|  |
| --- |
| 컴파일러 01분반 |

-term project 1 –



|  |  |
| --- | --- |
| 과목명 | 컴파일러 |
| 교수명 | 김효수 교수님 |
| 제출일 | 2022.04.28 |
| 학  번 | 20200453, 20201501 |
| 학  과 | 소프트웨어학부 |
| 이  름 | 김세진, 김은솔 |

1. Definition of tokens

Tokens

|  |  |
| --- | --- |
| Token | Lexeme |
| VARIABLE | int, INT, char, CHAR |
| KEYWORD | If, IF, else, ELSE, while, WHILE, return, RETURN |
| ID | i, j, k, abc, ab123, func1 |
| INTEGER | 0, -1, 10, 999, …. |
| LITERAL | “Hello World”, “I am Sejin” |
| OPERATOR | -, +, \*, / |
| COMPARISON | <, > , ==, !=, <=, >= |
| WHITESPACE | , \t, \n |
| BRACE | {, } |
| PAREN | (, ) |
| ASSIGN | = |
| SEMICOLON | ; |
| COMMA | , |

2. Definition of Regular Expressions

STRING  
-> 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, +, -, \*, /, <, >, “, !, =, {, }, (, ), ;, , , \t, \n

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 |
| VARIABLE | Int | INT | char | CHAR |
| KEYWORD | If | IF | else | ELSE | while | WHILE | return | RETURN |
| ID | letter ( letter | digit )\* |
| INTEGER | zero | (( - | ) non-zero ( zero | non-zero )\*) |
| LITERAL | “( letter | digit | )\*” |
| OPERATOR | - | + | \* | / |
| COMPARISON | < | > | == | != | <= | >= |
| WHITESPACE | ( | \t | \n)\* |
| BRACE | { | } |
| PAREN | ( | ) |
| ASSIGN | = |
| SEMICOLON | ; |
| COMMA | , |

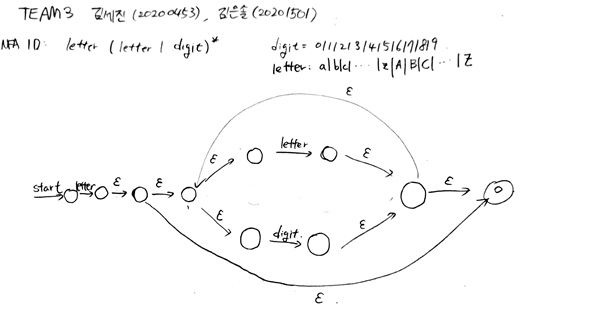
3. NFA (Non-deterministic Finite Automata)

Our team make 3 NFA which is ID, Integer, String. Other part was implemented with the code without using NFA and DFA.

3.1 ID

Regular Expression:

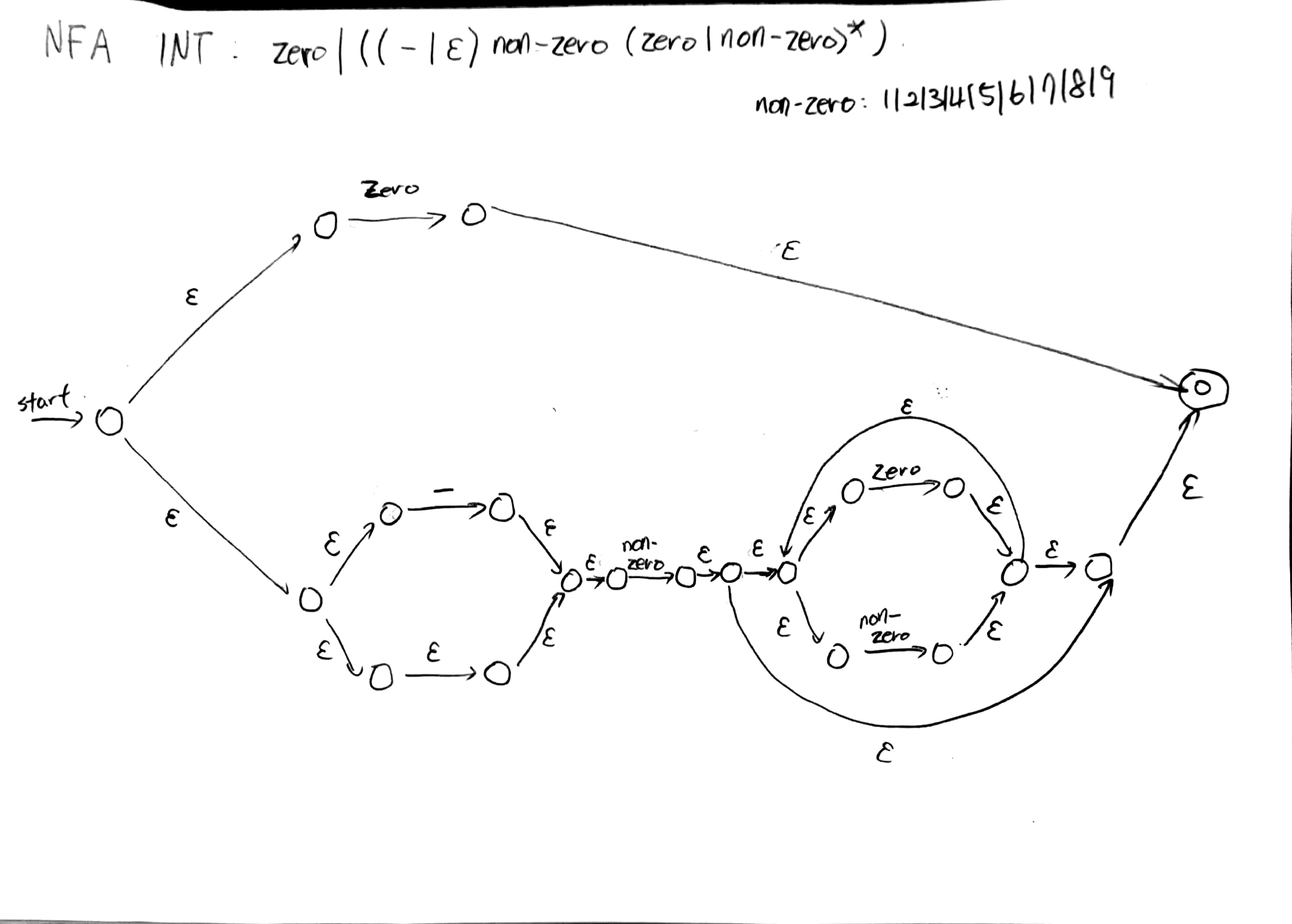
NFA:



