Team 10

Compiler Term-Project #1

The implementation of a lexical analyzer

Date	May 9, 2020
Instructor	Hyosu Kim
Team	#10
Members	Heesang Ro (20145001) Junhyuck Woo (20145337)

INDEX

LEXICAL SPECIFICATIONS

Variable type
- int fir a signed integer
- char for a literal string
- bool for a Boolean string
- float for a floating-point umber

Signed integer
- A single zero digit
- A non-empty sequence of digits, starting from a non-zero digit
- A non-empty sequence of digits, starting from a minus sign symbol and a non-zero digit

Literal string
- Any combination of digits, English letters, and blanks, starting from and terminating with a symbol “

Boolean string: true and false

Floating-point number
- A sequence that meets the following conditions:
  1) It starts with or without a negative sign symbol
  2) . (a decimal point) appears only once
  3) Scientific/exponential symbols like E are not allowed
  4) Both left and right side of the decimal point must not be empty sequence
  5) The left side of a decimal point must be a single digit 0 or a non-empty sequence starting from a non-zero digit
  6) The right side of a decimal point must be a single digit 0 or a non-empty sequence terminating with a non-zero digit

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

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

Arithmetic operators: +, -, *, and /

Bitwise operator: <<, >>, &, 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 symbol for separating input arguments in functions: ,

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

TOKENS

Token	Lexeme
VARIABLE	int, float, char, bool
KEYWORD	if, else, while, for, return
LOGIC	true, false
ID	i, j, k, ab_123, func1, func_, __func_bar__, ……
INTEGER	0, 1, 22, 123, 56, -1, -22, -123, -56, ……
FLOAT	0.5, 0.0, -10.0, 100.00001, ,,,,,
LITERAL	“Hello world”, “My student id is 12345678”, ……
OPERATOR	+, -, *, /, <<, >>, &, |
COMPARISON	<, >, ==, !=, <=, >=
WHITESPACE	, \t, \n
BRACE	{, }
PAREN	(,)
ASSIGN	=
TERM	;
COMMA	,

REGULAR EXPRESSION

Alphabet (Σ)

- 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, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, -, *, /, <, &, |, =, ;, , {, }, (,), ., ,

Symbol

- zero = 0
- non-zero = 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
- 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

Token	Regular Expression
ID	$(letter \mid \_) (letter \mid non\_zero \mid zero \mid \_)^*$
INTEGER	$zero \mid ((\varepsilon \mid -) non\_zero (zero \mid non\_zero)^*)$
FLOAT	$\big((\varepsilon \mid -)(zero \mid (non\_zero(zero \mid non\_zero)^*)) . (zero \mid (zero \mid non\_zero)^* non\_zero)$
LITERAL	" $(letter \mid non\_zero \mid zero \mid)^*$ "
VARIABLE	$int \mid float \mid char \mid bool$
KEYWORD	$if \mid else \mid while \mid for \mid return$
LOGIC	$true \mid false$
OPERATOR	$+ \mid - \mid * \mid / \mid << \mid >> \mid \& \mid \mid$
COMPARISON	$< \mid > \mid == \mid != \mid <= \mid >=$
WHITESPACE	$(\mid \backslash t \mid \backslash n)^+$
BRACE	{ | }
PAREN	(|)
ASSIGN	=
TERM	;
COMMA	,

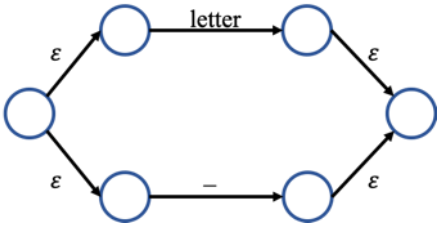
NFA (Non-deterministic Finite Automata)

Our team implemented the NAF (Non-deterministic Finite Automata) with the McNaughton-Yamada-Thompson algorithm. We only handle 4 kinds of tokens, and the other tokens will be implemented as code without a transition table.

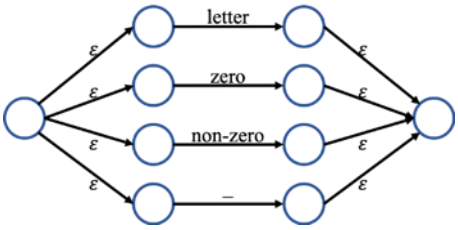
1. ID

$$(letter \mid \_) (letter \mid zero \mid non\_zero \mid \_)^*$$

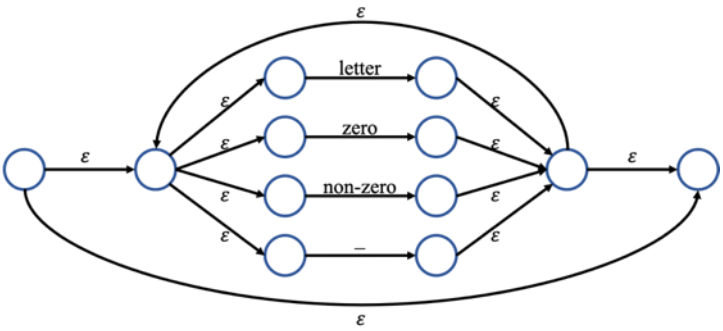
- $(letter \mid \_)$



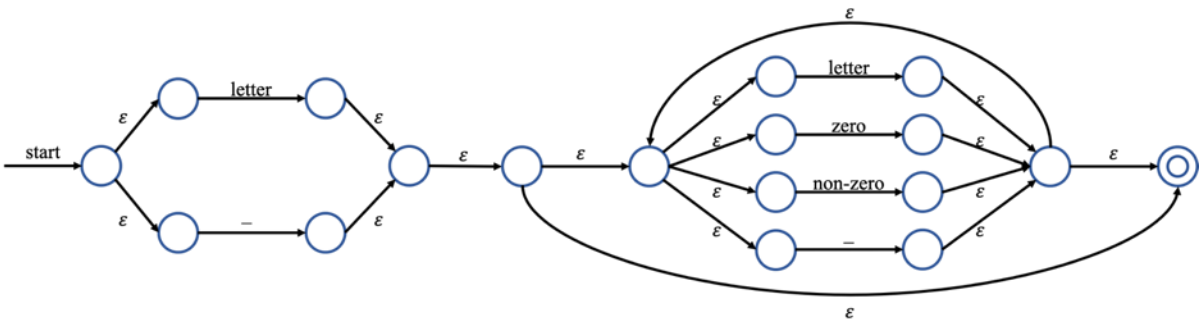
- $(letter \mid zero \mid non\_zero \mid \_)$



- $(letter \mid zero \mid non\_zero \mid \_)^*$



- $(letter \mid \_) (letter \mid zero \mid non\_zero \mid \_)^*$

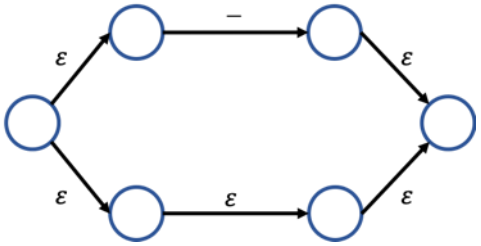


2. INTEGER

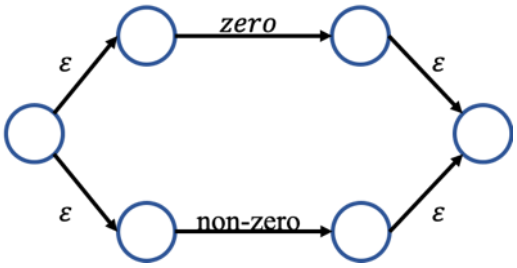
$$zero \mid ((- \mid \varepsilon)\ non\_zero\ (zero \mid non\_zero)^*)$$

- $zero$, non_zero

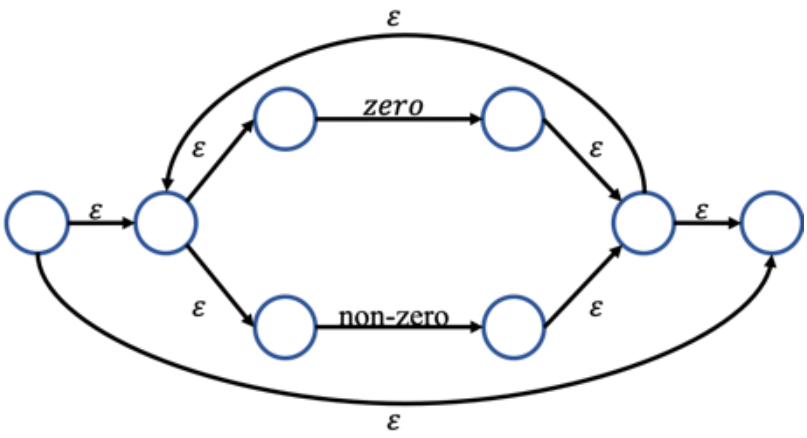
- $(- \mid \varepsilon)$



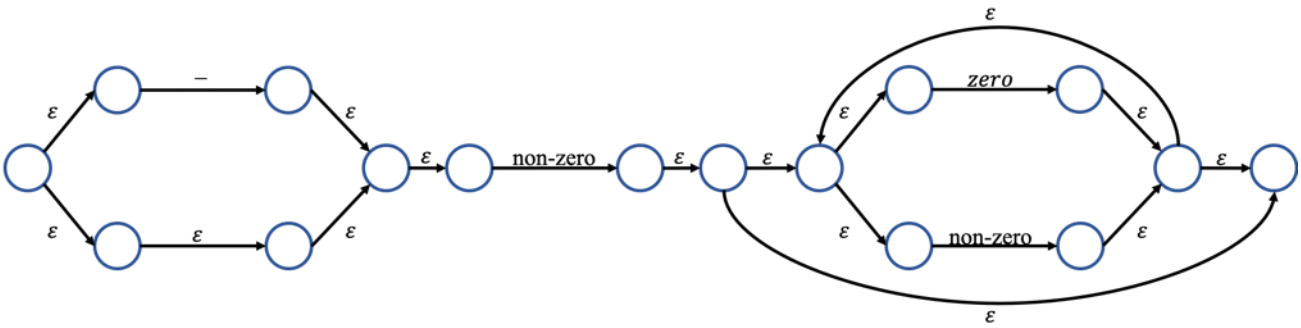
- $(zero \mid non\_zero)$



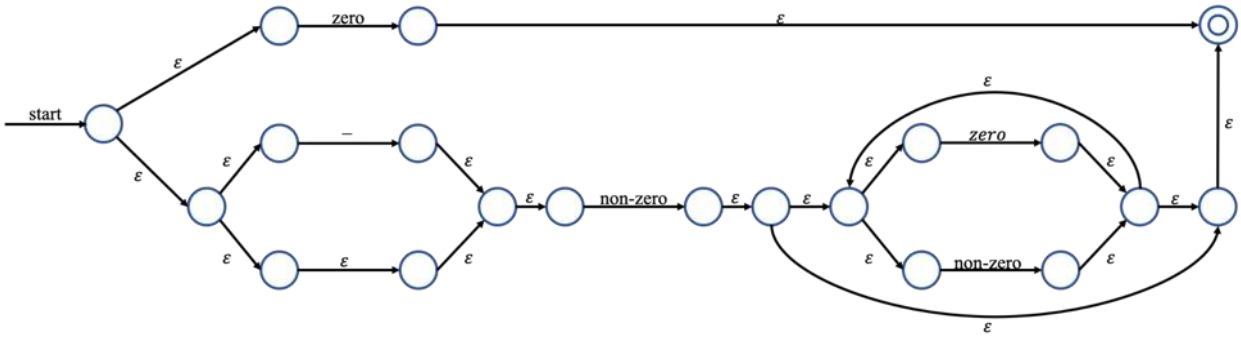
- $(zero \mid non\_zero)^*$



□ $((- \mid \varepsilon)\ non\_zero\ (zero \mid non\_zero)^*)$



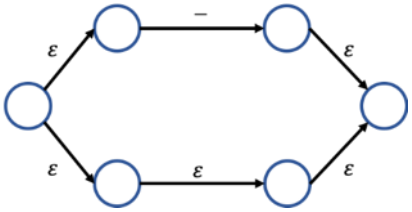
□ $zero \mid ((- \mid \varepsilon)\ non\_zero\ (zero \mid non\_zero)^*)$



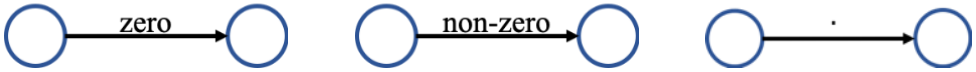
3. FLOAT

$$(-|\ \varepsilon)(zero|non\_zero(zero\ |\ non\_zero)^*)\ .\ (\ zero|(zero|non\_zero)^*\ non\_zero\ )$$

