1": ["", "", "T3", ""],
154          "T2": ["", "", "", ""],
155          "T3": ["", "", "", "T4"],
156          "T4": ["", "", "", "T4"]
157      }
158  )
```

lexical_analyzer.py

 # MERGED 생성 => DFA.py에 선언된 객체들로 이루어짐

불필요한 Token 체크를 줄이고자 MERGED를 총 4가지로 나눠서 생성함.

아래와 같이 경우를 나눴기 때문에, 처음 검사하는 문자가 3이라는 숫자일 때
MERGED1부터 MERGED2는 검사하지 않고 Integer, Operator 검사만 한다.

```
5  # The elements in each array are ordered according to priority
6  MERGED1 = [Literal, Character, Comma, Lbrace, Rbrace, Lparen, Rparen, Relop, Assign, Semicolon, Larray, Rarray]
7  MERGED2 = [TTrue, TFalse, Class, If, Else, While, Return, TInt, TChar, TBoolean, TString, Identifier]
8  MERGED3 = [Integer, Operator]
9  MERGED4 = [Whitespace]
```

1) MERGED1
   : 처음 검사하는 문자가 symbols일 경우
2) MERGED2
   : 처음 검사하는 문자가 DFA.py에 있는 LETTER나 '_'일 경우
3) MERGED3
   : 처음 검사하는 문자가 OPERATOR나 DFA.py에 있는 DIGIT일 경우
4) MERGED4
   : 처음 검사하는 문자가 WHITESPACE일 경우

 # Input.java 파일, Input.out 파일 I/O
조건에 따라 터미널에서 *python lexical_analyzer.py Input.java* 라고 실행한다고 가정했을 때 Input.java 파일을 읽고 Input.out 파일을 생성한다.

```
11    # file I/O
12    f = open(sys.argv[1], 'r')
13    inputline = f.readlines()
14    filename = sys.argv[1]
15
16    file_out = open(f"{filename.replace('.java', '')}.out", 'w')
17
18
19
20    def filewrite(file, string):
21        file.write(string)
```

filewrite(file, string) 함수로 file에 string에 해당하는 내용을 작성한다.

 # inputline들을 첫 줄 부터 한 줄 씩 lexical analyzing

```
23    # error report
24    error_line = 0
25    is_error = 0
26
27    for inputStr in inputline:...
363
364   f.close()
```

 # 읽고 있는 줄의 내용을 text1이라는 변수에 deque 형태로 저장

```
31        # Use deque to check and popleft() the first letter
32        text1 = deque(inputline)
33        textState = 0
```

text1[0]부터 검사해서 accept되면 popleft()시키기 위해 deque 형태로 저장

 # 플로우

```
41      # If an error occurs, not to be checked from the next input(java file) line.
42      if is_error == 1:
43          break
44      else:
45          # 1) Depending on what alphabet text1[0] is now,
46          #   perform lexical_analyzing to classify tokens and popleft() when classified as specific tokens
47          # 2) Repeat until all are pop and there are no more letters left in text1.
48          while len(text1) > 0:
49              error = []
50              # 1) When the alphabet you examine is an alphabet in the symbols array defined in line 38
51              ## Checking about MERGED1
52              if text1[textState] in symbols:
53                  for i in range(len(MERGED1)):...
118
119             # 2) When the alphabet you examine is LETTER or '_'
120             ## Checking about MERGED2
121             elif text1[textState] in LETTER or text1[textState] == "_":...
240
241             # 3) When the alphabet you examine is OPERATOR or DIGIT
242             ## Checking about MERGED3
243             elif text1[textState] in OPERATOR or text1[0] in DIGIT:...
304
305             # 4) When the alphabet you examine is WHITESPACE
306             ## Checking about MERGED4
307             elif text1[textState] in WHITESPACE:...
333
334             # 5) When the alphabet you examine is not defined alphabet
335             ## ex) &, $, \
336             else:...
356
357         # Create token table in out file only if token is well classified without error
358         if is_error == 0:
359             for i in range(len(table)):
360                 filewrite(file_out, str(table[i]))
361                 filewrite(file_out, "\n")
```

1) 에러 상태를 확인하고 에러가 발생하지 않으면 계속 진행한다.
에러가 발생하면 inputline의 다음 줄부터는 확인할 필요 없이 Input.out 파일에 에러만을 리포트하기 때문이다.

2) (text1 == 현재 확인하는 한 줄, textState == 해당 줄에서 확인하는 글자)

text1[textState] 이 무엇인 가에 따라 MERGED1, MERGED2, MERGED3, MERGED4 중 하나에 대해 어떤 토큰에 해당하는 지 검사를 하면 되기 때문에 if문으로 경우를 나누어 주었다.

이 때 마지막 else는 이번 과제에서 어떤 토큰들에도 정의되지 않은 '&', '^', '\' 와 같은 문자인 경우 에러를 리포트한다.

- 검사하는 Token의 유효한 alphabet(=MERGED[i].Sigma) 인 지 확인한다
- 유효하다면, MERGED[i].nextState(text1[textState])를 진행한다
- 유효하지 않은 alphabet이 등장할 때까지 반복하다가 유효하지 않은 alphabet이 등장하면 Accept/Reject를 체크한다
- Accept되면 Table에 Token의 이름과 value를 append한다. 이 때 value를 popleft()해 주고 textState를 다시 0으로 초기화한다.
- Reject되면 MERGED의 다음 토큰 FiniteAutoma에 대해 위 과정을 반복한다. 이 때 MERGED의 마지막 토큰에 대해 진행했다면 에러를 리포트한다.

3) 에러가 없는 경우(line 358)에만 .out 파일에 accept된 table을 작성한다.

#4 PROBLEM & SOLUTION

1) ERROR Report

input 중에는 모든 오토마타를 거쳤지만 토큰이 분류되지 않은 경우가 있다.
즉, simple JAVA language에 포함되지 않는 유효하지 않은 값이다.

> error report를 하는 부분

