**Compiler**

**Term Project 1 Report**

컴파일러 01분반

소프트웨어학부

20202475 이동훈

20206802 임도연

Contents

1. Tokens

2. Regular Expressions

3. NFA & DFA

1) Signed Integer

2) Literal String

3) ID

4) OP

5) Compare

4. Program explanation & implementation

1. Tokens

|  |  |
| --- | --- |
| VARTYPE | int, INT, char, CHAR |
| SIGNED INTEGER | 0, -1, 10, 999, … |
| LITERAL STRING | “Hello World”, “My student id is 12345678”, … |
| ID | i, j, k, abc, ab123, func1, … |
| KEYWORD | if, IF, else, ELSE, while, WHILE, return, RETURN |
| OP | +, -, \*, / |
| ASSIGN | = |
| COMPARE | <, > , ==, !=, <=, >= |
| SEMICOLON | ; |
| LB | { |
| RB | } |
| LPAREN | ( |
| RPAREN | ) |
| COMMA | , |
| WHITESPACE | ‘\t’, ‘\n’, blanks |

2. Regular Expressions

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’]

DIGIT = [‘1’, ‘2’, ‘3’, ‘4’, ‘5’, ‘6’, ‘7’, ‘8’, ‘9’]

ZERO = [‘0’]

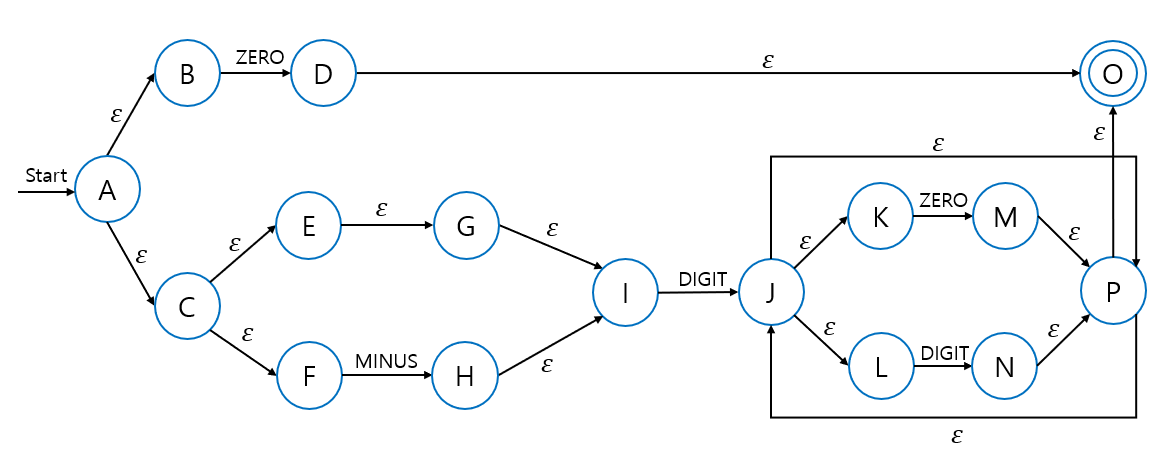
MINUS = [‘-‘]

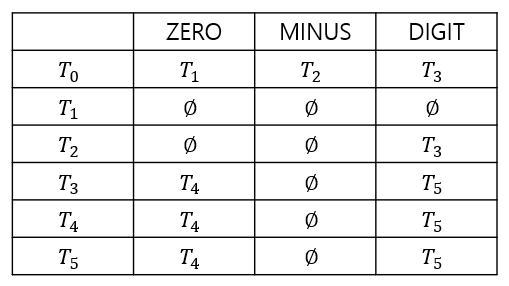
OPERATOR = [‘+’, ‘\*‘, ‘/’]

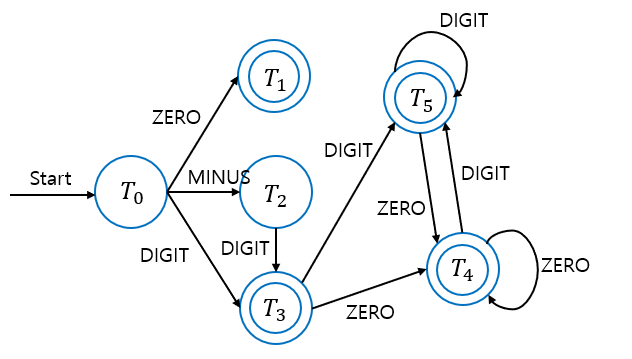
WHITESPACE = [‘\t’, ‘\n’, ‘ ‘]