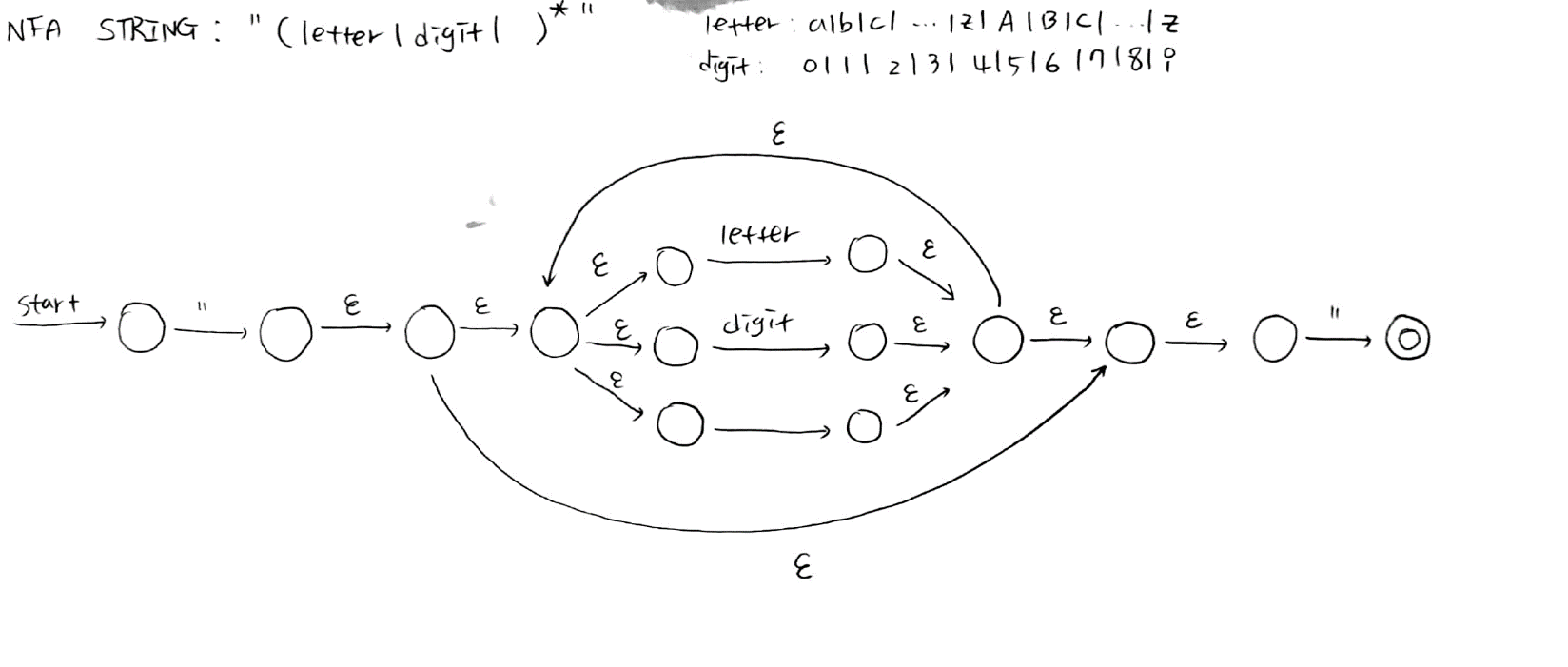
3.2 INTEGER

Regular Expression:

NFA:

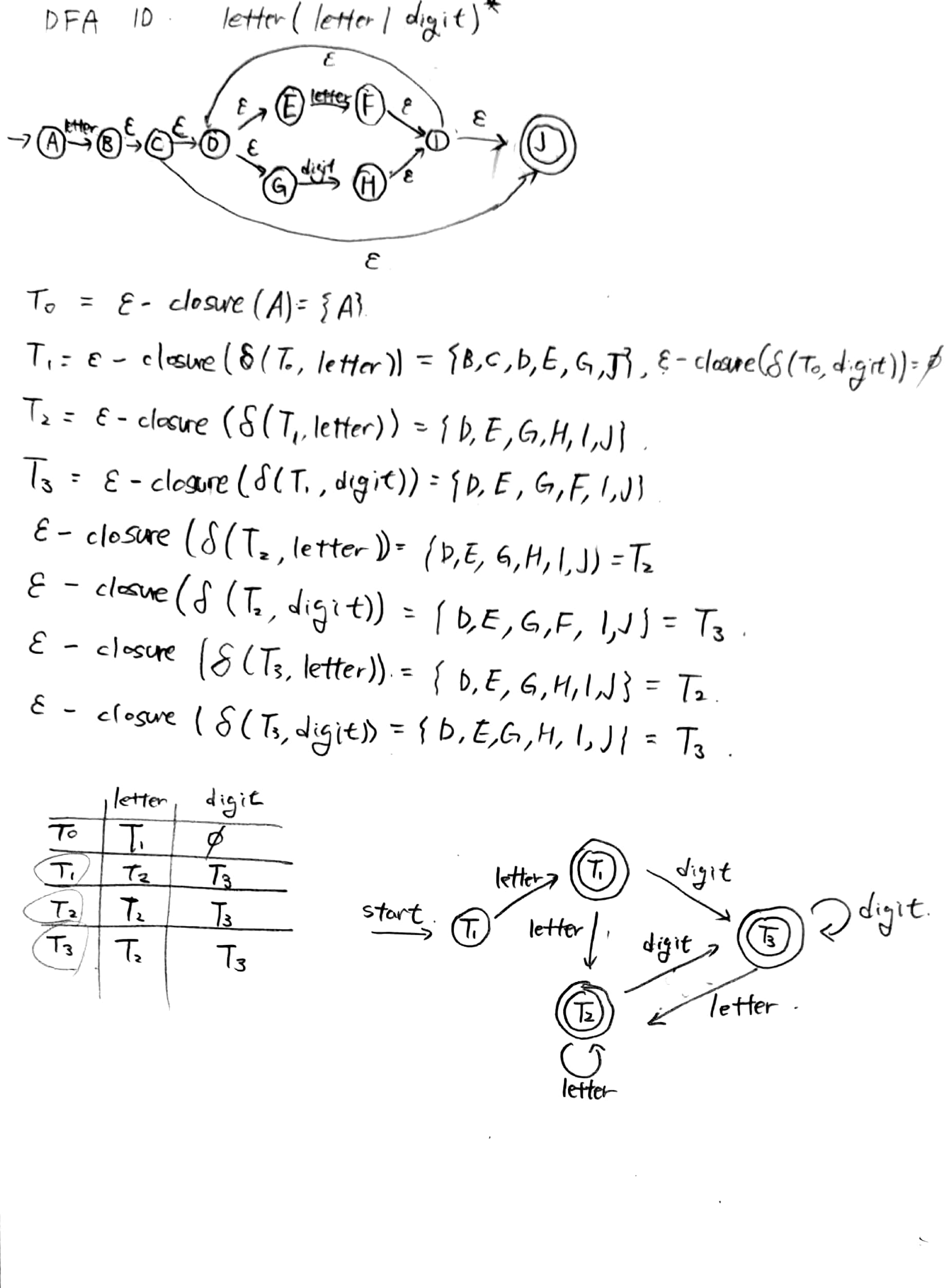


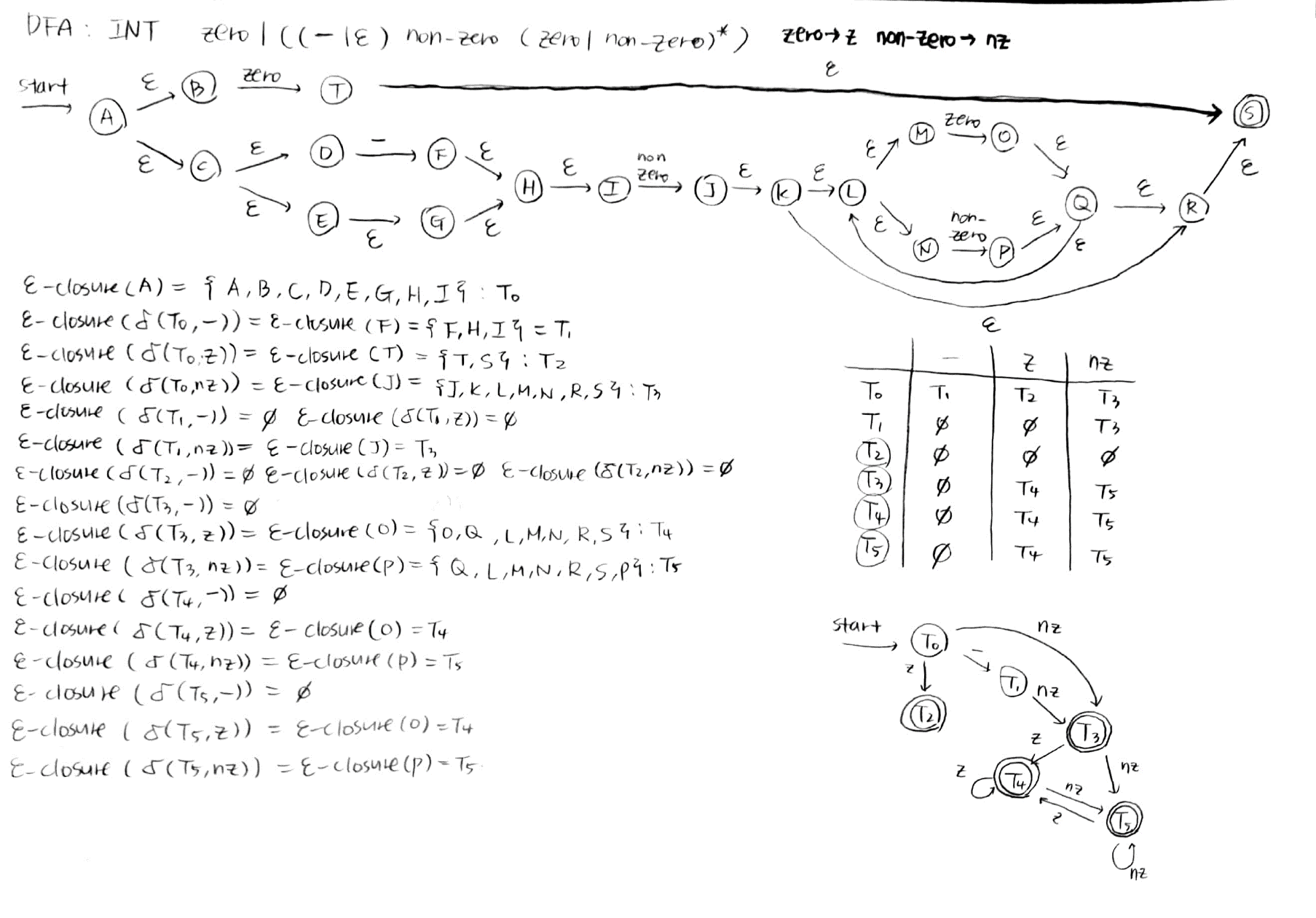
3.3 STRING