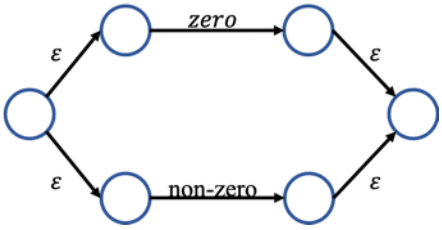
- $(-|\ \varepsilon)$



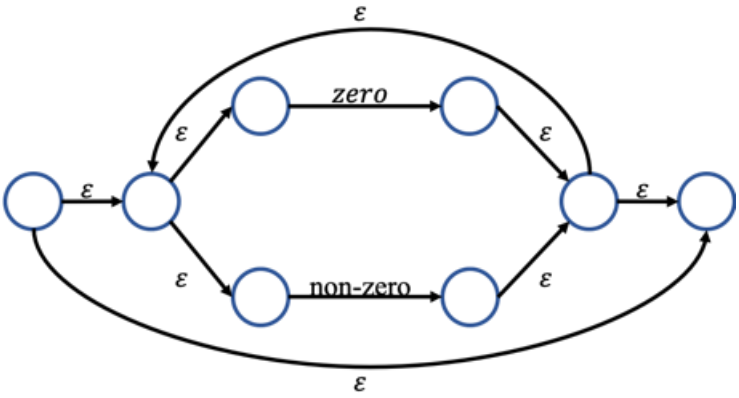
- $zero,\quad non\_zero,\quad .$



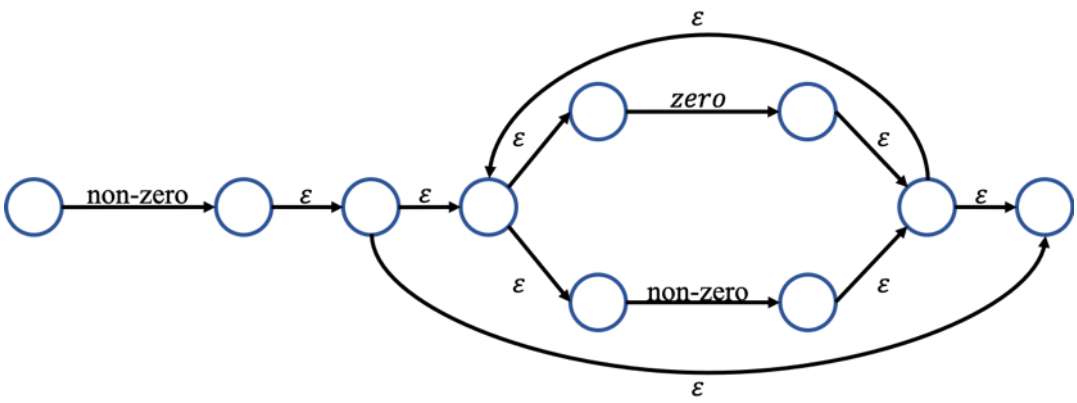
- $(zero\ |\ non\_zero)$



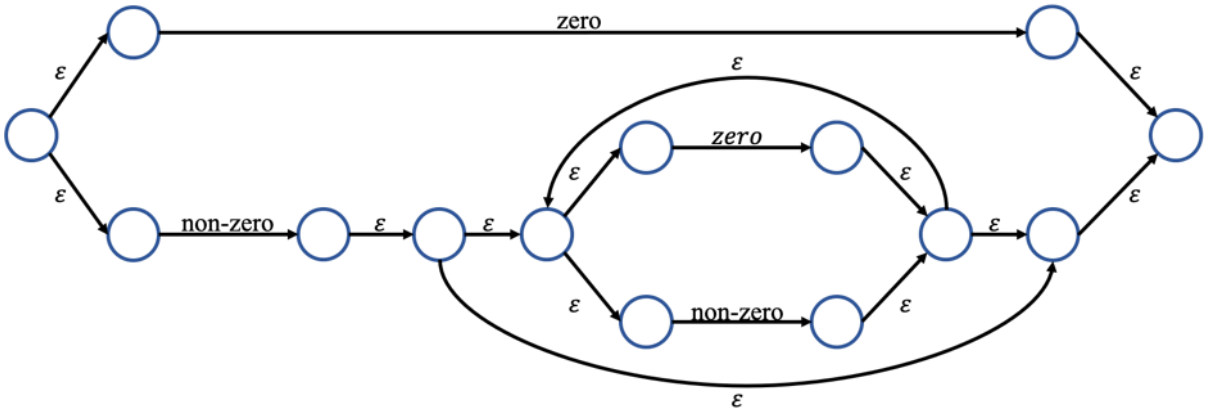
- $(zero\ |\ non\_zero)^*$



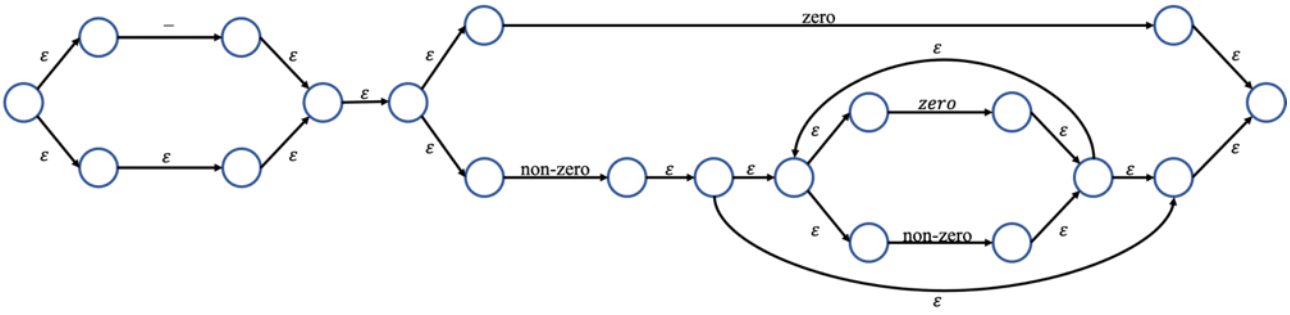
- $non\_zero(zero\ |\ non\_zero)^*$



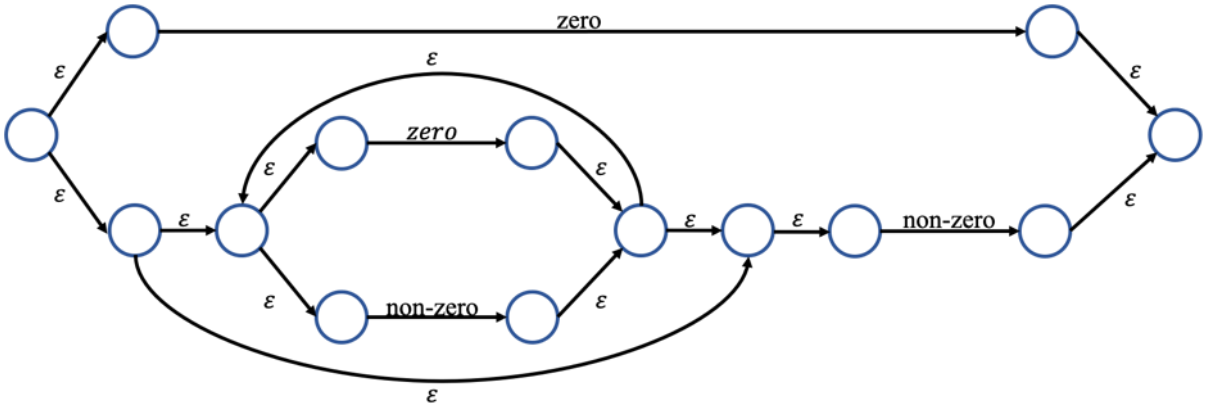
- $(zero|non\_zero(zero \mid non\_zero)^*$



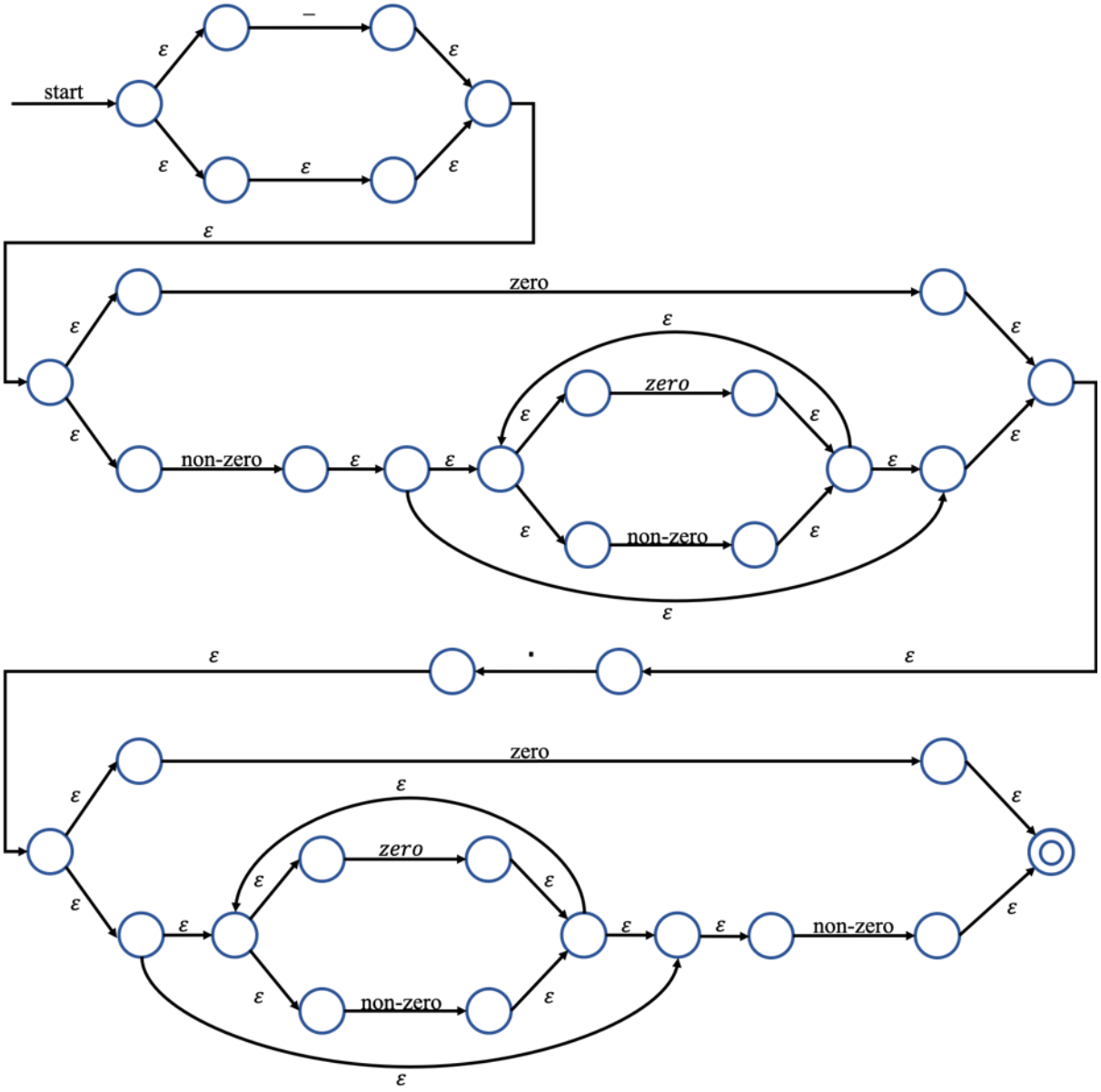
- $(- \mid \varepsilon)(zero|non\_zero(zero \mid non\_zero)^*)$



- $(\ zero|(zero|non\_zero)^*\ non\_zero)$



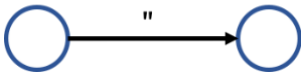
□ $(-|\varepsilon)(zero|non\_zero(zero \mid non\_zero)^*).(zero|(zero|non\_zero)^* non\_zero)$



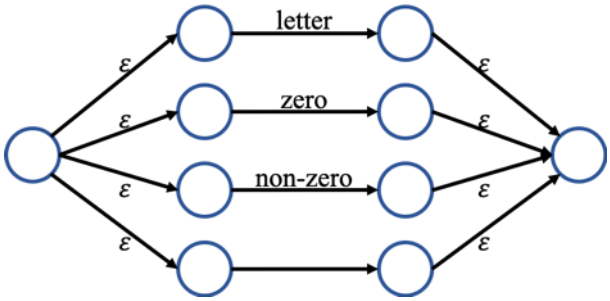
4. LITERAL

$$" \ (\ letter \mid zero \mid non - zero \mid \ )^* \ "$$

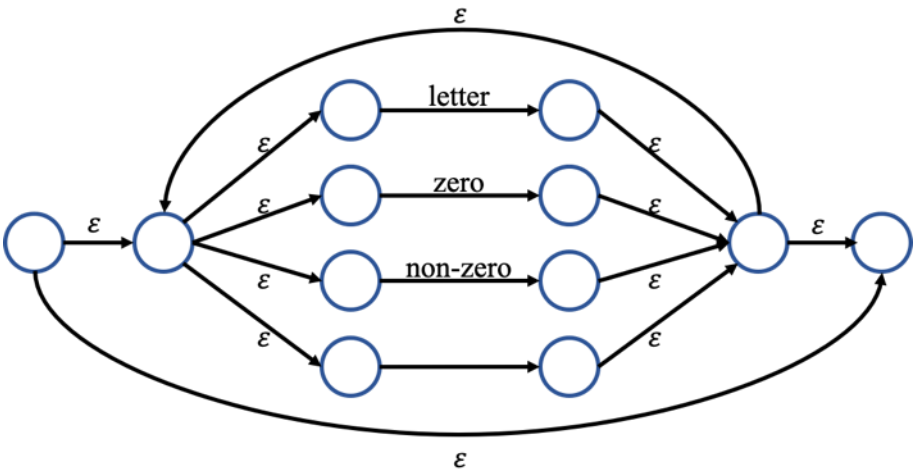
- "



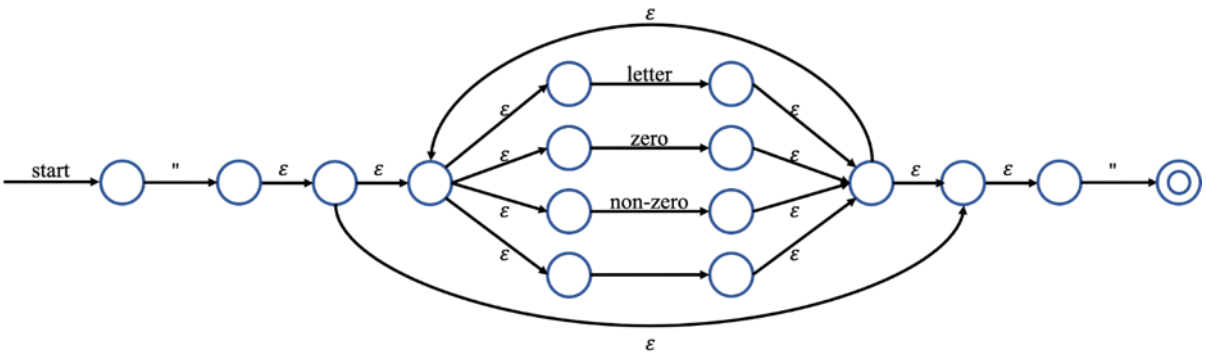
- $(\ letter \mid zero \mid non - zero \mid \ )$



- $(\ letter \mid zero \mid non - zero \mid \ )^*$



- $" \ (\ letter \mid zero \mid non - zero \mid \ )^* \ "$

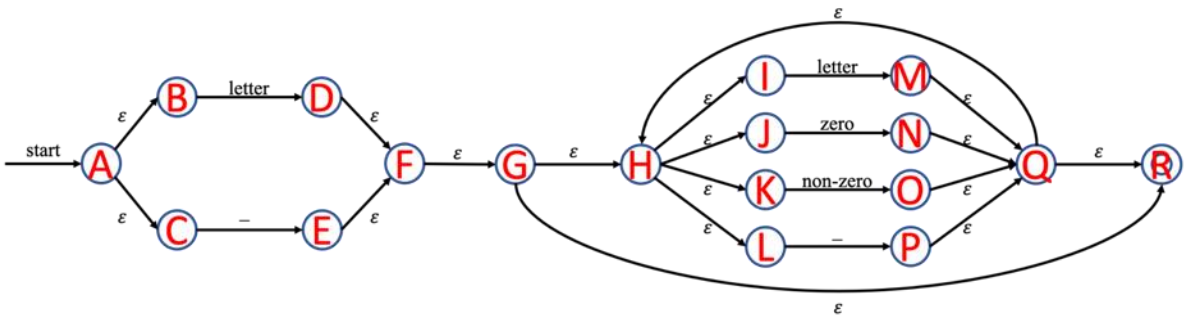


DFA (Deterministic Finite Automata)

We built DFA graph and transition table using subset (powerset) construction algorithm.