|  |  |
| --- | --- |
| VARTYPE | int|INT|char|CHAR |
| INTEGER | ZERO|((|MINUS)DIGIT(ZERO|DIGIT)\*) |
| STRING | “(ZERO|DIGIT|LETTER|WHITESPACE)\*” |
| ID | LETTER(LETTER|ZERO|DIGIT)\* |
| KEYWORD | if|IF|else|ELSE|while|WHILE|return|RETURN |
| OP | OPERATOR|MINUS |
| ASSIGN | = |
| COMPARE | <|>|==|!=|<=|>= |
| SEMICOLON | ; |
| LB | { |
| RB | } |
| LPAREN | ( |
| RPAREN | ) |
| COMMA | , |
| WHITESPACE | WHITESPACE |

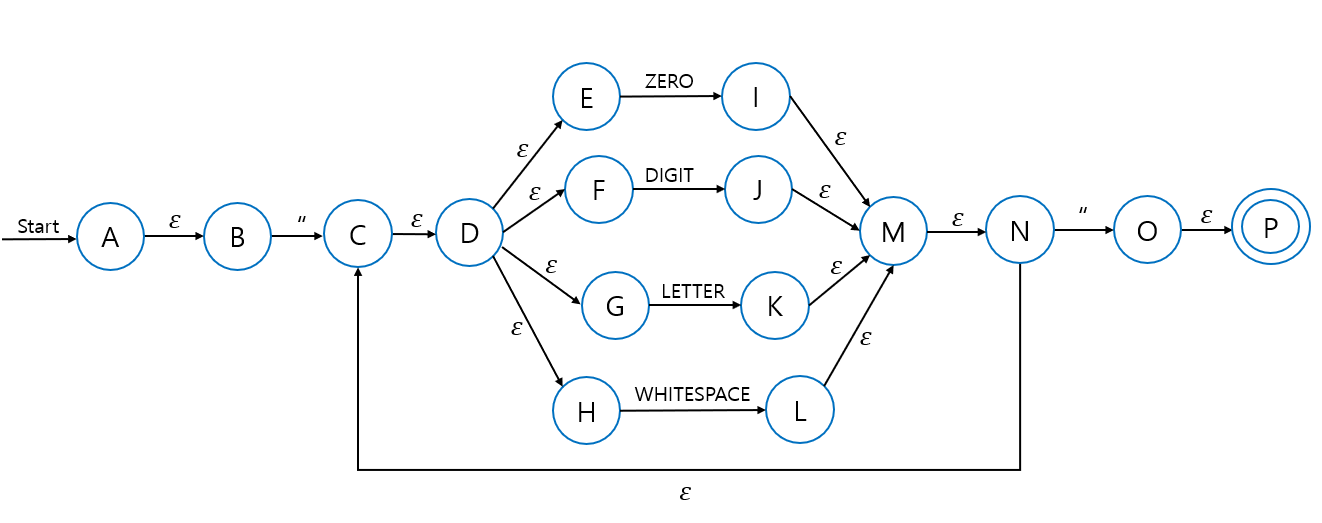
3. NFA & DFA

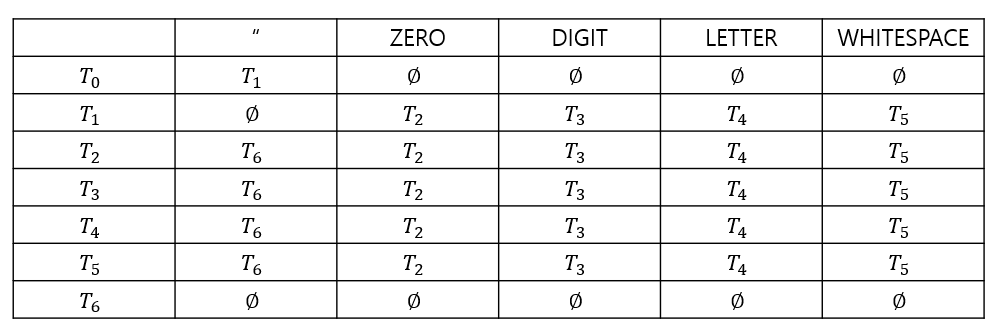
1) Signed Integer

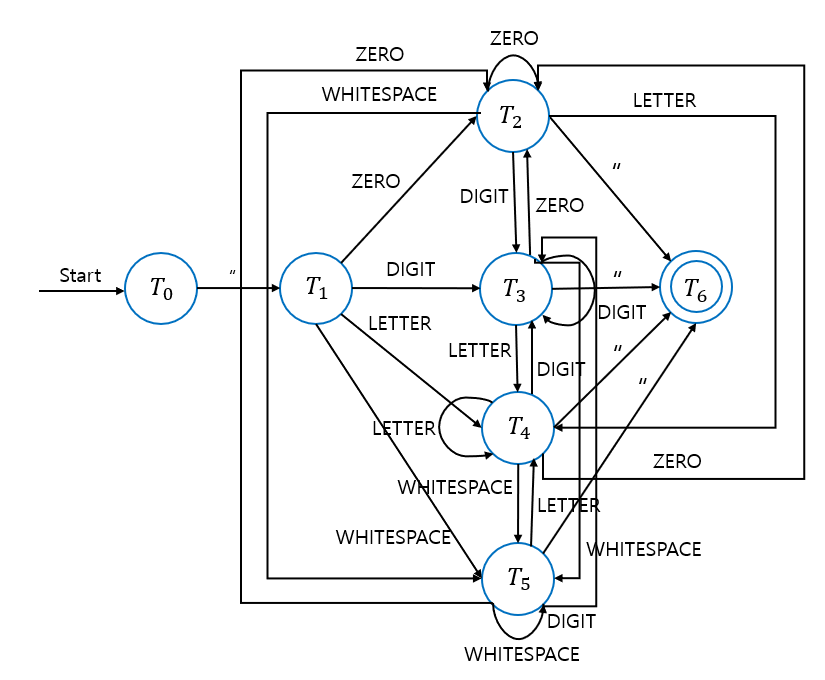




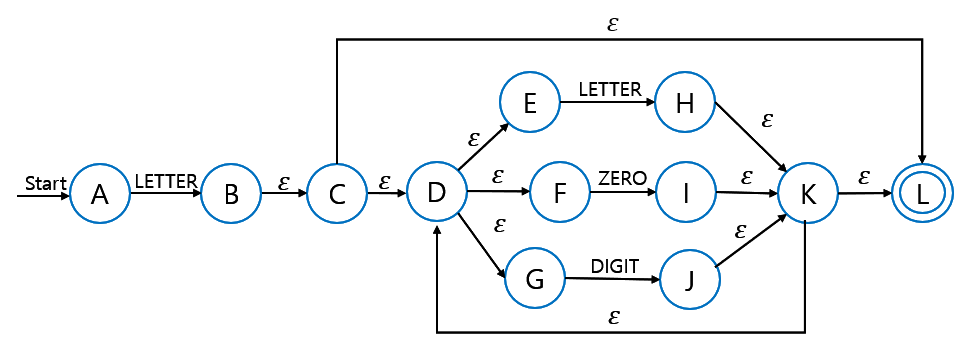
2) Literal String