Regular Expression:

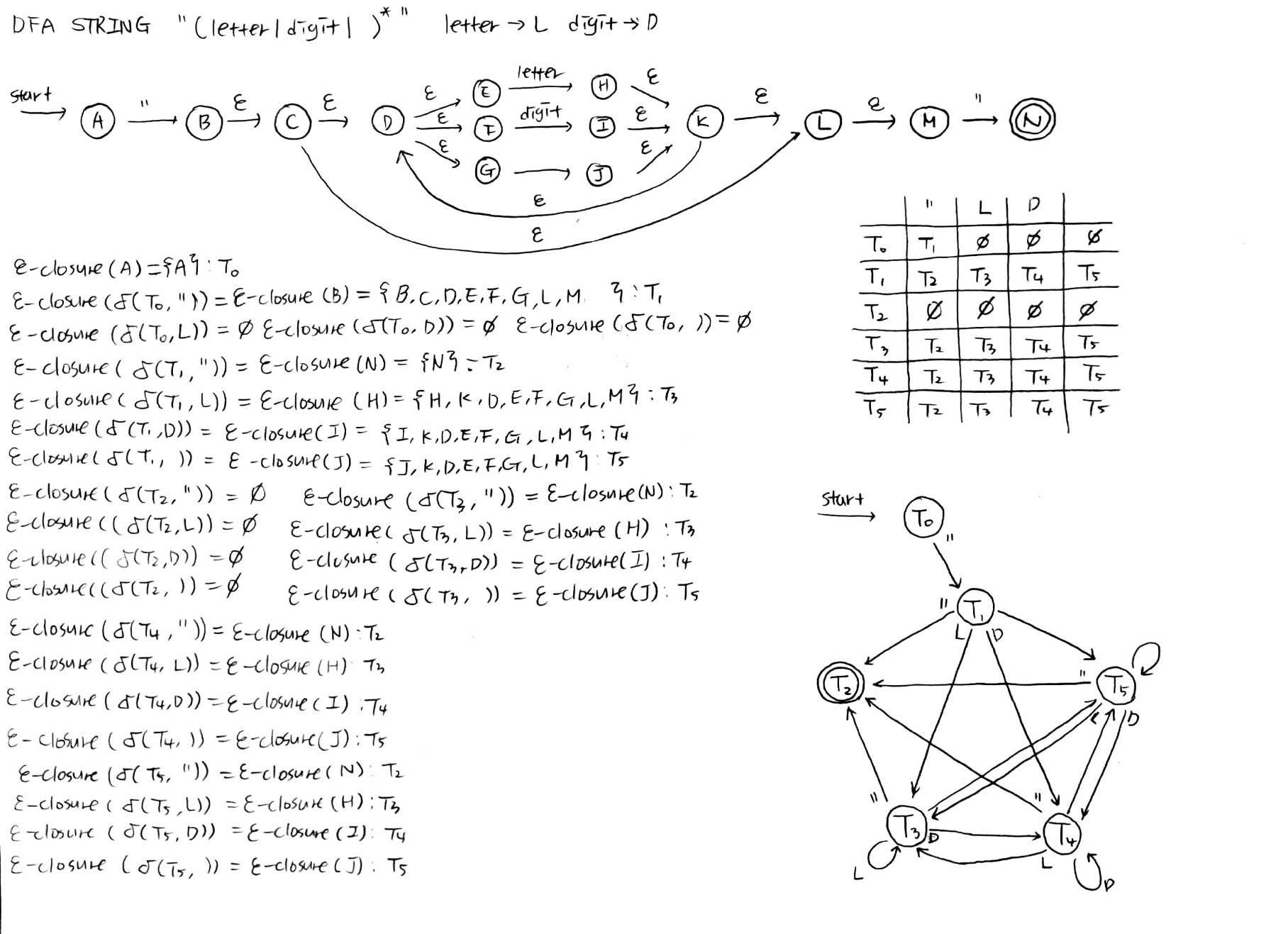
4. DFA (Deterministic Finite Automata)