```
while len(text1) > 0:
```

while문을 돌면서 토큰을 확인하는데, 각각 토큰이 Accepted 되지 않은 경우 error report를 던진다.

```
elif MERGED1[i].isAccepted() == False and i == (len(MERGED1) - 1):
printStr = "ERROR: is not match type, line:" + str(error_line)
```

input이 symbols에 있지만 MERGED 1에서 accept이 되지 않을 때, 예를 들어 “!”, ‘av’ 같은 경우가 있다. 문자열과 문자의 형태는 accept이 되지 않을 경우 match 되지 않기 때문에 위와 같은 에러 메시지를 출력하게 하였다.

```
elif MERGED2[i].isAccepted() == False and i == (len(MERGED2) - 1):
elif MERGED2[i].isAccepted() == False and i == (len(MERGED2) - 1):
```

input이 letter나 _로 시작할 때, accept이 되지 않은 경우이다. accept이 되는 경우는 `TTrue, TFalse, Class, If, Else, While, Return, TInt, TChar, TBoolean, TString, Identifier` 이다.

```
elif MERGED3[i].isAccepted() == False and i == (len(MERGED3) - 1):
```

가 숫자거나 -, *, /, + 연산자로 시작할 때 accept이 되지 않을 때 나타난다.

```
else:
```

유효하지 않은 입력(예: %, ^, &)일 때 위 MERGED2와 MERGED3일 때 발생하는 에러와 같다.

> error report 과정

1) 먼저 오류 메시지를 담기 위한 subStr을 만들어준다. 그리고 input에서 값을 pop 시킨다. 한 문장 안에서 오류가 난 부분은 오류 메시지에서 표시해주기 위해 ^ 로 표현하는데, 이를 위해서 한 문장안에 몇 번째 글자에서 오류가 났는지 error_num으로 센다. error_num은 word_count의 역할을 하기 때문에 오류가 나지 않은 부분이라도 토큰이 pop 될 때마다 글자를 세준다.

```
subStr = ""
subStr += text1.popleft()
error_num = error_num + 1
table.append(["Not Match", subStr])  # 없는 글자 처리해줌.
```

2) 기존에 열었던 아웃풋 파일은 닫고 오류 메시지를 출력할 파일을 연다. 오류의 종류에 따라 메시지를 입력한다. 메시지 종류는 2가지이다. 유효하지 않은 글자이거나, 유효한 글자이지만 매칭되는 토큰이 없는 경우이다. 그리고 오류 메시지를 파일에 적는다. 오류가 난 line number도 출력된다.

```
file_out.close()
file_out = open(f"{filename.replace('.java', '')}.out", 'w')
printStr = "ERROR: There is a letter " + "'" + subStr + "'" + " is not acceptable. line:" + str(error_line)

filewrite(file_out, printStr)
filewrite(file_out, '\n')
is_error = 1
```

3) 오류가 난 곳의 문장과 ^ 표시를 적는 부분이다.