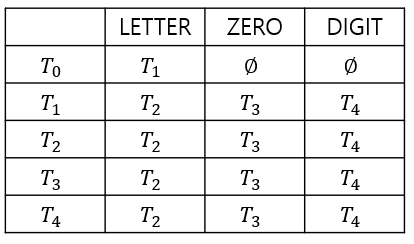


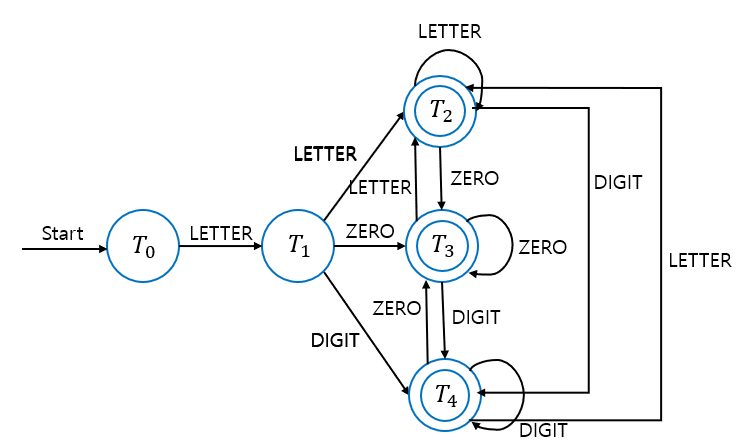




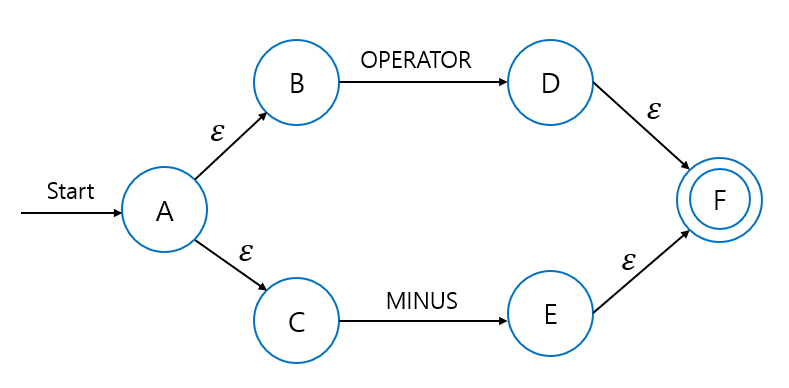
3) ID

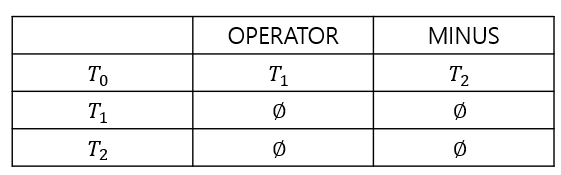


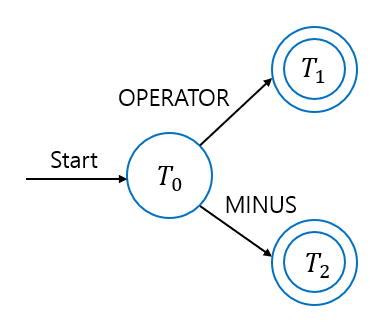




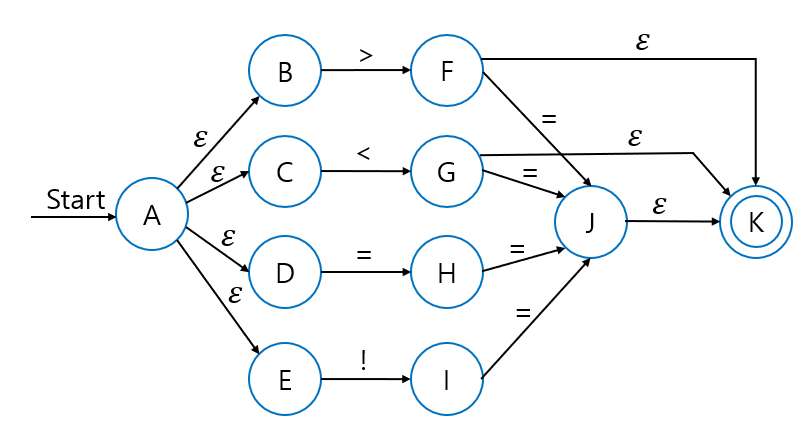
4) OP

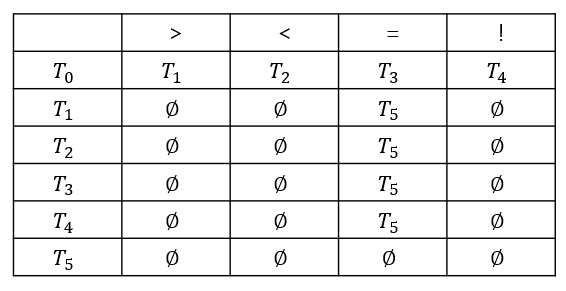


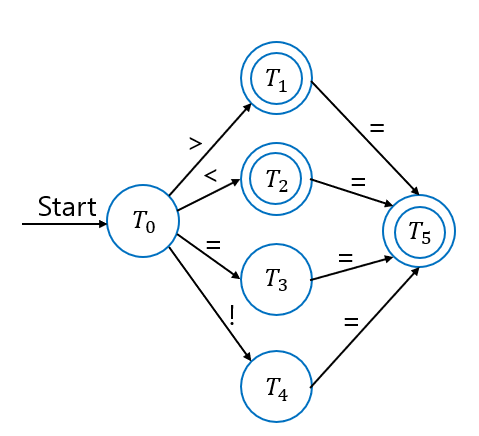




5) Compare



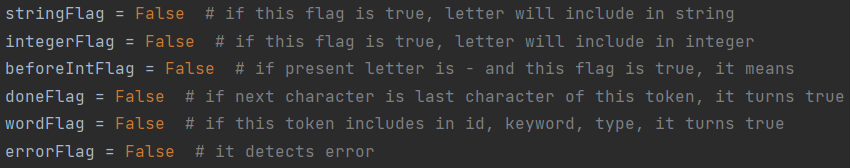




4. Program explanation & implementation

(1) Program explanation

1) Flag Setting



stringFlag – 큰 따옴표 입력을 처음 받으면 Flag를 표시하고, 다음 큰 따옴표가 들어오기 전까지는 모두 string으로 처리해서 token value 로 저장함

integerFlag – 숫자가 들어올 때 SIGNED INTEGER로 저장하기 위해 표시하는 Flag. – ID에 들어가는 숫자와 구분하기 위한 용도.