4.1 ID



4.2 INTEGER

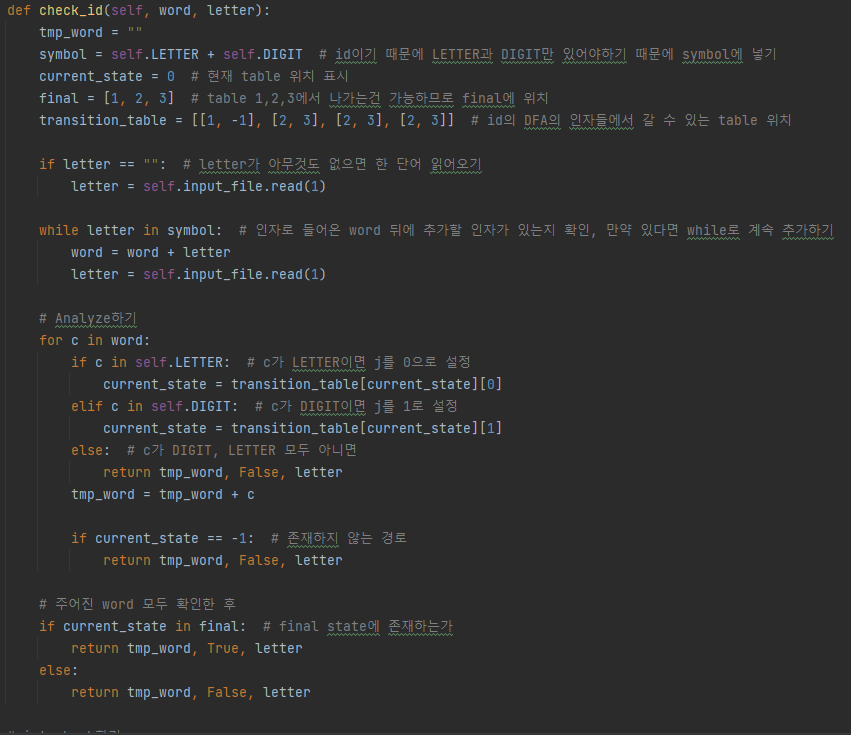
The picture quality may not be good because it’s drawn horizontally. If you want to see it in good quality, please refer to NFA&DFA(final).pdf.

4.3 STRING

The picture quality may not be good because it’s drawn horizontally. If you want to see it in good quality, please refer to NFA&DFA(final).pdf.

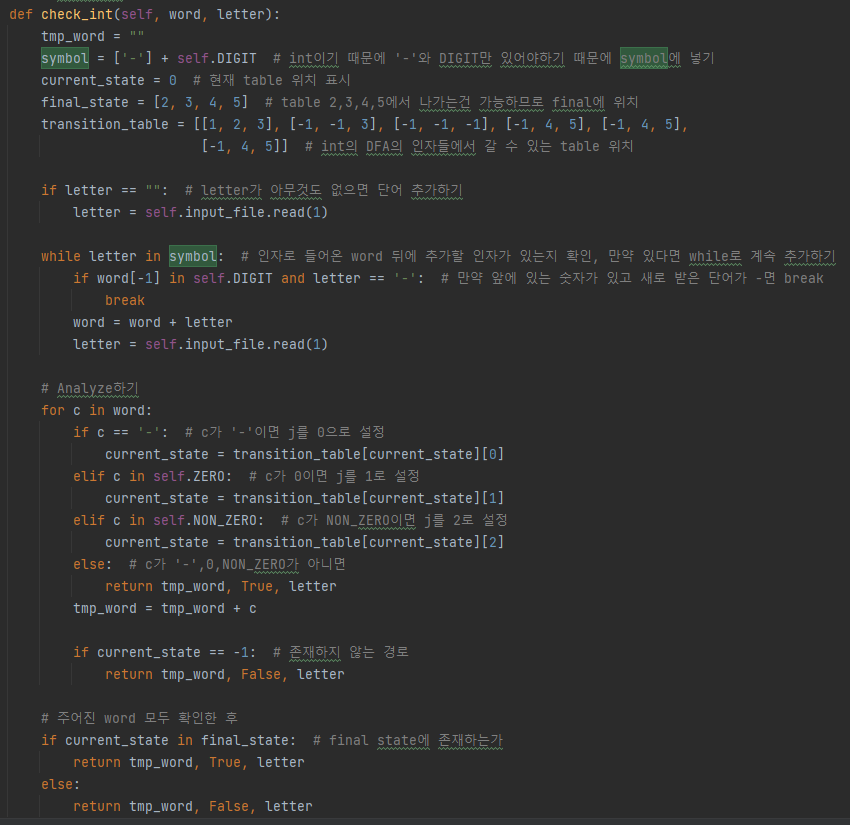
4. DFA (Deterministic Finite Automata)