```
for index in range(error_num):
    error.append(" ")
error[error_num - 1] = "^"
filewrite(file_out, inputline.rstrip('\n')+'\n')
filewrite(file_out,  ''.join(error) + '\n')
textState = 0
```

오류 report를 하고 나면 is_error 변수에 1이 저장되어 오류임을 밝힌다. 따라서 is_errir이 1이면 input 값을 모두 읽은 후 token table을 출력하지 않고 오류 리포트를 출력한다.

> INPUT TEST with ERROR CASE

없는 글자면 토큰 테이블을 만들지 않고 error 메시지를 output 파일에 넣는다.

파일: [error1 ~ error7.java / error1 ~ error1.out]

`'av'`
`int char`
	`ERROR: is not match type, line:1`
	`'av'`
	`^`
	오류: 문자가 아닌데 ‘를 사용했다

`"Compiler Term project D-1"`
`"writer. Bae & Jwa"`
	`ERROR: is not match type, line:1`
	`"Compiler Term project D-1"`
	`^`
	오류: 따옴표 안에 - 가 들어 있어 문자열이 아니다

`"new line`
`error message"`
	`ERROR: is not match type, line:1`
	`"new line`
	`^`
	오류: 공백이 아닌 개행 문자가 들어가서 오류가 생겼다

`a != b`
`b =! c`
	`ERROR: is not match type, line:2`
	`b =! c`
	`^`
	오류: 존재하지 않는 연산자이다.

`float 5.3`	`ERROR: There is a letter "." is not acceptable. line:1`
	`float 5.3`
	`^`
	오류: ‘.’는 유효하지 않은 글자인데다가, simple JAVA는 실수형 정수가 존재하지 않는다.

`\\\n`	`ERROR: There is a letter "\" is not acceptable. line:1`
	`\\\n`
	`  ^`
	오류:유효하지 않은 글자 \ 가 있다.

`"'interpol'"`	`ERROR: is not match type, line:1`
	`"'interpol'"`
	`^`
	오류: 문자열 안에는 따옴표가 올 수 없다.

2) MINUS problem

마이너스가 들어갈 수 있는 토큰은 음의 정수와 연산자 빼기이다. 따라서 마이너스 프로블럼은 input이 MERGED3에 있는 오토마타를 검사할 때 발생할 수 있다.

```
MERGED3 = [Integer, Operator]
```

처음에 설정한 토큰 우선순위에 따르면 기본적으로 MERGED3에서 토큰의 우선순위는 -가 들어올 때 정수로 판단한다. 따라서 input이 음의 정수와 같은 형태 일 때 (예: -1) 무조건 음의 정수라고 판단하는 문제점이 있었다. 따라서 a-1 이 인풋일 경우 <IDENTIFIER>, <INTEGER>로 판별한다.

하지만 syntax 관점에서 보면 a-1 일 때의 - 는 빼기 연산자를 뜻한다. 따라서 특정 경우에 대해서 -가 빼기 연산자인지 판단하는 오토마타를 정수를 판단하는 오토마타보다 먼저 검사해야한다. 마이너스 입력 앞에 있는 토큰의 종류에 따라 크게 3가지일 때 -는 빼기 연산자로 구분해야한다.

코드를 보면 이 3가지 경우를 만족하는지 아닌지 if 문을 통해 검사한다.

```
elif text1[textState] in OPERATOR or text1[0] in DIGIT:
    idx = 0
    if table and (table[-1][0] == "IDENTIFIER" or table[-1][0] == "INTEGER" or table[-1][0] == "CHARACTER") and text1[textState] == "-":
        idx = 1
```

`table[-1][0]` 는 앞에 있는 lexeme을 가리킨다. token table의 마지막에 들어있는 토큰이다.

1) 앞에 있는 인풋이 IDENTIFIER 토큰일 때
   ex) a-1
   변수는 값을 저장하고 있기 때문에 integer가 들어있다면 다른 정수와 함께 빼기 연산의 피연산자가 된다. 따라서 변수 타입 뒤에 있는 마이너스는 연산자가 된다.

2) 앞에 있는 인풋이 INTEGER 토큰일 때
   ex) 1-1
   두개의 정수가 나열되는 경우는 1, -1 처럼 콤마로 구분하기 때문에 두 개의 정수 사이의 마이너스는 빼기 연산자를 의미한다.

3) 앞에 있는 인풋이 CHARACTER 토큰 일 때
   ex) ‘a’-1
   JAVA 언어에서는 ‘a’와 같은 단일 문자는 ASCII 코드 값으로 숫자가 되어 정수와 연산할 수 있다. 만약에 단일 문자 뒤에 정수가 오는 경우 syntax까지 고려했을 때

유효한 경우는 ‘a’, -1 처럼 콤마로 구분되어야한다. 따라서 단일 문자 뒤 마이너스는 빼기 연산자가 된다.

> 해결한 결과는 다음과 같은 출력으로 확인할 수 있었다.

<table>
<tr><td colspan="2">앞에 있는 인풋이 IDENTIFIER 토큰일 때</td></tr>
<tr><td>

```
a = 1
a - 11
```

</td><td>

```
['IDENTIFIER', 'a']
['ASSIGN', '=']
['INTEGER', '1']
['IDENTIFIER', 'a']
['OPERATOR', '-']
['INTEGER', '11']
```

</td></tr>
<tr><td colspan="2">앞에 있는 인풋이 INTEGER 토큰일 때</td></tr>
<tr><td>

```
1234556-1
```

</td><td>

```
['INTEGER', '1234556']
['OPERATOR', '-']
['INTEGER', '1']
```

</td></tr>
<tr><td colspan="2">앞에 있는 인풋이 CHARACTER 토큰 일 때</td></tr>
<tr><td>