beforeIntFlag – ‘-‘가 들어올 때 OP로 구분할지 negative Integer로 구분할지 판단하기 위해 사용하는 Flag. 바로 이전 token이 INTEGER인 경우 표시, 아닌경우 표시하지 않음으로 구분.

doneFlag – 두 글자 짜리 비교 연산자를 체크하기 위해 사용하는 Flag. ‘<’, ‘>’, ‘!’가 들어오고 다음 글자가 ‘=’인 경우 Flag를 표시해서 입력받고 바로 tokenize 하도록 설정.

wordFlag – 들어오는 값이 VARTYPE, KEYWORD, ID 중 하나임을 표시하는 Flag.

errorFlag – 에러를 탐지하기 위한 용도

2) Define Token

텍스트이(가) 표시된 사진

자동 생성된 설명

후에 글자를 인식하기 위해 미리 구분을 해 놓았다. LETTER는 알파벳 대소문자, DIGIT은 1~9 등 본 analyzer에서 인식해야 할 글자들을 미리 구분해 놓고 사용했다.

3) Variable setting & input setting

텍스트이(가) 표시된 사진

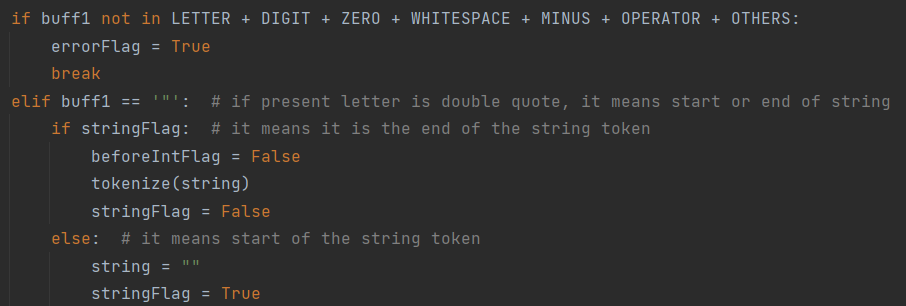
자동 생성된 설명

output 리스트는 토큰화한 값들을 저장하는 리스트이고, string 변수는 토큰화할 value값을 저장할 문자열이다.