4.1 ID



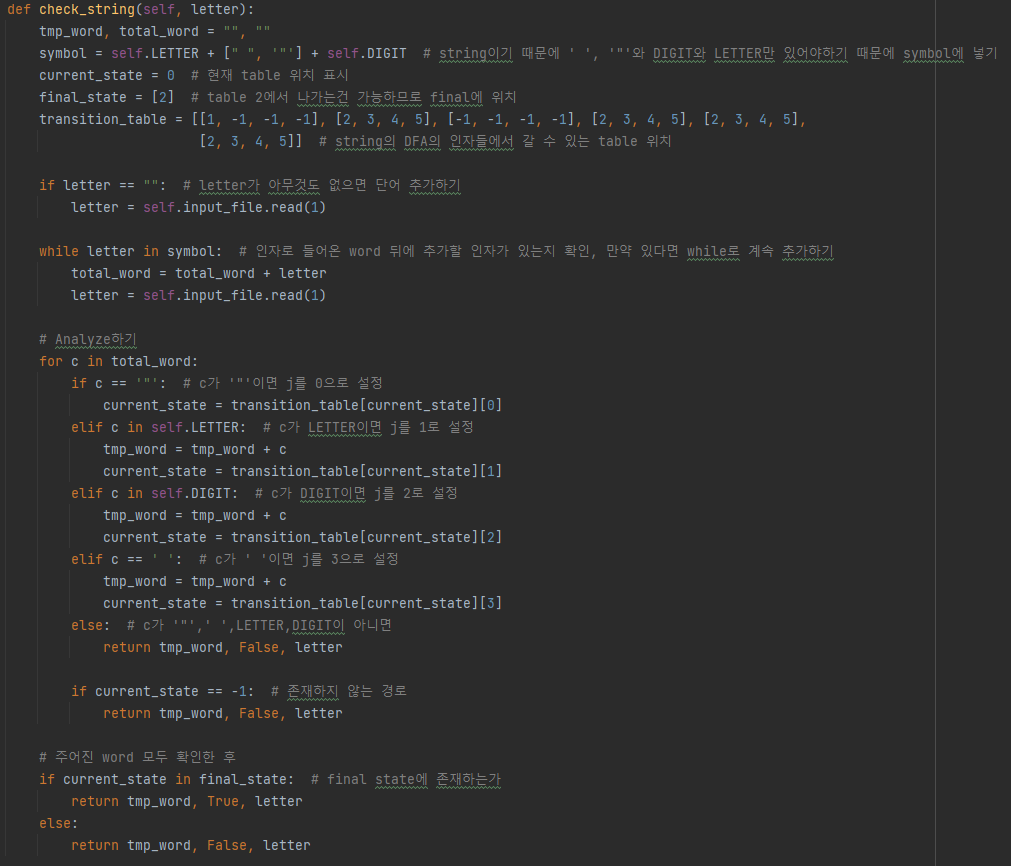
This code is for ID DFA which determine whether ‘word’ is id or not. There are 4 states and start state is T0 and final state is T1, T2, T3. State’s movement is the same as the DFA of the ID DFA above. This method starts with checking ‘word’ has all component of ID. If there is the letter in a DIGIT or LETTER right behind the ‘word’, then add the letter. If there are nothing to add right behind letter, then check whether each letter of ‘word’ is LETTER or DIGIT or neither. If it is LETTER, current\_state move to transitition\_table[current\_state][0]. If it is DIGIT, current\_state move to transitition\_table[current\_state][1]. If neither, then return False with tmp\_word and letter. If current\_state is -1, then return False. When all the words in the word is LETTER or DIGIT, then check current\_state is in T1, T2, T3. If current\_state is in T1, T2, T3, return True with tmp\_word. If it is not return False.

4.2 INTEGER



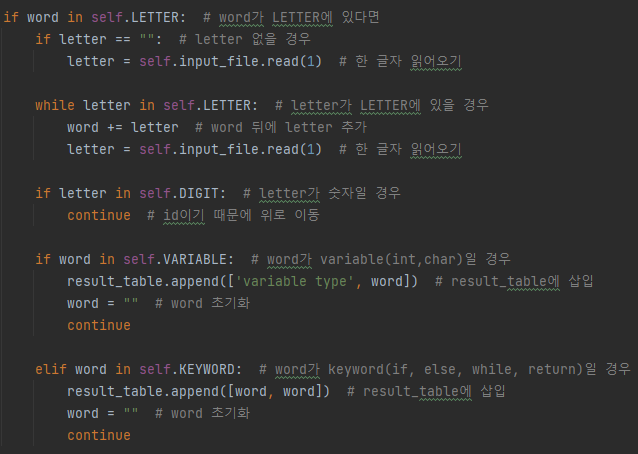
This code is for INT DFA which determine whether ‘word’ is integer or not. There are 6 states and start state is T0 and final state is T2, T3, T4, T5. State’s movement is the same as the DFA of the INT DFA above. This method starts with checking ‘word’ has all component of INT. If there is the letter in a DIGIT or ‘-‘ right behind the ‘word’, then add the letter. But if there right behind letter is ‘-‘ and word’s last letter is digit, then don’ t add ‘-‘ to ‘word’. If there are nothing to add right behind letter, then check whether each letter of ‘word’ is ‘-‘or DIGIT or neither. If it is ’-‘, current\_state move to transitition\_table[current\_state][0]. If it is ZERO, current\_state move to transitition\_table[current\_sta te][1]. If it is NON-ZERO, current\_state move to transitition\_table[current\_state][2]. If it is neither, then return False with tmp\_word and letter. If current\_state is -1, then return False. When all the words in the word is ‘-‘or DIGIT, then check current\_state is in T2, T3, T4, T5,. If current\_state is in T2, T3, T4, T5, return True with tmp\_word. If it is not return False.

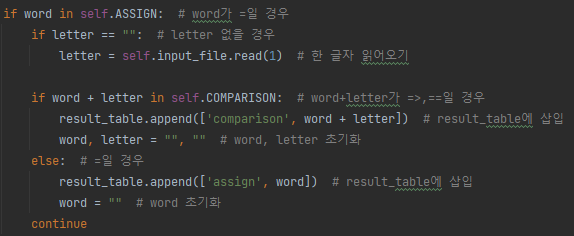
4.3 STRING

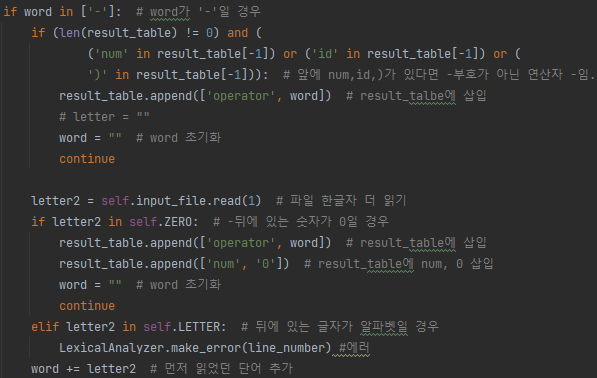
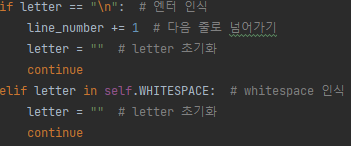