```
'a'-1
```

</td><td>

```
['CHARACTER', "'a'"]
['OPERATOR', '-']
['INTEGER', '1']
```

</td></tr>
</table>

#5 TEST INPUT & OUTPUT

파일 제목: input 파일과 output 파일은 이름이 같고 각각 확장자가 .java와 .out이다.

> INPUTS IN ANNOUNCED in E-CLASS

파일: [input1 ~ input12.java / output1 ~ output12.out]

001	['INTEGER', '0'] ['INTEGER', '0'] ['INTEGER', '1']
0010	['INTEGER', '0'] ['INTEGER', '0'] ['INTEGER', '10']
0010a0010	['INTEGER', '0'] ['INTEGER', '0'] ['INTEGER', '10'] ['IDENTIFIER', 'a0010']
0010-10	['INTEGER', '0'] ['INTEGER', '0'] ['INTEGER', '10'] ['OPERATOR', '-'] ['INTEGER', '10']
0010--10	['INTEGER', '0'] ['INTEGER', '0'] ['INTEGER', '10'] ['OPERATOR', '-'] ['INTEGER', '-10']
-0	['OPERATOR', '-'] ['INTEGER', '0']
0abc0	['INTEGER', '0'] ['IDENTIFIER', 'abc0']
123if	['INTEGER', '123'] ['KEYWORD', 'if']
123if0	['INTEGER', '123'] ['IDENTIFIER', 'if0']

' '	['CHARACTER', "' '"]
a-1	['IDENTIFIER', 'a'] ['OPERATOR', '-'] ['INTEGER', '1']
int main(){char if123='1';int 0a=a+-1;return -0;}	['TYPE', 'int'] ['IDENTIFIER', 'main'] ['LPAREN', '('] ['RPAREN', ')'] ['LBRACE', '{'] ['TYPE', 'char'] ['IDENTIFIER', 'if123'] ['ASSIGN', '='] ['CHARACTER', "'1'"] ['SEMICOLON', ';'] ['TYPE', 'int'] ['INTEGER', '0'] ['IDENTIFIER', 'a'] ['ASSIGN', '='] ['IDENTIFIER', 'a'] ['OPERATOR', '+'] ['INTEGER', '-1'] ['SEMICOLON', ';'] ['KEYWORD', 'return'] ['OPERATOR', '-'] ['INTEGER', '0'] ['SEMICOLON', ';'] ['RBRACE', '}']

> SPECIAL MINUS INPUTS

파일: [minus1 ~ minus8.java / minus1 ~ minus8.out]

`------1`	`['OPERATOR', '-']`
	`['OPERATOR', '-']`
	`['OPERATOR', '-']`
	`['OPERATOR', '-']`
	`['OPERATOR', '-']`
	`['INTEGER', '-1']`

`'k' - 1`	`['CHARACTER', "'k'"]`
	`['OPERATOR', '-']`
	`['INTEGER', '1']`

`a = 167`
`1 - a`
	`['IDENTIFIER', 'a']`
	`['ASSIGN', '=']`
	`['INTEGER', '167']`
	`['INTEGER', '1']`
	`['OPERATOR', '-']`
	`['IDENTIFIER', 'a']`

`a / --167`
`a - 1`
	`['IDENTIFIER', 'a']`
	`['OPERATOR', '/']`
	`['OPERATOR', '-']`
	`['INTEGER', '-167']`
	`['IDENTIFIER', 'a']`
	`['OPERATOR', '-']`
	`['INTEGER', '1']`

`number1 - number2 = number3`	`['IDENTIFIER', 'number1']`
	`['OPERATOR', '-']`
	`['IDENTIFIER', 'number2']`
	`['ASSIGN', '=']`
	`['IDENTIFIER', 'number3']`

`number -- float`	`['IDENTIFIER', 'number']`
	`['OPERATOR', '-']`
	`['OPERATOR', '-']`
	`['IDENTIFIER', 'float']`

123 -+-/------a	['INTEGER', '123'] ['OPERATOR', '-'] ['OPERATOR', '+'] ['OPERATOR', '-'] ['OPERATOR', '/'] ['OPERATOR', '-'] ['OPERATOR', '-'] ['OPERATOR', '-'] ['OPERATOR', '-'] ['OPERATOR', '-'] ['OPERATOR', '-'] ['IDENTIFIER', 'a']
123--------123	['INTEGER', '123'] ['OPERATOR', '-'] ['OPERATOR', '-'] ['OPERATOR', '-'] ['OPERATOR', '-'] ['OPERATOR', '-'] ['OPERATOR', '-'] ['OPERATOR', '-'] ['INTEGER', '-123']

> VARIOUS LEXICAL SPECIFICATION CASES

파일: [test1 ~ test9.java / test1.out~test9.out]

Variable type & IDENTIFIER & Terminating Symbol
 	
`int _a_;`
`char min2;`
`boolean _true;`
`String str__str;`
	`['TYPE', 'int']`
	`['IDENTIFIER', '_a_']`
	`['SEMICOLON', ';']`
	`['TYPE', 'char']`
	`['IDENTIFIER', 'min2']`
	`['SEMICOLON', ';']`
	`['TYPE', 'boolean']`
	`['IDENTIFIER', '_true']`
	`['SEMICOLON', ';']`
	`['TYPE', 'String']`
	`['IDENTIFIER', 'str__str']`
	`['SEMICOLON', ';']`

Signed Integer
 	
`0`
`123`
`-1234`
	`['INTEGER', '0']`
	`['INTEGER', '123']`
	`['INTEGER', '-1234']`

Single Character
 	
`'a'`
`'1'`
`' '`
	`['CHARACTER', "'a'"]`
	`['CHARACTER', "'1'"]`
	`['CHARACTER', "' '"]`

Boolean String
 	
`true`
`false`
	`['BOOLEAN', 'true']`
	`['BOOLEAN', 'false']`

Literal `"Literal String"`	`['LITERAL', '"Literal String"']`

Keywords for special statements & Comparison operators