If you want to see the hand-writing paper, please check the appendix 1.

1. ID

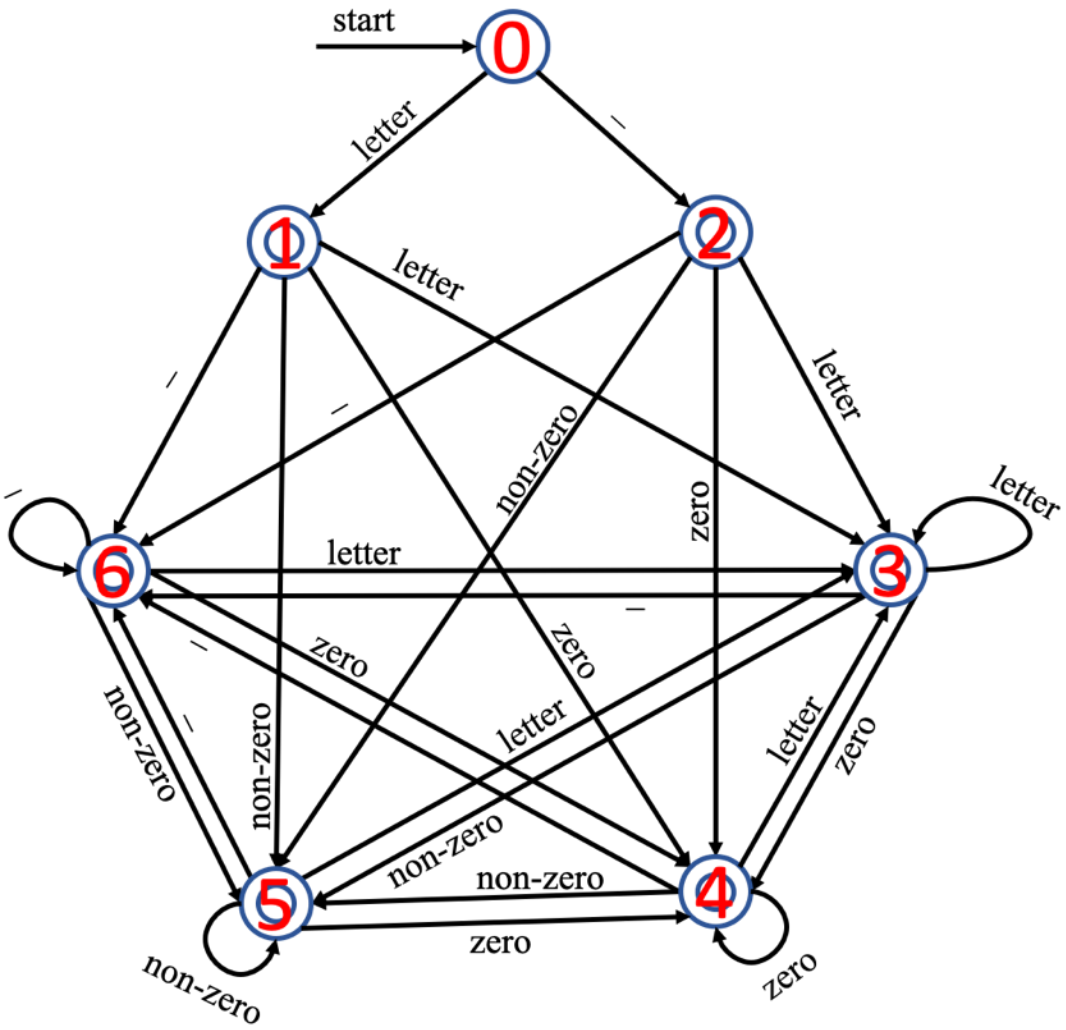


$\varepsilon - closure(A)$ = {A, B, C} = T_0
$\varepsilon - closure(\delta(T_0, letter))$ = $\varepsilon - closure(D)$ = {D, F, G, H, I, J, K, L, R} = T_1
$\varepsilon - closure(\delta(T_0, zero))$ = $\emptyset$
$\varepsilon - closure(\delta(T_0, non\_zero))$ = $\emptyset$
$\varepsilon - closure(\delta(T_0, \_))$ = $\varepsilon - closure(E)$ = {E, F, G, H, I, J, K, L, R} = T_2
$\varepsilon - closure(\delta(T_1, letter))$ = $\varepsilon - closure(M)$ = {M, Q, R, H, I, J, K, L} = T_3
$\varepsilon - closure(\delta(T_1, zero))$ = $\varepsilon - closure(N)$ = {N, Q, R, H, I, J, K, L} = T_4
$\varepsilon - closure(\delta(T_1, non\_zero))$ = $\varepsilon - closure(O)$= {O, Q, R, H, I, J, K, L} = T_5
$\varepsilon - closure(\delta(T_1, \_))$ = $\varepsilon - closure(P)$ = {P, Q, R, H, I, J, K, L} = T_6
$\varepsilon - closure(\delta(T_2, letter))$ = $\varepsilon - closure(M)$ = T_3
$\varepsilon - closure(\delta(T_2, zero))$ = $\varepsilon - closure(N)$ = T_4
$\varepsilon - closure(\delta(T_2, non\_zero))$ = $\varepsilon - closure(O)$ = T_5
$\varepsilon - closure(\delta(T_2, \_))$ = $\varepsilon - closure(P)$ = T_6
$\varepsilon - closure(\delta(T_3, letter))$ = $\varepsilon - closure(M)$ = T_3
$\varepsilon - closure(\delta(T_3, zero))$ = $\varepsilon - closure(N)$ = T_4
$\varepsilon - closure(\delta(T_3, non\_zero))$ = $\varepsilon - closure(O)$ = T_5
$\varepsilon - closure(\delta(T_3, \_))$ = $\varepsilon - closure(P)$ = T_6
$\varepsilon - closure(\delta(T_4, letter))$ = $\varepsilon - closure(M)$ = T_3
$\varepsilon - closure(\delta(T_4, zero))$ = $\varepsilon - closure(N)$ = T_4
$\varepsilon - closure(\delta(T_4, non\_zero))$ = $\varepsilon - closure(O)$ = T_5
$\varepsilon - closure(\delta(T_4, \_))$ = $\varepsilon - closure(P)$ = T_6
$\varepsilon - closure(\delta(T_5, letter))$ = $\varepsilon - closure(M)$ = T_3
$\varepsilon - closure(\delta(T_5, zero))$ = $\varepsilon - closure(N)$ = T_4
$\varepsilon - closure(\delta(T_5, non\_zero))$ = $\varepsilon - closure(O)$ = T_5
$\varepsilon - closure(\delta(T_5, \_))$ = $\varepsilon - closure(P)$ = T_6
$\varepsilon - closure(\delta(T_6, letter))$ = $\varepsilon - closure(M)$ = T_3
$\varepsilon - closure(\delta(T_6, zero))$ = $\varepsilon - closure(N)$ = T_4
$\varepsilon - closure(\delta(T_6, non\_zero))$ = $\varepsilon - closure(O)$ = T_5
$\varepsilon - closure(\delta(T_6, \_))$ = $\varepsilon - closure(P)$ = T_6

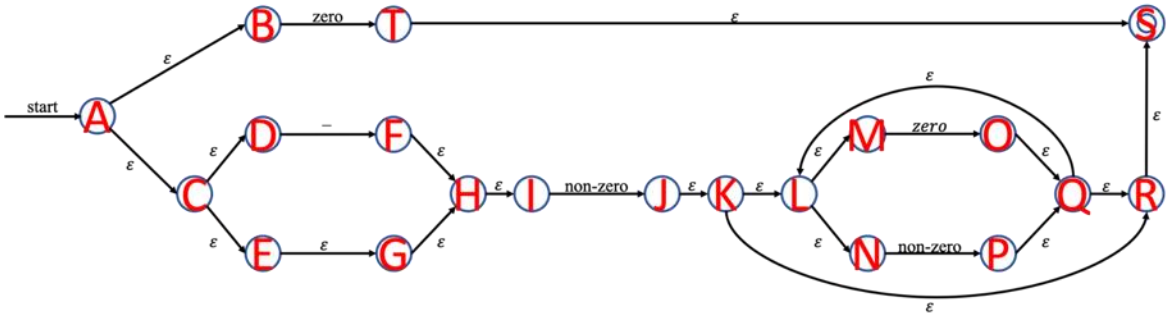
- Transition table

	letter	zero	non-zero	_
T0	T1	∅	∅	T2
T1	T3	T4	T5	T6
T2	T3	T4	T5	T6
T3	T3	T4	T5	T6
T4	T3	T4	T5	T6
T5	T3	T4	T5	T6
T6	T3	T4	T5	T6

- DFA Graph



2. INTEGER

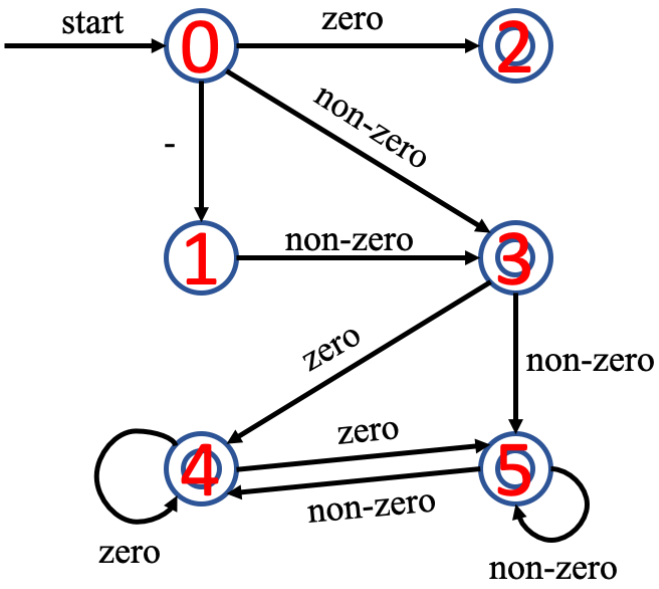


$\varepsilon - closure(A)$ = {A, B, C, D, E, G, H, I} = T_0
$\varepsilon - closure(\delta(T_0, -))$ = $\varepsilon - closure(F)$ = {F, H, I} = T_1
$\varepsilon - closure(\delta(T_0, zero))$ = $\varepsilon - closure(T)$ = {T, S} =T_2
$\varepsilon - closure(\delta(T_0, non\_zero))$ = $\varepsilon - closure(J)$ = {J, K, L, M, N, R, S} =T_3
$\varepsilon - closure(\delta(T_1, -))$ = $\emptyset$
$\varepsilon - closure(\delta(T_1, zero))$ = $\emptyset$
$\varepsilon - closure(\delta(T_1, non\_zero))$ = $\varepsilon - closure(J)$= T_3
$\varepsilon - closure(\delta(T_2, -))$ = $\emptyset$
$\varepsilon - closure(\delta(T_2, zero))$ = $\emptyset$
$\varepsilon - closure(\delta(T_2, non\_zero))$ = $\emptyset$
$\varepsilon - closure(\delta(T_3, -))$ = $\emptyset$
$\varepsilon - closure(\delta(T_3, zero))$ = $\varepsilon - closure(O)$ = {O, Q, L, M, N, R, S} = T_4
$\varepsilon - closure(\delta(T_3, non\_zero))$ = $\varepsilon - closure(P)$ = {P, Q, L, M, N, R, S} = T_5
$\varepsilon - closure(\delta(T_4, -))$ = $\emptyset$
$\varepsilon - closure(\delta(T_4, zero))$ = $\varepsilon - closure(O)$ = T_4
$\varepsilon - closure(\delta(T_4, non\_zero))$ = $\varepsilon - closure(P)$ = T_5
$\varepsilon - closure(\delta(T_5, -))$ = $\emptyset$
$\varepsilon - closure(\delta(T_5, zero))$ = $\varepsilon - closure(O)$ = T_4
$\varepsilon - closure(\delta(T_5, non\_zero))$ = $\varepsilon - closure(P)$ = T_5

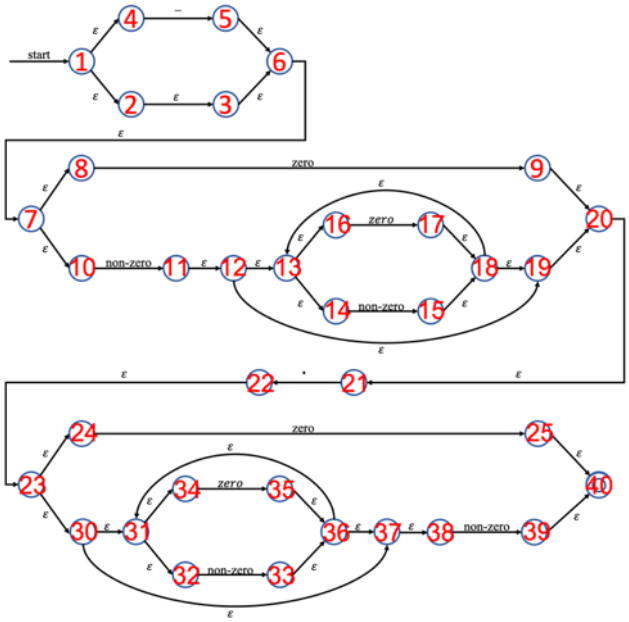
- Transition table

	-	zero	non-zero
T_0	T_1	T_2	T_3
T_1	∅	∅	T_3
T_2	∅	∅	∅
T_3	∅	T_4	T_5
T_4	∅	T_4	T_5
T_5	∅	T_4	T_5