This code is for STRING DFA which determine whether ‘word’ is string or not. There are 5 states and start state is T0 and final state is T2. State’s movement is the same as the DFA of the STRING DFA above. This method starts with checking ‘word’ has all component of STRING. If there is the letter in a DIGIT or ‘”‘ or LETTER right behind the ‘word’, then add the letter. If there are nothing to add right behind letter, then check whether each letter of ‘word’ is ‘”‘or DIGIT or LETTER or neither. If it is ’”‘, current\_state move to transitition\_table[current\_state][0]. If it is LETTER, current\_state move to transitition\_table[current\_state][1]. If it is DIGIT, current\_state move to transitition\_table[current\_st ate][2]. If it is ‘ ‘(whitespace), current\_state move to transitition\_table[current\_state][3]. If it is ’”’, current\_state move to transitition\_table[current\_state][4]. If it is neither, then return False with tmp\_word and letter. If current\_state is -1, then return False. When all the words in the word is ‘”‘or DIGIT or LETTER or ‘ ‘, then check current\_state is in T2,. If current\_state is in T2, return True with tmp\_word. If it is not return False.

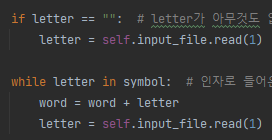
4.4 OTHER TOKENS

- Variable, Keyword -BRACE, PAREN, COMMA, OPERATOR, COMPASRION

- ASSIGN, COMPARSION

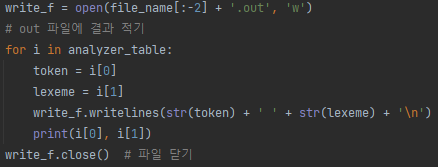
- SUBTRACT - WHITESPACE

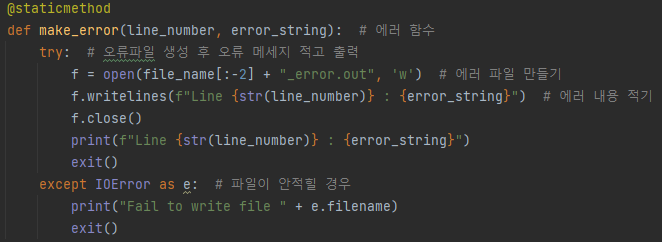
4.5 FILE I/O

- READ FILE -READ ONE LETTER





-WRITE FILE

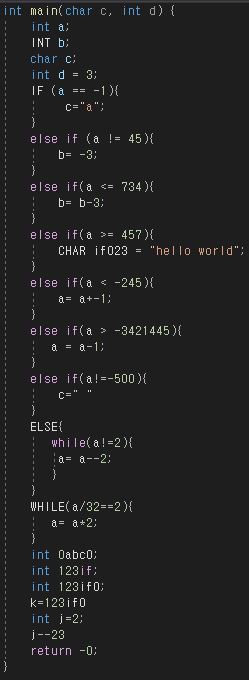
-ERROR CODE

If there are error in the input file, then make FILE\_NAME\_error.out. It has 6 types of error which is “Unsupported characters exist”, “Unacceptable format”, “Invalid COMPARSION COMBINATION”, “Invalid format for INT”, “Invalid format for ID”, “Invalid format for STRING.” The error\_string depends on the factors function received. Also, it shows which line error occurred.

4.6 TEST

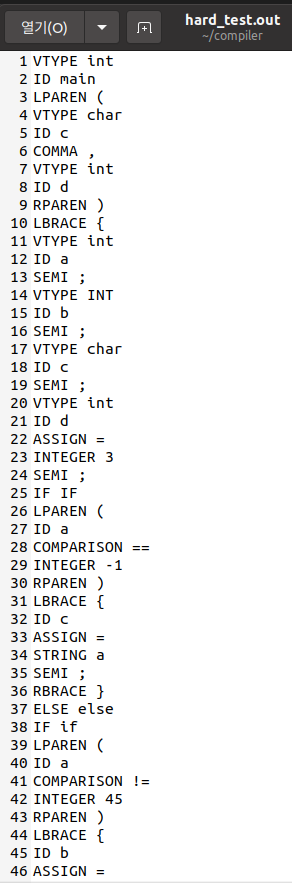
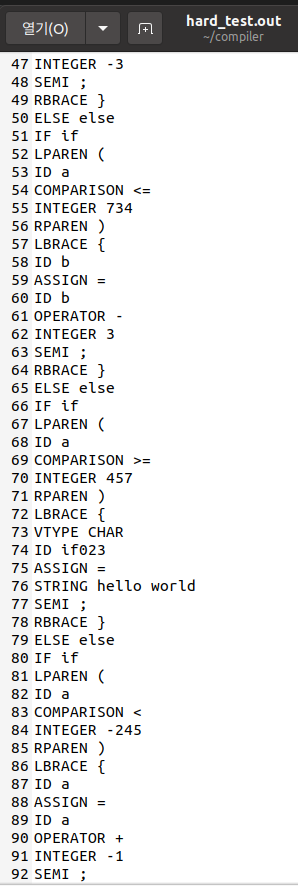
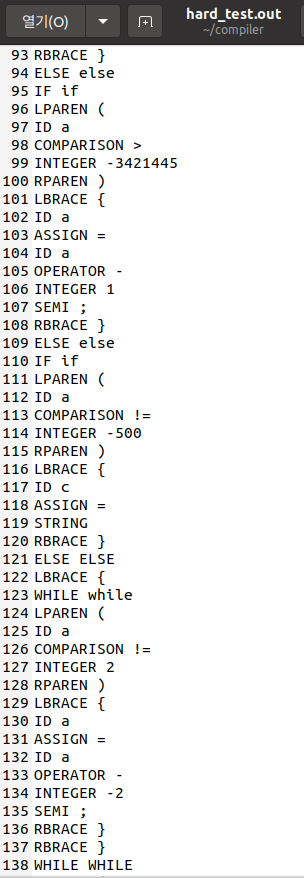
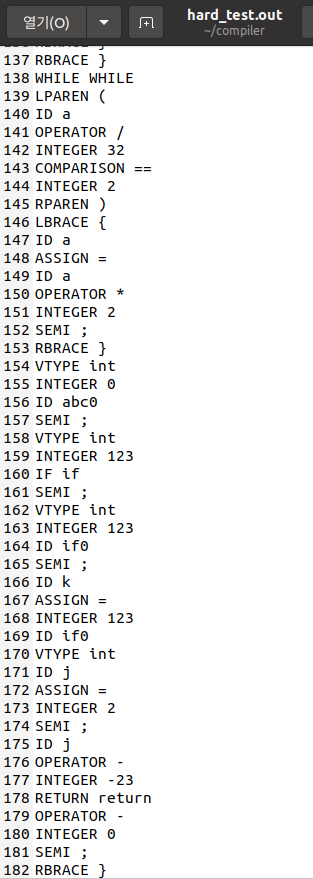
4.6.1 INPUT FILE