```
if a <<> b
else a == b
while a != c
class <= cc
return a >= b
```

```
['KEYWORD', 'if']
['IDENTIFIER', 'a']
['RELOP', '<']
['RELOP', '<']
['RELOP', '>']
['IDENTIFIER', 'b']
['KEYWORD', 'else']
['IDENTIFIER', 'a']
['RELOP', '==']
['IDENTIFIER', 'b']
['KEYWORD', 'while']
['IDENTIFIER', 'a']
['RELOP', '!=']
['IDENTIFIER', 'c']
['KEYWORD', 'class']
['RELOP', '<=']
['IDENTIFIER', 'cc']
['KEYWORD', 'return']
['IDENTIFIER', 'a']
['RELOP', '>=']
['IDENTIFIER', 'b']
```

Arithmetic Operators

```
a * / + - / b
```

```
['IDENTIFIER', 'a']
['OPERATOR', '*']
['OPERATOR', '/']
['OPERATOR', '+']
['OPERATOR', '-']
['OPERATOR', '/']
['IDENTIFIER', 'b']
```

Assign & Braces & Parens & Arrays

```
int  main {
    [a] = 1
    funv(a[1]);
}
```

```
['TYPE', 'int']
['IDENTIFIER', 'main']
['LBRACE', '{']
['LARRAY', '[']
['IDENTIFIER', 'a']
['RARRAY', ']']
['ASSIGN', '=']
['INTEGER', '1']
['IDENTIFIER', 'funv']
['LPAREN', '(']
['IDENTIFIER', 'a']
['LARRAY', '[']
['INTEGER', '1']
['RARRAY', ']']
['RPAREN', ')']
['SEMICOLON', ';']
['RBRACE', '}']
```

Whitespaces

```
'\n'      "\t"
```

```
ERROR: is not match type, line:1
'\n'      "\t"
^
```

오류: 문자열에는 blank 공백 한 글자는 가능하지만 whitespace는 되지 않는다.

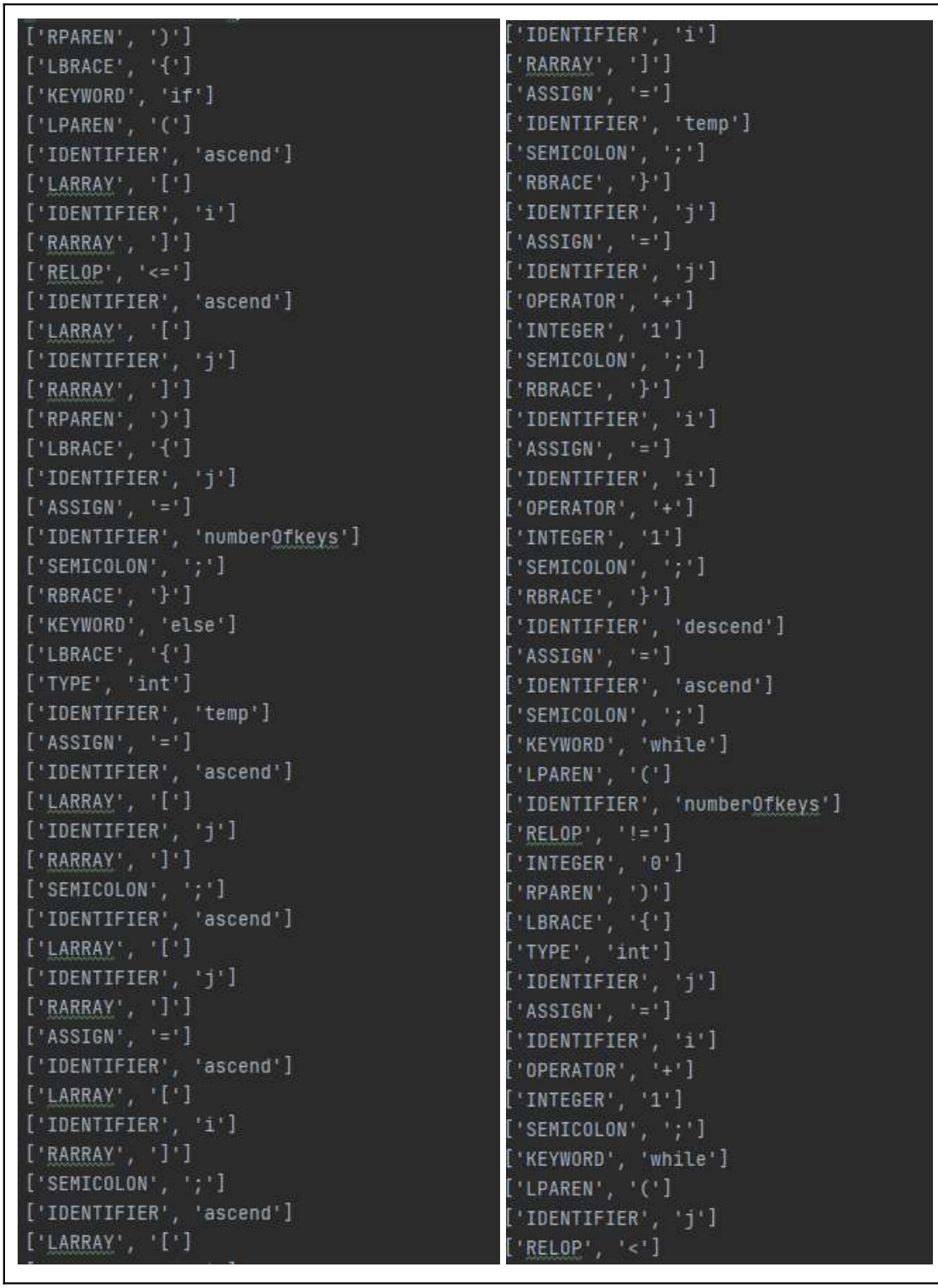
> LONG SOURCE CODE

파일: [longSource1.java / longSource1.out]