- DFA Graph



3. FLOAT



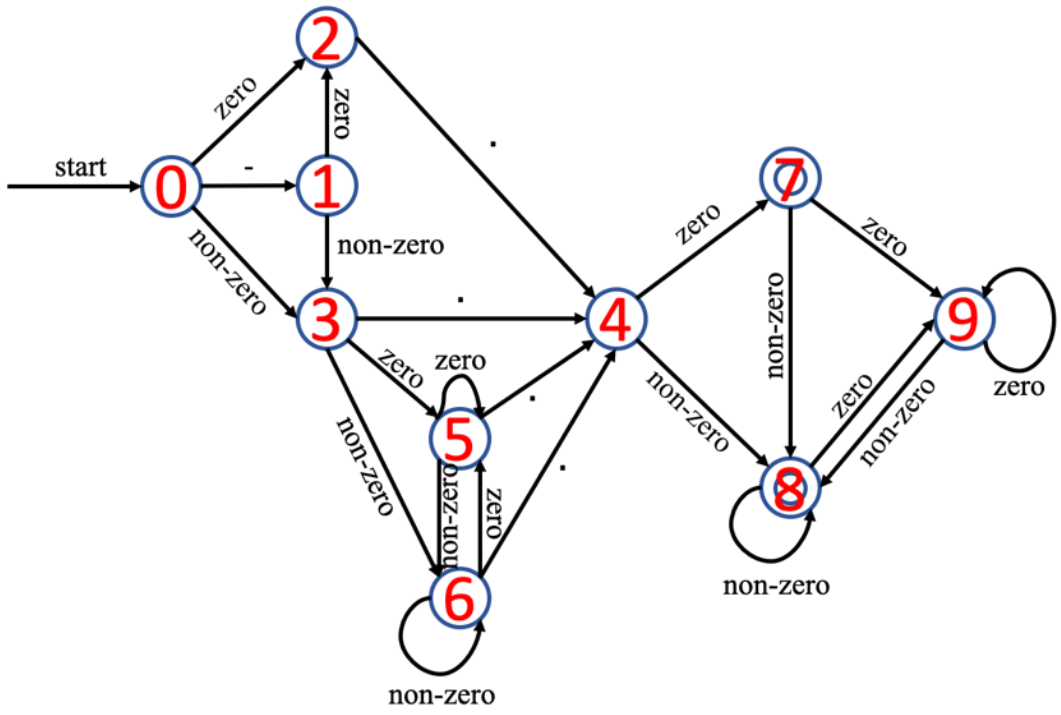
$\varepsilon - closure(1)$ = {1, 2, 3, 4, 6, 7, 8, 10} = T_0
$\varepsilon - closure(\delta(T_0, -))$ = $\varepsilon - closure(5)$ = {5, 6, 7, 8, 10} = T_1
$\varepsilon - closure(\delta(T_0, zero))$ = $\varepsilon - closure(9)$ = {9, 20, 21} =T_2
$\varepsilon - closure(\delta(T_0, non\_zero))$ = $\varepsilon - closure(11)$ = {11, 12, 13, 14, 16, 19, 20, 21} =T_3
$\varepsilon - closure(\delta(T_0, .))$ = $\emptyset$
$\varepsilon - closure(\delta(T_1, -))$ = $\emptyset$
$\varepsilon - closure(\delta(T_1, zero))$ = $\varepsilon - closure(9)$ = T_2
$\varepsilon - closure(\delta(T_1, non\_zero))$ = $\varepsilon - closure(11)$ = T_3
$\varepsilon - closure(\delta(T_1, .))$ = $\emptyset$
$\varepsilon - closure(\delta(T_2, -))$ = $\emptyset$
$\varepsilon - closure(\delta(T_2, zero))$ = $\emptyset$
$\varepsilon - closure(\delta(T_2, non\_zero))$ = $\emptyset$
$\varepsilon - closure(\delta(T_2, .))$ = $\varepsilon - closure(22)$ = {22, 23, 24, 30, 31, 32, 34, 37, 38} = T_4
$\varepsilon - closure(\delta(T_3, -))$ = $\emptyset$
$\varepsilon - closure(\delta(T_3, zero))$ = $\varepsilon - closure(15)$ = {13, 14, 15, 16, 18, 19, 20, 21} = T_5
$\varepsilon - closure(\delta(T_3, non\_zero))$ = $\varepsilon - closure(17)$ = {13, 14, 16, 17, 18, 19, 20, 21} = T_6
$\varepsilon - closure(\delta(T_3, .))$ = $\varepsilon - closure(22)$ = T_4
$\varepsilon - closure(\delta(T_4, -))$ = $\emptyset$
$\varepsilon - closure(\delta(T_4, zero))$ = $\varepsilon - closure(25, 33)$ = {25, 31, 32, 33, 34, 36, 37, 38, 40} = T_7
$\varepsilon - closure(\delta(T_4, non\_zero))$ = $\varepsilon - closure(35, 39)$ = {31, 32, 34, 35, 36, 37, 38, 39, 40} = T_8
$\varepsilon - closure(\delta(T_4, .))$ = $\emptyset$
$\varepsilon - closure(\delta(T_5, -))$ = $\emptyset$
$\varepsilon - closure(\delta(T_5, zero))$ = $\varepsilon - closure(15)$ = T_5
$\varepsilon - closure(\delta(T_5, non\_zero))$ = $\varepsilon - closure(17)$ = T_6
$\varepsilon - closure(\delta(T_5, .))$ = $\varepsilon - closure(22)$ = T_4
$\varepsilon - closure(\delta(T_6, -))$ = $\emptyset$
$\varepsilon - closure(\delta(T_6, zero))$ = $\varepsilon - closure(15)$ = T_5
$\varepsilon - closure(\delta(T_6, non\_zero))$ = $\varepsilon - closure(17)$ = T_6
$\varepsilon - closure(\delta(T_6, .))$ = $\varepsilon - closure(22)$ = T_4
$\varepsilon - closure(\delta(T_7, -))$ = $\emptyset$
$\varepsilon - closure(\delta(T_7, zero))$ = $\varepsilon - closure(33)$ = {31, 32, 33, 34, 36, 37, 38} = T_9

$\varepsilon - closure\big(\delta(T_7, non\_zero)\big) = \varepsilon - closure(35, 39) = T_8$
$\varepsilon - closure\big(\delta(T_7, .)\big) = \emptyset$
$\varepsilon - closure\big(\delta(T_8, -)\big) = \emptyset$
$\varepsilon - closure(\delta(T_8, zero)) = \varepsilon - closure(33) = T_9$
$\varepsilon - closure\big(\delta(T_8, non\_zero)\big) = \varepsilon - closure(35, 39) = T_8$
$\varepsilon - closure\big(\delta(T_8, .)\big) = \emptyset$
$\varepsilon - closure\big(\delta(T_9, -)\big) = \emptyset$
$\varepsilon - closure(\delta(T_9, zero)) = \varepsilon - closure(33) = T_9$
$\varepsilon - closure\big(\delta(T_9, non\_zero)\big) = \varepsilon - closure(35, 39) = T_8$
$\varepsilon - closure\big(\delta(T_9, .)\big) = \emptyset$

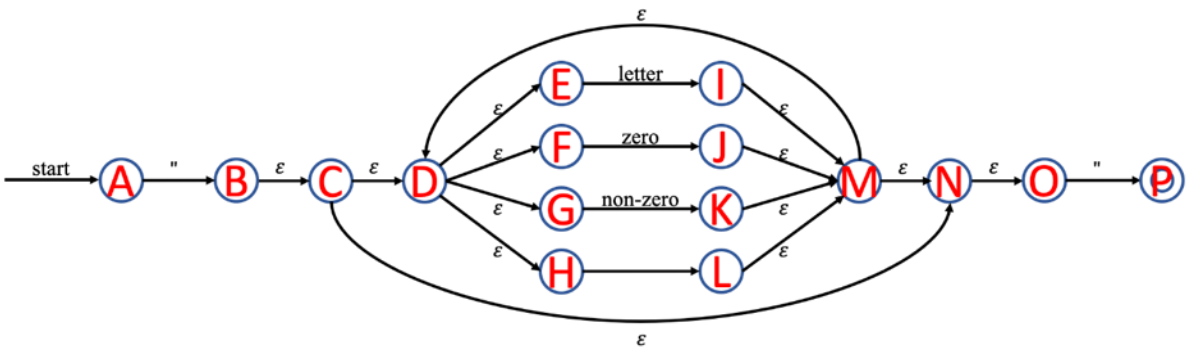
- Transition table

	-	zero	.	non-zero
T_0	T_1	T_2	Ø	T_3
T_1	Ø	T_2	Ø	T_3
T_2	Ø	Ø	T_4	Ø
T_3	Ø	T_5	T_4	T_6
T_4	Ø	T_7	Ø	T_8
T_5	Ø	T_5	T_4	T_6
T_6	Ø	T_5	T_4	T_6
T_7	Ø	T_9	Ø	T_8
T_8	Ø	T_9	Ø	T_8
T_9	Ø	T_9	Ø	T_8