We do hard\_test.c file.

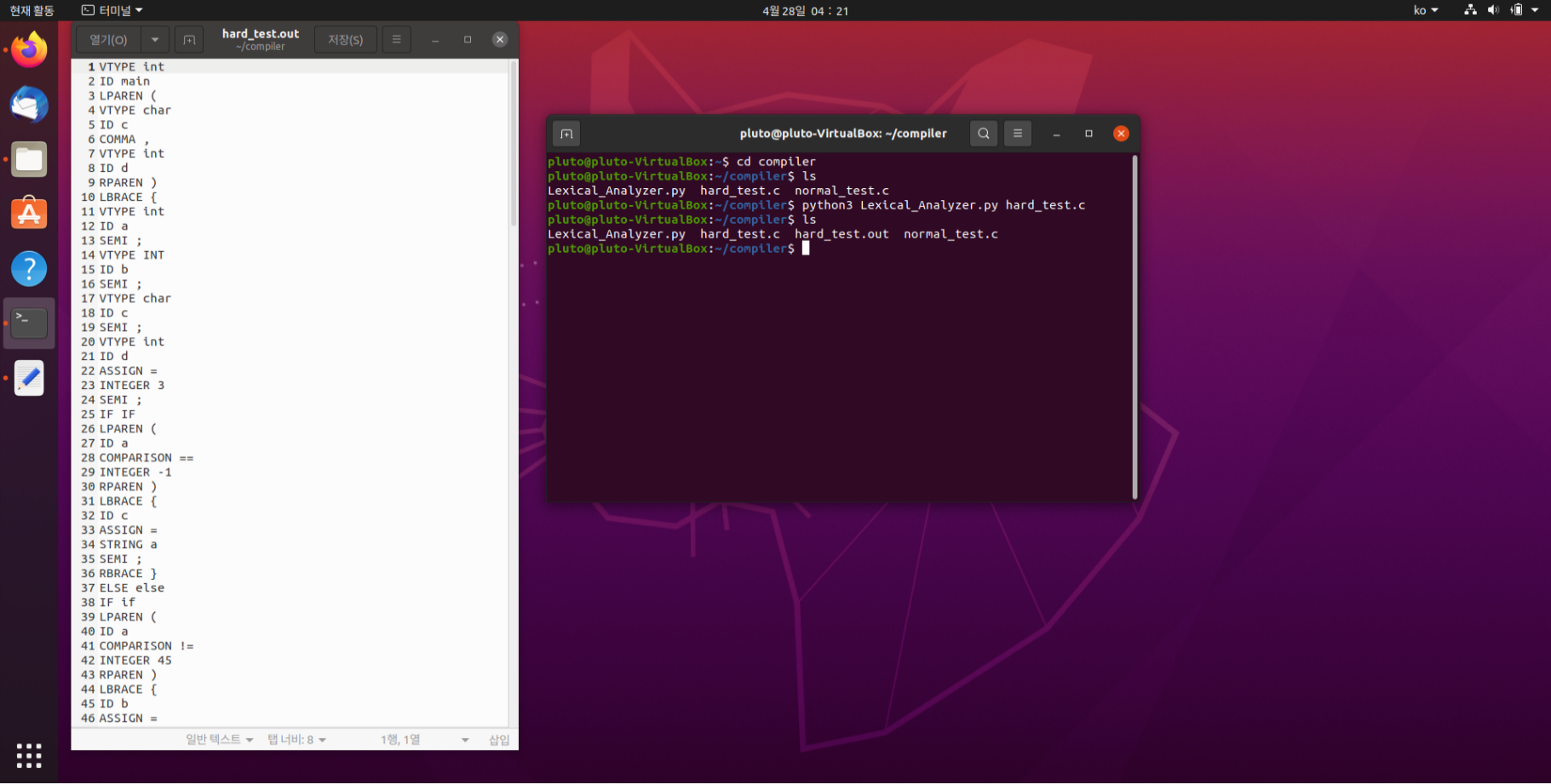


We made an input file to check as many cases as possible and conducted the test.

How to execute code: python3 Lexcial\_Analyzer.py hard\_test.c

4.6.2 OUTPUT FILE

All the cases in the input file were properly output. Through this, we found that our Lexical Analyzer operates according to the assignment conditions.

4.6.3 RUNNING ON LINUX

Our code runs perfectly on Linux.

The picture quality may not be good because it’s captured horizontally. If you want to see it in good quality, please refer to linux\_working.png.