```
class Simplesort {
 int[] sort(int numOfelements) {
   String subject = "simple sort";
   int numberOfkeys = 3;
   if (numOfelements == numberOfkeys){
     subject = "sort start";
   }
   int[] key={2,1,3};
   int[] ascend;
   int[] descend;
   int i=0;

   ascend = key;
   while(numberOfkeys != 0){
     int j = i + 1;
     while(numberOfkeys > j){
       if(ascend[i]<=ascend[j]){
         j = numberOfkeys;
       }else{
         int temp = ascend[j];
         ascend[j]=ascend[i];
         ascend[i]=temp;
       }
       j = j + 1;
     }
     i = i + 1;
   }

   descend = ascend;
   while(numberOfkeys != 0){
     int j = i + 1;
     while(j < numberOfkeys ){
       if(descend[i] >= descend[j]){
         j = numberOfkeys;
       }else{
         int temp = descend[j];
         descend[j]=descend[i];
         descend[i]=temp;
       }
       j = j + 1;
```

```
      }
      i = i + 1;
    }

    return key;
  }
}
```

```
['KEYWORD', 'class']
['IDENTIFIER', 'Simplesort']
['LBRACE', '{']
['TYPE', 'int']
['LARRAY', '[']
['RARRAY', ']']
['IDENTIFIER', 'sort']
['LPAREN', '(']
['TYPE', 'int']
['IDENTIFIER', 'numOfelements']
['RPAREN', ')']
['LBRACE', '{']
['TYPE', 'String']
['IDENTIFIER', 'subject']
['ASSIGN', '=']
['LITERAL', '"simple sort"']
['SEMICOLON', ';']
['TYPE', 'int']
['IDENTIFIER', 'numberOfkeys']
['ASSIGN', '=']
['INTEGER', '3']
['SEMICOLON', ';']
['KEYWORD', 'if']
['LPAREN', '(']
['IDENTIFIER', 'numOfelements']
['RELOP', '==']
['IDENTIFIER', 'numberOfkeys']
['RPAREN', ')']
['LBRACE', '{']
['IDENTIFIER', 'subject']
['ASSIGN', '=']
['LITERAL', '"sort start"']
['SEMICOLON', ';']
['RBRACE', '}']
['TYPE', 'int']
['LARRAY', '[']
['RARRAY', ']']
['IDENTIFIER', 'key']
['ASSIGN', '=']
['LBRACE', '{']
['INTEGER', '2']
['COMMA', ',']
['INTEGER', '1']
['COMMA', ',']
['INTEGER', '3']
['RBRACE', '}']
['SEMICOLON', ';']
['TYPE', 'int']
['LARRAY', '[']
['RARRAY', ']']
['IDENTIFIER', 'ascend']
['SEMICOLON', ';']
['TYPE', 'int']
['LARRAY', '[']
['RARRAY', ']']
['IDENTIFIER', 'descend']
['SEMICOLON', ';']
['TYPE', 'int']
['IDENTIFIER', 'i']
['ASSIGN', '=']
['INTEGER', '0']
['SEMICOLON', ';']
['IDENTIFIER', 'ascend']
['ASSIGN', '=']
['IDENTIFIER', 'key']
['SEMICOLON', ';']
['KEYWORD', 'while']
['LPAREN', '(']
['IDENTIFIER', 'numberOfkeys']
['RELOP', '!=']
['INTEGER', '0']
['RPAREN', ')']
['LBRACE', '{']
['TYPE', 'int']
['IDENTIFIER', 'j']
['ASSIGN', '=']
['IDENTIFIER', 'i']
['OPERATOR', '+']
['INTEGER', '1']
['SEMICOLON', ';']
['KEYWORD', 'while']
['LPAREN', '(']
['IDENTIFIER', 'numberOfkeys']
['RELOP', '>']
['IDENTIFIER', 'j']
```