- DFA Graph



4. LITERAL

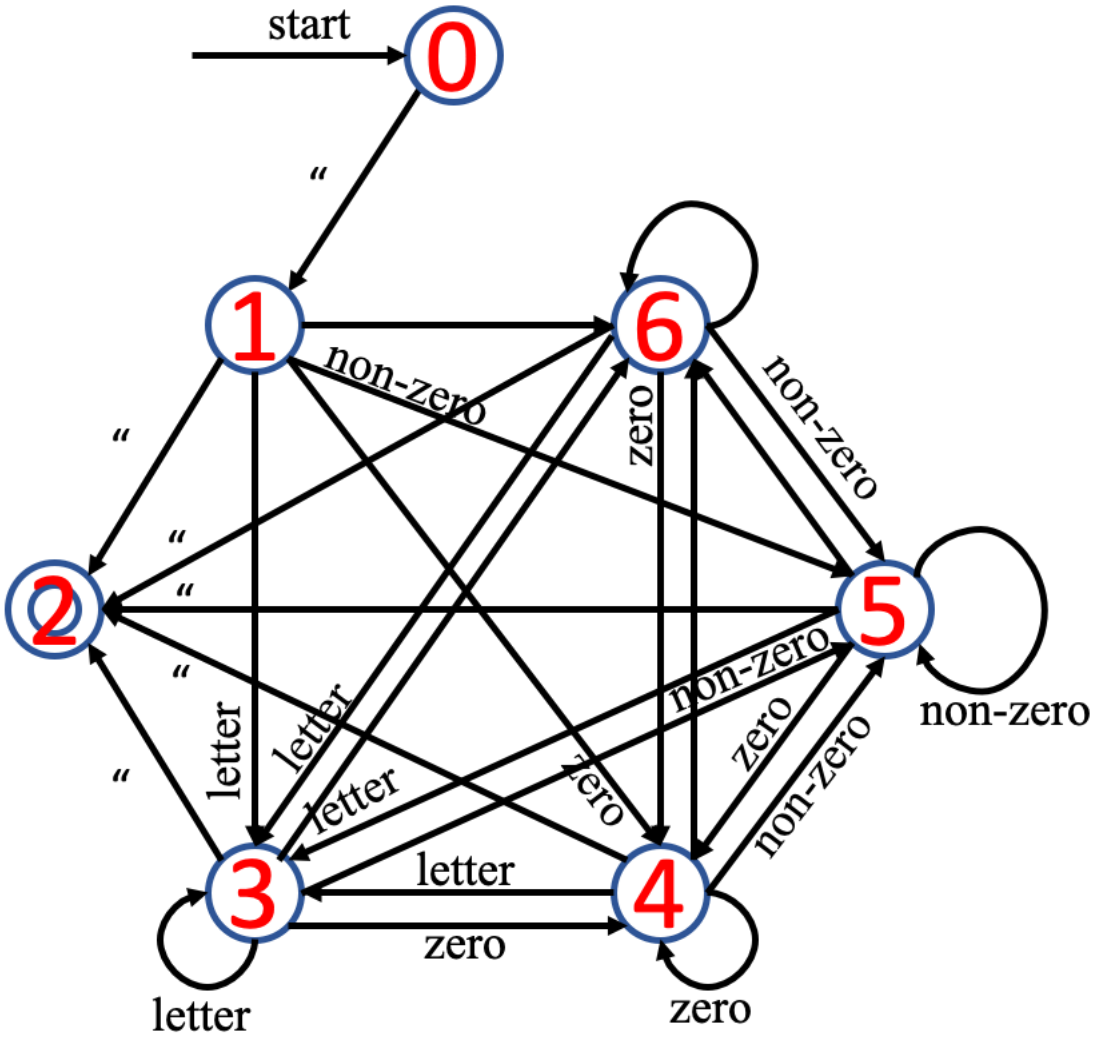


$\varepsilon - closure(A)$ = {A} = T_0
$\varepsilon - closure(\delta(T_0, "))$ = $\varepsilon - closure(B)$ = {B, C, D, E, F, G, H, N, O} = T_1
$\varepsilon - closure(\delta(T_0, letter))$ = $\emptyset$
$\varepsilon - closure(\delta(T_0, zero))$ = $\emptyset$
$\varepsilon - closure(\delta(T_0, non\_zero))$ = $\emptyset$
$\varepsilon - closure(\delta(T_0,))$ = $\emptyset$
$\varepsilon - closure(\delta(T_1, "))$ = $\varepsilon - closure(P)$ = {P} = T_2
$\varepsilon - closure(\delta(T_1, letter))$ = $\varepsilon - closure(I)$ = {I, M, N, O, D, E, F, G, H} = T_3
$\varepsilon - closure(\delta(T_1, zero))$ = $\varepsilon - closure(J)$ = {J, M, N, O, D, E, F, G, H} = T_4
$\varepsilon - closure(\delta(T_1, non\_zero))$ = $\varepsilon - closure(K)$= {K, M, N, O, D, E, F, G, H} = T_5
$\varepsilon - closure(\delta(T_1,))$ = $\varepsilon - closure(L)$= {L, M, N, O, D, E, F, G, H} = T_6
$\varepsilon - closure(\delta(T_2, "))$ = $\emptyset$
$\varepsilon - closure(\delta(T_2, letter))$ = $\emptyset$
$\varepsilon - closure(\delta(T_2, zero))$ = $\emptyset$
$\varepsilon - closure(\delta(T_2, non\_zero))$ = $\emptyset$
$\varepsilon - closure(\delta(T_2,))$ = $\emptyset$
$\varepsilon - closure(\delta(T_3, "))$ = $\varepsilon - closure(P)$ = T_2
$\varepsilon - closure(\delta(T_3, letter))$ = $\varepsilon - closure(I)$ = T_3
$\varepsilon - closure(\delta(T_3, zero))$ = $\varepsilon - closure(J)$ = T_4
$\varepsilon - closure(\delta(T_3, non\_zero))$ = $\varepsilon - closure(K)$ = T_5
$\varepsilon - closure(\delta(T_4,))$ = $\varepsilon - closure(L)$ = T_6
$\varepsilon - closure(\delta(T_4, "))$ = $\varepsilon - closure(P)$ = T_2
$\varepsilon - closure(\delta(T_4, letter))$ = $\varepsilon - closure(I)$ = T_3
$\varepsilon - closure(\delta(T_4, zero))$ = $\varepsilon - closure(J)$ = T_4
$\varepsilon - closure(\delta(T_4, non\_zero))$ = $\varepsilon - closure(K)$ = T_5
$\varepsilon - closure(\delta(T_5,))$ = $\varepsilon - closure(L)$ = T_6
$\varepsilon - closure(\delta(T_5, "))$ = $\varepsilon - closure(P)$ = T_2
$\varepsilon - closure(\delta(T_5, letter))$ = $\varepsilon - closure(I)$ = T_3
$\varepsilon - closure(\delta(T_5, zero))$ = $\varepsilon - closure(J)$ = T_4
$\varepsilon - closure(\delta(T_5, non\_zero))$ = $\varepsilon - closure(K)$ = T_5
$\varepsilon - closure(\delta(T_5,))$ = $\varepsilon - closure(L)$ = T_6
$\varepsilon - closure(\delta(T_6, "))$ = $\varepsilon - closure(P)$ = T_2
$\varepsilon - closure(\delta(T_6, letter))$ = $\varepsilon - closure(I)$ = T_3
$\varepsilon - closure(\delta(T_6, zero))$ = $\varepsilon - closure(J)$ = T_4
$\varepsilon - closure(\delta(T_6, non\_zero))$ = $\varepsilon - closure(K)$ = T_5
$\varepsilon - closure(\delta(T_6,))$ = $\varepsilon - closure(L)$ = T_6

- Transition table

	“	letter	zero	non-zero	
T0	T1	∅	∅	∅	∅
T1	T2	T3	T4	T5	T6
T2	∅	∅	∅	∅	∅
T3	T2	T3	T4	T5	T6
T4	T2	T3	T4	T5	T6
T5	T2	T3	T4	T5	T6
T6	T2	T3	T4	T5	T6

- DFA Graph



TROUBLE & SOLUTION

1. Longest matching

- Trouble

  When we tested identifier that includes keyword and variable (e.g. if, while, int, char), longest matching didn't work well.

Input	Expected result	Result
`int main() {` `  int int_ABC = 3;` `  int intABC = 4;` `  float ifABC = 3.3;` `  bool while_abc = true;` `  char while3ABC = "HI";` `  return 0;` `}`	..... VARIABLE int ID int_ABC ASSIGN = ...	..... ['VARIABLE', 'int'] ['VARIABLE', 'int'] ['ID', '_ABC'] ['ASSIGN', '='] .....

- Solution

  We add more if condition in variable, keyword, logic part so that our program can do longest matching.

```
# Variable, Keyword, Logic
if sub_string in self.LETTER:
    if c == "":
        c = self.input_stream.read(1)
        flag = False

    while (c in self.LETTER):
        sub_string = sub_string + c
        c = self.input_stream.read(1)

    if c == '_':
        flag = False
        continue

    if c in self.DIGIT:
        flag = False
        continue
```

revise

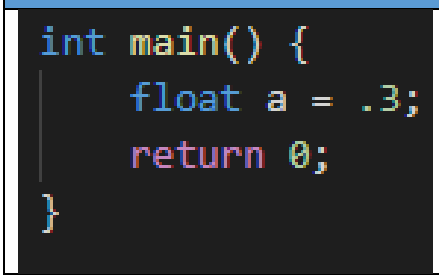
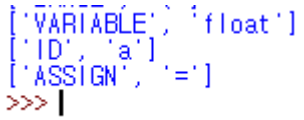
- Result

Input	Previous result	Revised result
`int main() {` `  int int_ABC = 3;` `  int intABC = 4;` `  float ifABC = 3.3;` `  bool while_abc = true;` `  char while3ABC = "HI";` `  return 0;` `}`	..... ['VARIABLE', 'int'] ['VARIABLE', 'int'] ['ID', '_ABC'] ['ASSIGN', '=']	VARIABLE int ID main PAREN (PAREN) BRACE { VARIABLE int ID int_ABC ASSIGN = INT 3 TERM ; VARIABLE int ID intABC ASSIGN = INT 4 TERM ; VARIABLE float ID ifABC ASSIGN = FLOAT 3.3 TERM ; VARIABLE bool ID while_abc ASSIGN = LOGIC true TERM ; VARIABLE char ID while3ABC ASSIGN = LITERAL "HI" TERM ; KEYWORD return INT 0 TERM ; BRACE }

2. Period

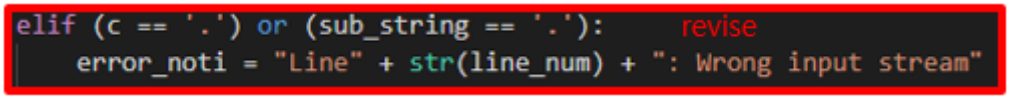
- Trouble

When we tested . (Period) without any prefix, our program is finished that line.

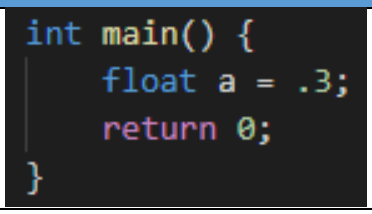
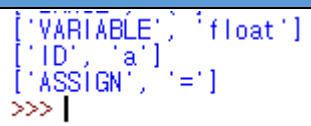
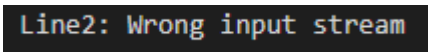
Input	Expected result	Result
	Line2 error!	 Program is finished unexpected.

- Solution

At first, our program didn't include the condition that what we did if the first substring is . (Period). So, we add . (Period) condition in our program.



- Result

Input	Previous result	Revised result
	 Program is finished unexpected.	

IMPLEMENTATION

Before explaining our works, we will introduce the developing environment and how to manage our project, please check the appendix-2.

Language
- Python3 (version: 3.7.4)

Operating System
- macOS Catalina
- Windows 10

IDE (Integrated Development Environment)
- Visual Studio Code (version: 1.45.0)
- PyCharm (version: 3.9.4)

Project Management
- Git (version: 2.24.2)
- Git-Hub

We defined the four tokens: ID, INTEGER, FLOAT, LITERAL as function, and the others are defined as a list type variable. In addition, we defined five symbols; LETTER, SYMBOL, ZERO, NON_ZERO and DIGIT to improve the productivity and readability.

1. Definition of Tokens, Alphabet

We defined the four tokens: ID, INTEGER, FLOAT, LITERAL as function, and the others are defined as a list type variable. In addition, we defined five symbols; LETTER, SYMBOL, ZERO, NON_ZERO and DIGIT to improve the productivity and readability.

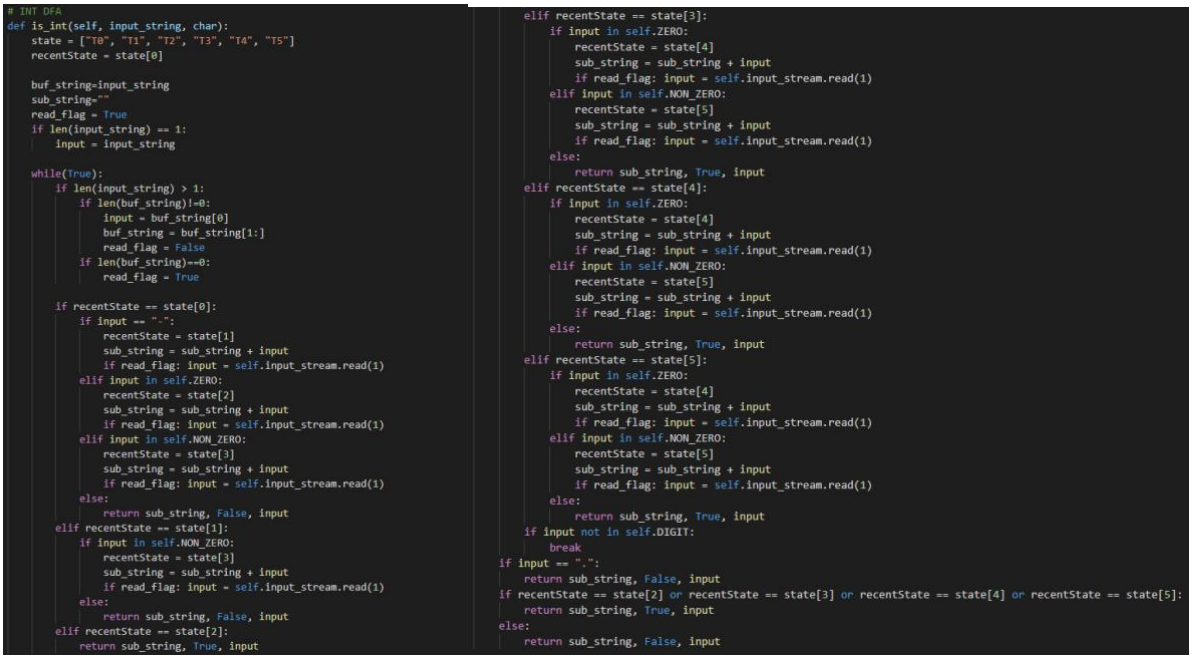
```
# Token Definition
VARIABLE = ['int', 'float','char', 'bool']
KEYWORD = ['if', 'else', 'while', 'for', 'return']
LOGIC = ['true', 'false']
OPERATOR = ['-', '+', '*', '/', '<<', '>>', '&', '|']
COMPARISON = ['<', '>', '==', '!=', '<=', '>=']
WHITESPACE = ['\t', '\n', ' ']
BRACE = ['{', '}']
PAREN = ['(', ')']
ASSIGN = ['=']
TERM = [';']
COMMA = [',']
MERGE = BRACE + PAREN + TERM + COMMA + OPERATOR[1:] + COMPARISON

# Alphabet Definition
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']
SYMBOL = ['&', '|', '+', '-', '*', '/', '<', '>', '"', '.', '_', '!'] + BRACE + PAREN + ASSIGN + TERM + COMMA
ZERO = ['0']
NON_ZERO = ['1', '2', '3', '4', '5', '6', '7', '8', '9']
DIGIT = ZERO + NON_ZERO
ALPHABET = LETTER + DIGIT + SYMBOL + WHITESPACE
```

2. ID-DFA

```
# ID DFA
def is_id(self, input_string, char):
    sub_string = ""
    symbol = self.LETTER + self.ZERO + self.NON_ZERO + ['_']
    i = 0; j = 0
    final = [1, 2, 3, 4, 5, 6]
    transition_table = [[1, -1, -1, 2], [3, 4, 5, 6], [3, 4, 5, 6], [3, 4, 5, 6], \
                        [3, 4, 5, 6], [3, 4, 5, 6], [3, 4, 5, 6]]

    if char == "":
        char = self.input_stream.read(1)

    # Read string
    while char in symbol:
        input_string = input_string + char
        char = self.input_stream.read(1)

    # Analyze
    for c in input_string:
        if c in self.LETTER: j = 0
        elif c in self.ZERO: j = 1
        elif c in self.NON_ZERO: j = 2
        elif c in ['_']: j = 3
        else:
            return sub_string, False, char

        tmp_i = i
        i = transition_table[i][j]
        sub_string = sub_string + c

        if i == -1:
            return sub_string, False, char

    if i in final:
        return sub_string, True, char
    else:
        return sub_string, False, char
```

This part of code is for ID DFA that decides whether the input_string is ID or not. This method has 7 states, and the start state is 0. In a for-loop, it works exactly the same as the ID DFA that was designed in the previous page. This method reads input_string in order, and a recent state is changed by input_string. If it is finished in the final states, which are 1 to 6, the input_string will be accepted. Otherwise, the input_string will be denied, and it means the input_string isn't ID.