```
['RPAREN', ')']
['LBRACE', '{']
['KEYWORD', 'if']
['LPAREN', '(']
['IDENTIFIER', 'ascend']
['LARRAY', '[']
['IDENTIFIER', 'i']
['RARRAY', ']']
['RELOP', '<=']
['IDENTIFIER', 'ascend']
['LARRAY', '[']
['IDENTIFIER', 'j']
['RARRAY', ']']
['RPAREN', ')']
['LBRACE', '{']
['IDENTIFIER', 'j']
['ASSIGN', '=']
['IDENTIFIER', 'numberOfkeys']
['SEMICOLON', ';']
['RBRACE', '}']
['KEYWORD', 'else']
['LBRACE', '{']
['TYPE', 'int']
['IDENTIFIER', 'temp']
['ASSIGN', '=']
['IDENTIFIER', 'ascend']
['LARRAY', '[']
['IDENTIFIER', 'j']
['RARRAY', ']']
['SEMICOLON', ';']
['IDENTIFIER', 'ascend']
['LARRAY', '[']
['IDENTIFIER', 'j']
['RARRAY', ']']
['ASSIGN', '=']
['IDENTIFIER', 'ascend']
['LARRAY', '[']
['IDENTIFIER', 'i']
['RARRAY', ']']
['SEMICOLON', ';']
['IDENTIFIER', 'ascend']
['LARRAY', '[']
['IDENTIFIER', 'i']
['RARRAY', ']']
['ASSIGN', '=']
['IDENTIFIER', 'temp']
['SEMICOLON', ';']
['RBRACE', '}']
['IDENTIFIER', 'j']
['ASSIGN', '=']
['IDENTIFIER', 'j']
['OPERATOR', '+']
['INTEGER', '1']
['SEMICOLON', ';']
['RBRACE', '}']
['IDENTIFIER', 'i']
['ASSIGN', '=']
['IDENTIFIER', 'i']
['OPERATOR', '+']
['INTEGER', '1']
['SEMICOLON', ';']
['RBRACE', '}']
['IDENTIFIER', 'descend']
['ASSIGN', '=']
['IDENTIFIER', 'ascend']
['SEMICOLON', ';']
['KEYWORD', 'while']
['LPAREN', '(']
['IDENTIFIER', 'numberOfkeys']
['RELOP', '!=']
['INTEGER', '0']
['RPAREN', ')']
['LBRACE', '{']
['TYPE', 'int']
['IDENTIFIER', 'j']
['ASSIGN', '=']
['IDENTIFIER', 'i']
['OPERATOR', '+']
['INTEGER', '1']
['SEMICOLON', ';']
['KEYWORD', 'while']
['LPAREN', '(']
['IDENTIFIER', 'j']
['RELOP', '<']
```

```
['IDENTIFIER', 'numberOfkeys']
['RPAREN', ')']
['LBRACE', '{']
['KEYWORD', 'if']
['LPAREN', '(']
['IDENTIFIER', 'descend']
['LARRAY', '[']
['IDENTIFIER', 'i']
['RARRAY', ']']
['RELOP', '>=']
['IDENTIFIER', 'descend']
['LARRAY', '[']
['IDENTIFIER', 'j']
['RARRAY', ']']
['RPAREN', ')']
['LBRACE', '{']
['IDENTIFIER', 'j']
['ASSIGN', '=']
['IDENTIFIER', 'numberOfkeys']
['SEMICOLON', ';']
['RBRACE', '}']
['KEYWORD', 'else']
['LBRACE', '{']
['TYPE', 'int']
['IDENTIFIER', 'temp']
['ASSIGN', '=']
['IDENTIFIER', 'descend']
['LARRAY', '[']
['IDENTIFIER', 'j']
['RARRAY', ']']
['SEMICOLON', ';']
['IDENTIFIER', 'descend']
['LARRAY', '[']
['IDENTIFIER', 'j']
['RARRAY', ']']
['ASSIGN', '=']
['IDENTIFIER', 'descend']
['LARRAY', '[']
['IDENTIFIER', 'i']
['RARRAY', ']']
['SEMICOLON', ';']
['IDENTIFIER', 'descend']
```

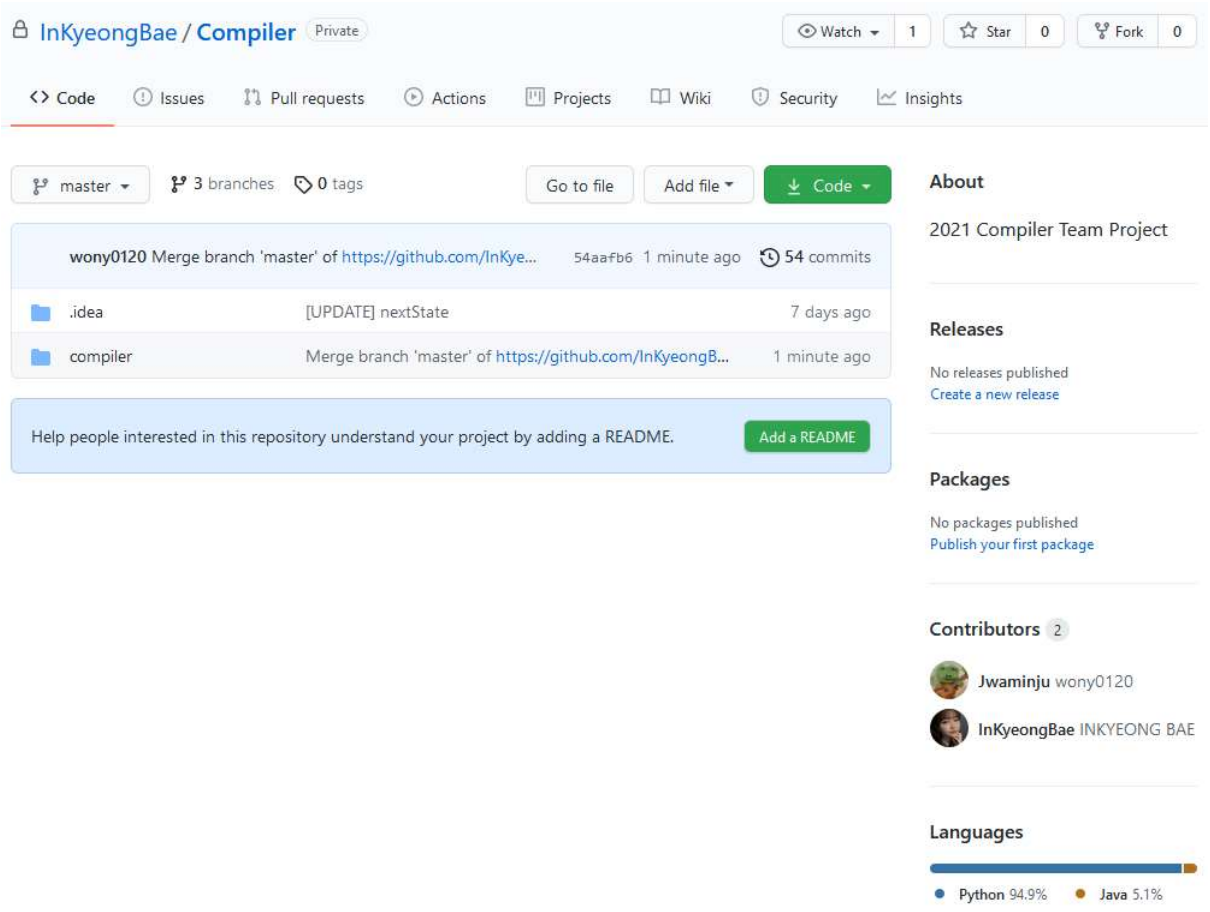
```
['LARRAY', '[']
['IDENTIFIER', 'i']
['RARRAY', ']']
['ASSIGN', '=']
['IDENTIFIER', 'temp']
['SEMICOLON', ';']
['RBRACE', '}']
['IDENTIFIER', 'j']
['ASSIGN', '=']
['IDENTIFIER', 'j']
['OPERATOR', '+']
['INTEGER', '1']
['SEMICOLON', ';']
['RBRACE', '}']
['IDENTIFIER', 'i']
['ASSIGN', '=']
['IDENTIFIER', 'i']
['OPERATOR', '+']
['INTEGER', '1']
['SEMICOLON', ';']
['RBRACE', '}']
['KEYWORD', 'return']
['IDENTIFIER', 'key']
['SEMICOLON', ';']
['RBRACE', '}']
['RBRACE', '}']
```

#6 GITHUB & Google Docs & KakaoTalk & Zoom

팀플에 있어서 협업 방식으로 GITHUB와 Google Docs, KakaoTalk, Zoom 등을 이용하였다.

20210331 : Google Docs 이용하여 Token Name 정의
~ 20210407 : Regular expression, NFA, Transition Table, DFA 손으로 그리고 확인
~ 20210410 : 코드 전체 flow 정리 및 Github repository 생성
~ 20210413 : Token별 DFA 및 Transition Table 구현 코드 완성
~ 20210415 : lexical_analyzer.py 구현 및 error report, MINUS problem 등 해결
~ 20210416 : 추가 에러 수정 및 보고서 작성
~ 20210417 : 보고서 작성 및 코드 주석 추가

> 깃허브 링크 : https://github.com/InKyeongBae/Compiler



#7 LINUX

VMware에서 우분투 리눅스 환경에서 실행이 잘 되는 것을 볼 수 있다.

$ python3 lexical_analyzer.py input.java