3. INTEGER - DFA



```
# INT DFA
def is_int(self, input_string, char):
    state = ["T0", "T1", "T2", "T3", "T4", "T5"]
    recentState = state[0]

    buf_string=input_string
    sub_string=""
    read_flag = True
    if len(input_string) == 1:
        input = input_string

    while(True):
        if len(input_string) > 1:
            if len(buf_string)!=0:
                input = buf_string[0]
                buf_string = buf_string[1:]
                read_flag = False
            if len(buf_string)==0:
                read_flag = True

        if recentState == state[0]:
            if input == ".":
                recentState = state[1]
                sub_string = sub_string + input
                if read_flag: input = self.input_stream.read(1)
            elif input in self.ZERO:
                recentState = state[2]
                sub_string = sub_string + input
                if read_flag: input = self.input_stream.read(1)
            elif input in self.NON_ZERO:
                recentState = state[3]
                sub_string = sub_string + input
                if read_flag: input = self.input_stream.read(1)
            else:
                return sub_string, False, input
        elif recentState == state[1]:
            if input in self.NON_ZERO:
                recentState = state[3]
                sub_string = sub_string + input
                if read_flag: input = self.input_stream.read(1)
            else:
                return sub_string, False, input
        elif recentState == state[2]:
            return sub_string, True, input
        elif recentState == state[3]:
            if input in self.ZERO:
                recentState = state[4]
                sub_string = sub_string + input
                if read_flag: input = self.input_stream.read(1)
            elif input in self.NON_ZERO:
                recentState = state[5]
                sub_string = sub_string + input
                if read_flag: input = self.input_stream.read(1)
            else:
                return sub_string, True, input
        elif recentState == state[4]:
            if input in self.ZERO:
                recentState = state[4]
                sub_string = sub_string + input
                if read_flag: input = self.input_stream.read(1)
            elif input in self.NON_ZERO:
                recentState = state[5]
                sub_string = sub_string + input
                if read_flag: input = self.input_stream.read(1)
            else:
                return sub_string, True, input
        elif recentState == state[5]:
            if input in self.ZERO:
                recentState = state[4]
                sub_string = sub_string + input
                if read_flag: input = self.input_stream.read(1)
            elif input in self.NON_ZERO:
                recentState = state[5]
                sub_string = sub_string + input
                if read_flag: input = self.input_stream.read(1)
            else:
                return sub_string, True, input
        if input not in self.DIGIT:
            break
    if input == ".":
        return sub_string, False, input
    if recentState == state[2] or recentState == state[3] or recentState == state[4] or recentState == state[5]:
        return sub_string, True, input
    else:
        return sub_string, False, input
```

This part of code is for INT DFA that decides whether the input_string is int or not. This method has 6 states, and the start state is T0. In a for-loop, it works exactly the same as the INT DFA that was designed in the previous page. This method reads input_string in order, and a recent state is changed by input_string. If it is finished in the final states, which are state2, state3, state4, and state5, the input_string will be accepted. Otherwise, the input_string will be denied, and it means input_string isn't int.

4. FLOAT - DFA

```
# FLOAT DFA
def is_float(self, input_string, char):
    state = ["T0", "T1", "T2", "T3", "T4", "T5", "T6", "T7", "T8", "T9"]
    recentState = state[0]

    sub_string1 = ""
    sub_string2 = ""
    buf_string = input_string

    if len(input_string) == 1:
        input = input_string
        input_string = ""
    read_flag = True

    while(True):
        if len(input_string) > 1:
            if len(buf_string)!=0:
                input = buf_string[0]
                buf_string = buf_string[1:]
                read_flag = False
            if len(buf_string)==0:
                read_flag = True

        if recentState == state[7] or recentState == state[8]:
            sub_string1 = sub_string2

        if recentState == state[0]:
            if input == "-":
                recentState = state[1]
                sub_string1 = sub_string1 + input
                sub_string2 = sub_string1
                if read_flag: input = self.input_stream.read(1)
            elif input in self.ZERO:
                recentState = state[2]
                sub_string1 = sub_string1 + input
                sub_string2 = sub_string1
                if read_flag: input = self.input_stream.read(1)
            elif input in self.NON_ZERO:
                recentState = state[3]
                sub_string1 = sub_string1 + input
                sub_string2 = sub_string1
                if read_flag: input = self.input_stream.read(1)
            else:
                buf = sub_string2.replace(sub_string1, "", 1)
                return sub_string1, False, (buf + input)
        elif recentState == state[1]:
            if input in self.ZERO:
                recentState = state[2]
                sub_string2 = sub_string2 + input
                if read_flag: input = self.input_stream.read(1)
            elif input in self.NON_ZERO:
                recentState = state[3]
                sub_string2 = sub_string2 + input
                if read_flag: input = self.input_stream.read(1)
            else:
                buf = sub_string2.replace(sub_string1, "", 1)
                return sub_string1, False, (buf + input)
        elif recentState == state[2]:
            if input == ".":
                recentState = state[4]
                sub_string2 = sub_string2 + input
                if read_flag: input = self.input_stream.read(1)
            else:
                buf = sub_string2.replace(sub_string1, "", 1)
                return sub_string1, False, (buf + input)
        elif recentState == state[3]:
            if input in self.ZERO:
                recentState = state[5]
                sub_string2 = sub_string2 + input
                if read_flag: input = self.input_stream.read(1)
            elif input == ".":
                recentState = state[4]
                sub_string2 = sub_string2 + input
                if read_flag: input = self.input_stream.read(1)
            elif input in self.NON_ZERO:
                recentState = state[6]
                sub_string2 = sub_string2 + input
                if read_flag: input = self.input_stream.read(1)
            else:
                buf = sub_string2.replace(sub_string1, "", 1)
                return sub_string1, False, (buf + input)
        elif recentState == state[4]:
            if input in self.ZERO:
                recentState = state[7]
                sub_string2 = sub_string2 + input
                if read_flag: input = self.input_stream.read(1)
            elif input in self.NON_ZERO:
                recentState = state[8]
                sub_string2 = sub_string2 + input
                if read_flag: input = self.input_stream.read(1)
            else:
                buf = sub_string2.replace(sub_string1, "", 1)
                return sub_string1, False, (buf + input)
```

```
        elif recentState == state[5]:
            if input in self.ZERO:
                recentState = state[5]
                sub_string2 = sub_string2 + input
                if read_flag: input = self.input_stream.read(1)
            elif input == ".":
                recentState = state[4]
                sub_string2 = sub_string2 + input
                if read_flag: input = self.input_stream.read(1)
            elif input in self.NON_ZERO:
                recentState = state[6]
                sub_string2 = sub_string2 + input
                if read_flag: input = self.input_stream.read(1)
            else:
                buf = sub_string2.replace(sub_string1, "", 1)
                return sub_string1, False, (buf + input)
        elif recentState == state[6]:
            if input in self.ZERO:
                recentState = state[5]
                sub_string2 = sub_string2 + input
                if read_flag: input = self.input_stream.read(1)
            elif input == ".":
                recentState = state[4]
                sub_string2 = sub_string2 + input
                if read_flag: input = self.input_stream.read(1)
            elif input in self.NON_ZERO:
                recentState = state[6]
                sub_string2 = sub_string2 + input
                if read_flag: input = self.input_stream.read(1)
            else:
                buf = sub_string2.replace(sub_string1, "", 1)
                return sub_string1, False, (buf + input)
        elif recentState == state[7]:
            if input in self.ZERO:
                recentState = state[9]
                sub_string2 = sub_string2 + input
                if read_flag: input = self.input_stream.read(1)
            elif input in self.NON_ZERO:
                recentState = state[8]
                sub_string2 = sub_string2 + input
                if read_flag: input = self.input_stream.read(1)
            else:
                return sub_string1, True, input
        elif recentState == state[8]:
            if input in self.ZERO:
                recentState = state[9]
                sub_string2 = sub_string2 + input
                if read_flag: input = self.input_stream.read(1)
            elif input in self.NON_ZERO:
                recentState = state[8]
                sub_string2 = sub_string2 + input
                sub_string1 = sub_string2
                if read_flag: input = self.input_stream.read(1)
            else:
                return sub_string1, True, input
        elif recentState == state[9]:
            if input in self.ZERO:
                recentState = state[9]
                sub_string2 = sub_string2 + input
                if read_flag: input = self.input_stream.read(1)
            elif input in self.NON_ZERO:
                recentState = state[8]
                sub_string2 = sub_string2 + input
                if read_flag: input = self.input_stream.read(1)
            else:
                buf = sub_string2.replace(sub_string1, "", 1)
                return sub_string1, False, (buf+input)
        if input not in (self.DIGIT + ['.', '-']):
            break
    if recentState == state[7] or recentState == state[8]:
        sub_string1 = sub_string2
        return sub_string1, True, input
    else:
        buf = sub_string2.replace(sub_string1, "", 1)
        return sub_string1, False, (buf+input)
```

This part of the code is for FLOAT DFA that decides whether the input_string is float or not. This method has 10 states, and the start state is T0. In a for-loop, it works exactly the same as the FLOAT DFA that was designed in the previous page. This method reads input_string in order, and a recent state is changed by input_string. If it is finished in the final states, which are state7 and state8, the input_string will be accepted. Otherwise, the input_string will be denied, and it means input_string isn't a float.

5. LITERAL - DFA

```
# Literal DFA
def is_string(self, input_string, char):
    sub_string = ""
    symbol = self.LETTER + self.ZERO + self.NON_ZERO + [' ', '"']
    i = 0; j = 0
    final = [2]
    transition_table = [[1, -1, -1, -1, -1], [2, 3, 4, 5, 6], [-1, -1, -1, -1, -1], \
                        [2, 3, 4, 5, 6], [2, 3, 4, 5, 6], [2, 3, 4, 5, 6], [2, 3, 4, 5, 6]]

    if char == "":
        char = self.input_stream.read(1)

    while char in symbol:
        input_string = input_string + char
        char = self.input_stream.read(1)

    # Analyze
    for c in input_string:
        if c in '"': j = 0
        elif c in self.LETTER: j = 1
        elif c in self.ZERO: j = 2
        elif c in self.NON_ZERO: j = 3
        elif c in [' ']: j = 4
        else:
            return sub_string, False, char

        i = transition_table[i][j]
        sub_string = sub_string + c

        if i == -1:
            return sub_string, False, char

    if i in final:
        return sub_string, True, char
    else:
        return sub_string, False, char
```

This part of code is for LITERAL DFA that decides whether the input_string is a string or not. This method has 7 states, and the start state is 0. In a for-loop, it works exactly the same as the LITERAL DFA that was designed in the previous page. This method reads input_string in order, and a recent state is changed by input_string. If it is finished in the final state, which is 2, the input_string will be accepted. Otherwise, the input_string will be denied, and it means input_string isn't a string.

6. Other Tokens

The software checks the other tokens following matching processing.

- Variable, Keyword, Logic

```
# Variable, Keyword, Logic
if sub_string in self.LETTER:
    if c == "":
        c = self.input_stream.read(1)
        flag = False

    while (c in self.LETTER):
        sub_string = sub_string + c
        c = self.input_stream.read(1)

    if c == '_':
        flag = False
        continue

    if c in self.DIGIT:
        flag = False
        continue

    if sub_string in self.VARIABLE:
        symbol_table.append(['VARIABLE', sub_string])
        sub_string = ""
        continue
    elif sub_string in self.KEYWORD:
        symbol_table.append(['KEYWORD', sub_string])
        sub_string = ""
        continue
    elif sub_string in self.LOGIC:
        symbol_table.append(['LOGIC', sub_string])
        sub_string = ""
        continue
```

- ASSIGN, COMPARISON

```
# ASSIGN, COMPARISON
if sub_string in self.ASSIGN:
    if c == "":
        c = self.input_stream.read(1)
        flag = False

    # COMPARISON ==
    if sub_string + c in self.COMPARISON:
        symbol_table.append(['COMPARISON', sub_string + c])
        sub_string = ""
        c = ""; flag = True
        continue
    # ASSIGN
    else:
        symbol_table.append(['ASSIGN', sub_string])
        sub_string = ""
        continue
```

- BRACE, PAREN, TERM, COMMA, OPERATOR, COMPARISON

```
# BRACE, PAREN, TERM, COMMA, OPERATOR, COMPARISON
if sub_string in self.MERGE + ['!']:
    if sub_string in ['<', '>']:
        if c == "":
            c = self.input_stream.read(1)
            flag = False
        if c == sub_string:
            symbol_table.append(['OPERATOR', sub_string + c])
            sub_string = ""; flag = True
            continue
        elif sub_string + c in self.COMPARISON:
            symbol_table.append(['COMPARISON', sub_string + c])
            sub_string = ""; flag = True
            continue

    if sub_string == '!':
        if c == "":
            c = self.input_stream.read(1)
            flag = False
        if c == '=':
            symbol_table.append(['COMPARISON', sub_string + c])
            sub_string = ""; flag = True
            continue
        else:
            error_noti = "Line" + str(line_num) + ": Wrong input stream"
            # Open file for writing Error
            try:
                f = open(file_name[:-2]+'_error.out', 'w')
            except:
                print("Fail to write file")
                exit()

            for i in error_noti:
                f.writelines(i)
            f.close()
            print(error_noti)
            exit()

    if sub_string in self.BRACE:
        symbol_table.append(['BRACE', sub_string])
    elif sub_string in self.PAREN:
        symbol_table.append(['PAREN', sub_string])
    elif sub_string in self.TERM:
        symbol_table.append(['TERM', sub_string])
    elif sub_string in self.COMMA:
        symbol_table.append(['COMMA', sub_string])
    elif sub_string in self.OPERATOR:
        symbol_table.append(['OPERATOR', sub_string])
    elif sub_string in self.COMPARISON:
        symbol_table.append(['COMPARISON', sub_string])
    sub_string = ""
    continue
```

- Subtract

```
# Subtract
if sub_string in ['-']:
    # if ('INT' or 'FLOAT' or 'ID') in symbol_table[-1]:
    if ('INT' in symbol_table[-1]) or ('FLOAT' in symbol_table[-1]) or ('ID' in symbol_table[-1]):
        symbol_table.append(['OPERATOR', sub_string])
        sub_string = ""
        continue
```

7. Other - File I/O

- Read File

```
# Open file for reading
try:
    file_name = sys.argv[1]
    f = open(file_name)
except:
    print("Fail to read file")
    exit()
```

```
# Check the character is read or not
if flag:
    c = self.input_stream.read(1)
    flag = True
```

- Write File

```
# Open file for writing result
try:
    f = open(file_name[:-2]+'.out', 'w')
except:
    print("Fail to write file")
    exit()

for i in symbol_table:
    token = i[0]
    lexeme = i[1]
    f.writelines(token + ' ' + lexeme + '\n')
f.close()
```

- Error Case

If the code is written as out of correct grammar, the software returns the error message and creates the error message file as followed format FIEL_NAME_error.out. There are 5 points where check the error; each of them use the same code, so we attached a single picture.

```
error_noti = "Line" + str(line_num) + ": Wrong input stream"
# Open file for writing Error
try:
    f = open(file_name[:-2]+'_error.out', 'w')
except:
    print("Fail to write file")
    exit()

for i in error_noti:
    f.writelines(i)
f.close()
print(error_noti)
exit()
```

TEST CASES & RESULT

1. Correct Test Code

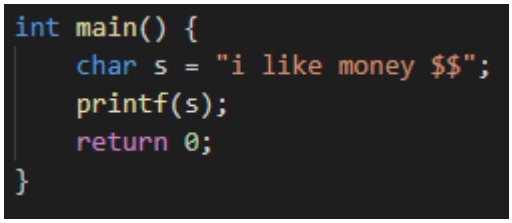
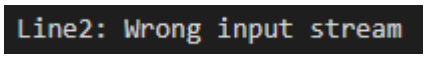
Input	Result

```
int main() {
    int iteration = 5;
    bool logic = true;

    while (iteration) {
        printf(iteration);
        if (iteration == 5){
            printf("First")
        } else {
            printf("Iteration ");
        }
        iteration = iteration - 1;
    }

    return 0;
}
```

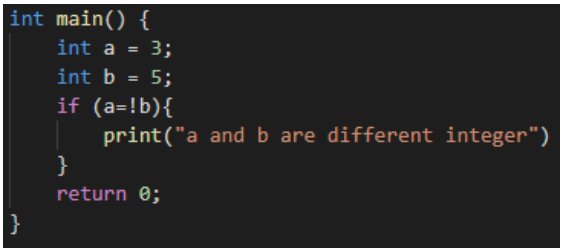
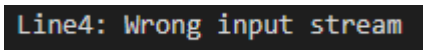
```
VARIABLE int
ID main
PAREN (
PAREN )
BRACE {
VARIABLE int
ID iteration
ASSIGN =
INT 5
TERM ;
VARIABLE bool
ID logic
ASSIGN =
LOGIC true
TERM ;
KEYWORD while
PAREN (
ID iteration
PAREN )
BRACE {
ID printf
PAREN (
ID iteration
PAREN )
TERM ;
KEYWORD if
PAREN (
ID iteration
COMPARISON ==
INT 5
PAREN )
BRACE {
ID printf
PAREN (
LITERAL "First"
PAREN )
BRACE }
KEYWORD else
BRACE {
ID printf
PAREN (
LITERAL "Iteration "
PAREN )
TERM ;
BRACE }
ID iteration
ASSIGN =
ID iteration
OPERATOR -
INT 1
TERM ;
BRACE }
KEYWORD return
INT 0
TERM ;
BRACE }
```

2. Error Test Code

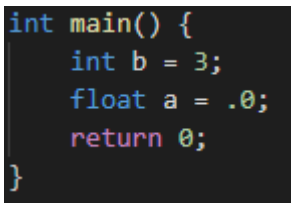
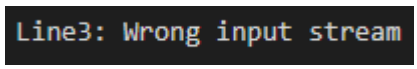
- In our lexical analyzer, we don't allow to use the character '$'.

Input	Result
	

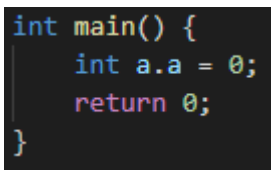
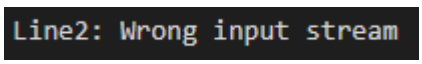
- '=!' isn't correct comparison.

Input	Result
	

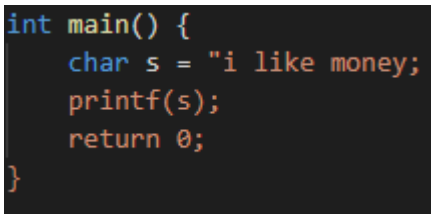
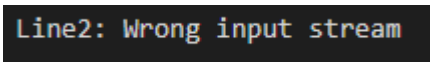
- '.0' isn't correct float.

Input	Result
	

- 'a.a' isn't correct input.

Input	Result
	

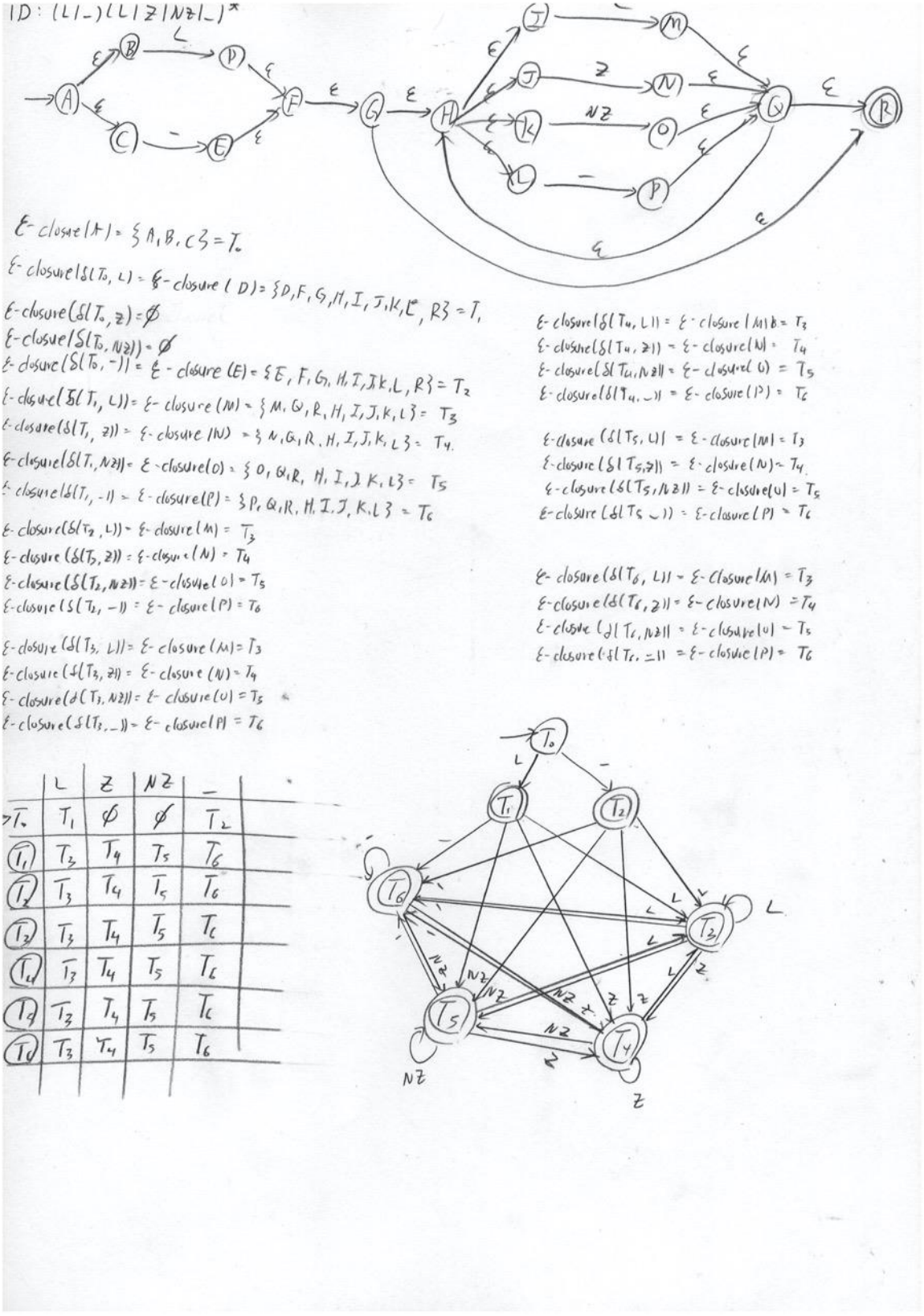
- Literal string should be terminated with a symbol “.

Input	Result
	

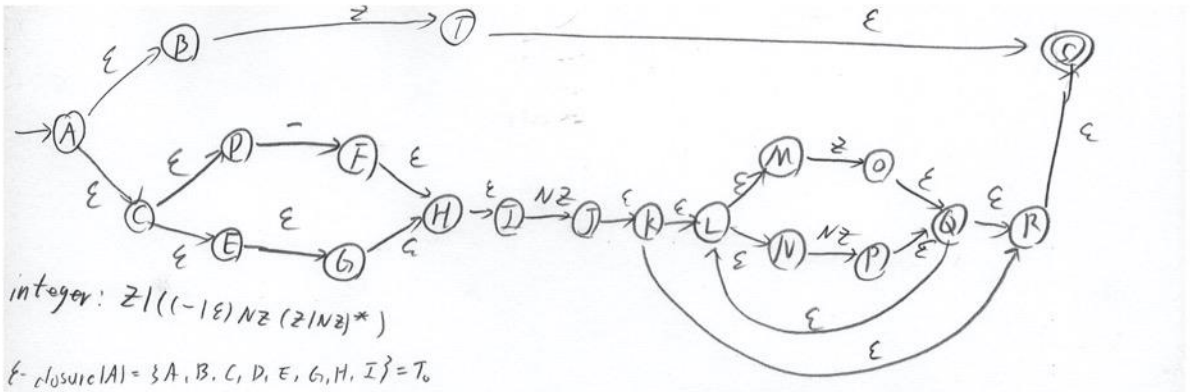
APPENDIX

1. NFA to DFA with transition table

- ID



- INTEGER



integer : Z|((-|ε)NZ (Z|NZ)*)

ε-closure(A) = {A, B, C, D, E, G, H, I} = T_0

ε-closure(δ(T_0, -)) = ε-closure(F) = {F, H, I} = T_1

ε-closure(δ(T_0, Z)) = ε-closure(T) = {T, S} = T_2

ε-closure(δ(T_0, NZ)) = ε-closure(J) = {J, K, L, M, N, R, S} = T_3

ε-closure(δ(T_1, -)) = ∅

ε-closure(δ(T_1, Z)) = ∅

ε-closure(δ(T_1, NZ)) = ε-closure(J) = T_3

ε-closure(δ(T_2, -)) = ∅

ε-closure(δ(T_2, Z)) = ∅

ε-closure(δ(T_2, NZ)) = ∅

ε-closure(δ(T_3, -)) = ∅

ε-closure(δ(T_3, Z)) = ε-closure(O) = {O, Q, L, M, N, P, R, S} = T_4

ε-closure(δ(T_3, NZ)) = ε-closure(P) = {P, Q, L, M, N, P, R, S} = T_4

ε-closure(δ(T_4, -)) = ∅

ε-closure(δ(T_4, Z)) = ε-closure(Q) = T_4

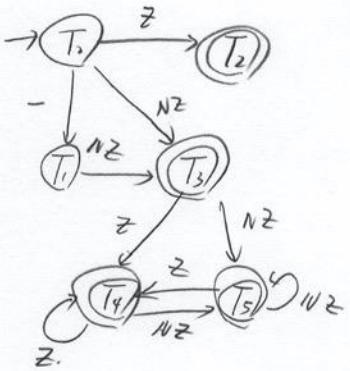
ε-closure(δ(T_4, NZ)) = ε-closure(P) = T_5

ε-closure(δ(T_5, -)) = ∅

ε-closure(δ(T_5, Z)) = T_4

ε-closure(δ(T_5, NZ)) = T_5

	-	Z	NZ
→ T_0	T_1	T_4	T_3
T_1	∅	∅	T_3
(T_2)	∅	∅	∅
(T_3)	∅	T_4	T_5
(T_4)	∅	T_4	T_5
(T_5)	∅	T_4	T_5



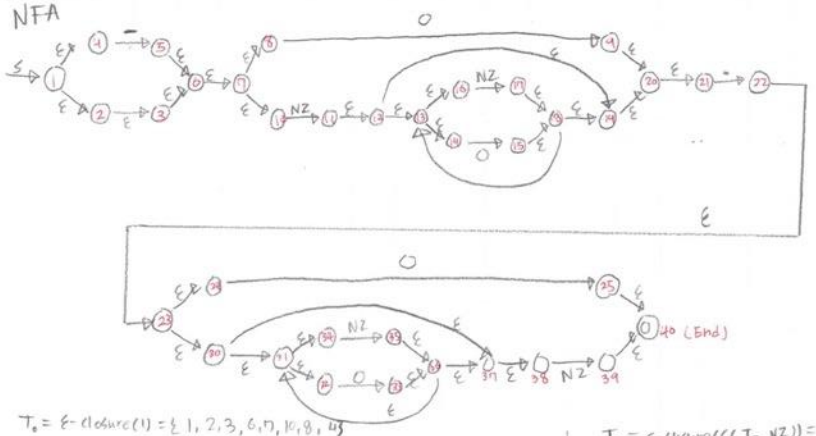
□ FLOAT

$(-|\varepsilon)(0|NZ\cdot(0|NZ)^*)\cdot(0|(0|NZ)^*NZ)$ ⇒ regular expression

NZ = 1|2|3 ... |8|9 Input = {−, 0, ., NZ}

(NON Zero)

NFA



$T_0 = \varepsilon\text{-closure}(1) = \{1,2,3,6,7,10,8,4\}$
$= \{1,2,3,4,6,7,8,10\}$

$T_1 = \varepsilon\text{-closure}(\delta(T_0,-)) = \varepsilon\text{-closure}(5)$
$= \{5,6,7,8,10\}$

$T_2 = \varepsilon\text{-closure}(\delta(T_0,0)) = \varepsilon\text{-closure}(9)$
$= \{9,20,21\}$

$T_3 = \varepsilon\text{-closure}(\delta(T_0,NZ)) = \varepsilon\text{-closure}(11)$
$= \{11,12,13,14,16,17,20,21\}$

$\varepsilon\text{-closure}(\delta(T_1,0)) = \varepsilon\text{-closure}(9)$
$= T_2$

$\varepsilon\text{-closure}(\delta(T_1,NZ)) = \varepsilon\text{-closure}(11)$
$= T_3$

$T_4 = \varepsilon\text{-closure}(\delta(T_2,\cdot)) = \varepsilon\text{-closure}(22)$
$= \{22,23,24,30,31,32,34,37,38\}$

$T_5 = \varepsilon\text{-closure}(\delta(T_3,0)) = \varepsilon\text{-closure}(15)$
$= \{15,18,19,20,21,13,14,16\}$
$= \{13,14,15,16,18,19,20,21\}$

$T_6 = \varepsilon\text{-closure}(\delta(T_3,NZ)) = \varepsilon\text{-closure}(17)$
$= \{17,18,19,20,21,13,14,16\}$
$= \{13,14,16,17,18,19,20,21\}$

$\varepsilon\text{-closure}(\delta(T_3,\cdot)) = \varepsilon\text{-closure}(22)$
$= T_4$

$T_7 = \varepsilon\text{-closure}(\delta(T_4,0)) = \varepsilon\text{-closure}(25,33)$
$= \{25,40,32,36,20,38,31,22,34\}$
$= \{25,31,32,33,34,36,37,38,40\}$

$T_8 = \varepsilon\text{-closure}(\delta(T_4,NZ)) = \varepsilon\text{-closure}(35,24)$
$= \{35,36,20,38,31,32,34,24,40\}$
$= \{31,32,34,35,36,37,38,24,40\}$

$\varepsilon\text{-closure}(\delta(T_5,0)) = \varepsilon\text{-closure}(15)$
$= T_5$

$\varepsilon\text{-closure}(\delta(T_5,NZ)) = \varepsilon\text{-closure}(17)$
$= T_6$

$\varepsilon\text{-closure}(\delta(T_5,\cdot)) = \varepsilon\text{-closure}(22)$
$= T_4$

$\varepsilon\text{-closure}(\delta(T_6,0)) = \varepsilon\text{-closure}(15)$
$= T_5$

$\varepsilon\text{-closure}(\delta(T_6,NZ)) = \varepsilon\text{-closure}(17)$
$= T_6$

$\varepsilon\text{-closure}(\delta(T_6,\cdot)) = \varepsilon\text{-closure}(22)$
$= T_4$

$T_7 = \varepsilon\text{-closure}(\delta(T_7,0)) = \varepsilon\text{-closure}(33)$
$= \{33,36,20,38,31,32,34\}$
$= \{31,32,33,34,36,37,38\}$

$\varepsilon\text{-closure}(\delta(T_7,NZ)) = \varepsilon\text{-closure}(35,24)$
$= T_8$

$\varepsilon\text{-closure}(\delta(T_8,0)) = \varepsilon\text{-closure}(33)$
$= T_7$

$\varepsilon\text{-closure}(\delta(T_8,NZ)) = \varepsilon\text{-closure}(35,24)$
$= T_8$

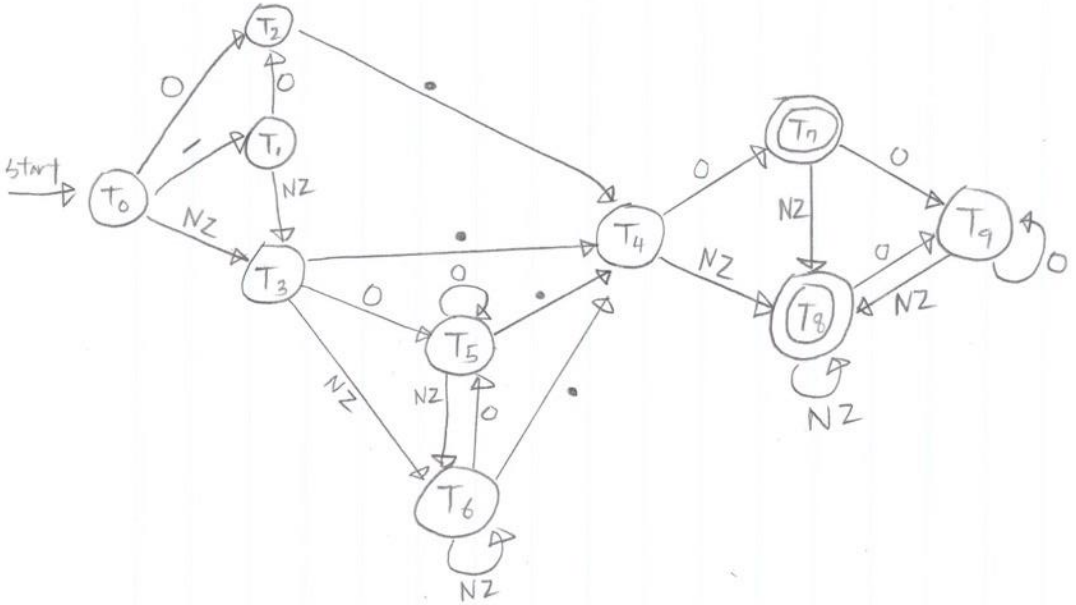
$\varepsilon\text{-closure}(\delta(T_7,0)) = \varepsilon\text{-closure}(33)$
$= T_7$

$\varepsilon\text{-closure}(\delta(T_7,NZ)) = \varepsilon\text{-closure}(35,24)$
$= T_8$

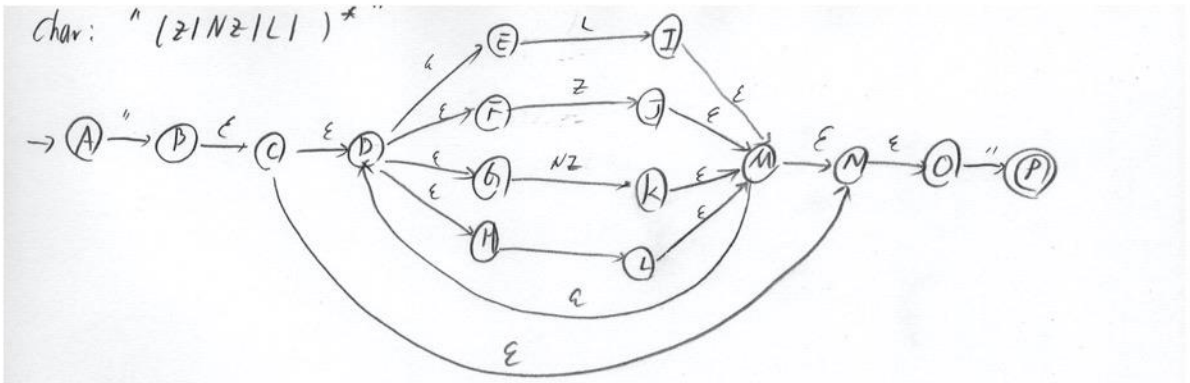
	−	0	.	NZ	
T_0	T_1	T_2	ϕ	T_3	3개
T_1	ϕ	T_2	ϕ	T_3	2개
T_2	ϕ	ϕ	T_4	ϕ	1개
T_3	ϕ	T_5	T_4	T_6	3개
T_4	ϕ	T_7	ϕ	T_8	2개
T_5	ϕ	T_5	T_4	T_6	3개
T_6	ϕ	T_5	T_4	T_6	3개
T_7	ϕ	T_7	ϕ	T_8	2개
T_8	ϕ	T_7	ϕ	T_8	2개
T_9	ϕ	T_7	ϕ	T_8	2개

NZ = 1|2|3 ··· 8|9

Input = {−, 0, •, NZ}



- LITERAL



ε- closure {A} = {A} = T_0

ε-closure(δ(T_0, ")) = ε-closure(B) = {B, C, D, E, F, G, H, N, O} = T_1
ε-closure(δ(T_0, L)) = ∅
ε-closure(δ(T_0, Z)) = ∅
ε-closure(δ(T_0, NZ)) = ∅
ε-closure(δ(T_0,)) = ∅

ε-closure(δ(T_1, ")) = ε-closure(P) = {P} = T_2
ε-closure(δ(T_1, L)) = ε-closure(I) = {I, M, N, O, D, E, F, G, H} = T_3
ε-closure(δ(T_1, Z)) = ε-closure(J) = {J, M, N, O, D, E, F, G, H} = T_4
ε-closure(δ(T_1, NZ)) = ε-closure(K) = {K, M, N, O, D, E, F, G, H} = T_5
ε-closure(δ(T_1,)) = ε-closure(L) = {L, M, N, O, D, E, F, G, H} = T_6

ε-closure(δ(T_2, ")) = ∅
ε-closure(δ(T_2, L)) = ∅
ε-closure(δ(T_2, Z)) = ∅
ε-closure(δ(T_2, NZ)) = ∅
ε-closure(δ(T_2,)) = ∅

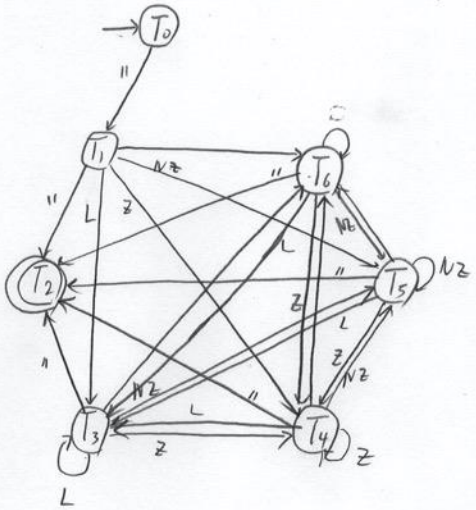
ε-closure(δ(T_3, ")) = ε-closure(P) = T_2
ε-closure(δ(T_3, L)) = ε-closure(I) = T_3
ε-closure(δ(T_3, Z)) = ε-closure(J) = T_4
ε-closure(δ(T_3, NZ)) = ε-closure(K) = T_5
ε-closure(δ(T_3,)) = ε-closure(L) = T_6

ε-closure(δ(T_4, ")) = ε-closure(P) = T_2
ε-closure(δ(T_4, L)) = ε-closure(I) = T_3
ε-closure(δ(T_4, Z)) = ε-closure(J) = T_4
ε-closure(δ(T_4, NZ)) = ε-closure(K) = T_5
ε-closure(δ(T_4,)) = ε-closure(L) = T_6

ε-closure(δ(T_5, ")) = ε-closure(P) = T_2
ε-closure(δ(T_5, L)) = ε-closure(I) = T_3
ε-closure(δ(T_5, Z)) = ε-closure(J) = T_4
ε-closure(δ(T_5, NZ)) = ε-closure(K) = T_5
ε-closure(δ(T_5,)) = ε-closure(L) = T_6

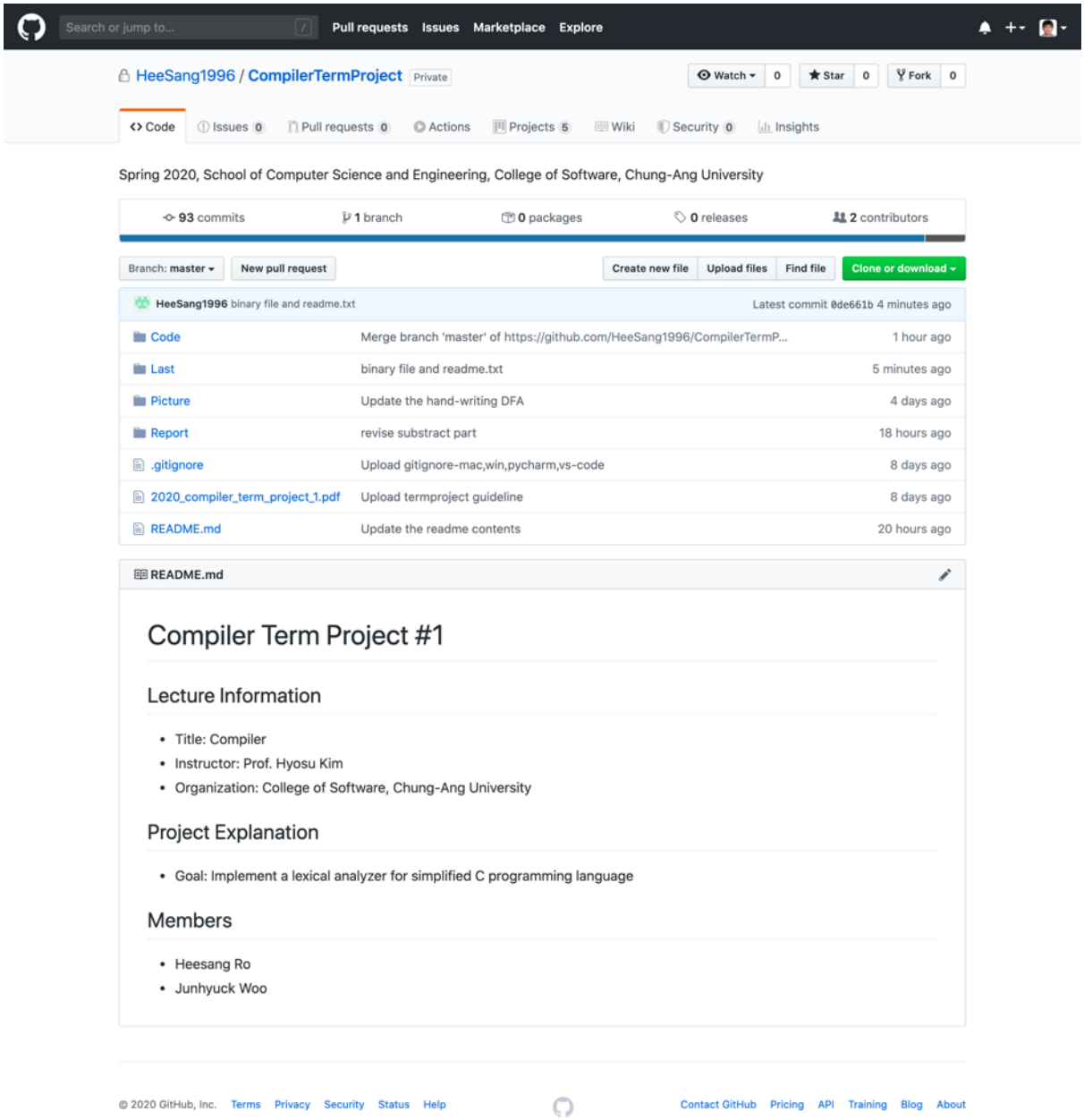
ε-closure(δ(T_6, ")) = ε-closure(P) = T_2
ε-closure(δ(T_6, L)) = ε-closure(I) = T_3
ε-closure(δ(T_6, Z)) = ε-closure(J) = T_4
ε-closure(δ(T_6, NZ)) = ε-closure(K) = T_5
ε-closure(δ(T_6,)) = ε-closure(L) = T_6

	"	L	Z	NZ	
→T_0	T_1	∅	∅	∅	∅
T_1	T_2	T_3	T_4	T_5	T_6
(T_2)	∅	∅	∅	∅	∅
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	T_2	T_3	T_4	T_5	T_6



2. Git-Hub

- Project Overview



- Projects Management