test.c라는 파일에서 C언어 코드를 받아와서 한 글자씩 buff1에 입력을 받고 글자를 판단한다.

buff2를 통해 다음 글자를 미리 입력받는 데, 이는 현재 글자에서 해당 토큰이 끝나는지 판단하기 위해서이다.

4) Double-quotation Mark



buff1이 LETTER, DIGIT, ZERO, WHITESPACE, MINUS, OPERATOR, OTHERS이 아니라면 errorFlag를 True로 하고 나온다.

큰 따옴표가 입력되는 경우, STRING token을 시작하거나 끝을 내는 경우로 판단할 수 있다.

이때, stringFlag가 True라면 이미 입력 받고 있던 상황이므로 끝임을, False라면 입력을 시작하는 상황이므로 시작임을 알 수 있다. 이때, 끝낼 경우에는 tokenize 함수를 호출해 tokenize한 후 output list에 저장한다.

이후 Flag 설정을 하게 되는데, 현재 토큰이 String 이므로 beforeIntFlag를 False, string token이 종료되었으므로 stringFlag를 False 로 변경한다.

5) stringFlag & doneFlag

텍스트이(가) 표시된 사진

자동 생성된 설명

buff1이 큰 따옴표가 아닌데, stringFlag가 True인 경우 들어오는 글자는 모두 STRING token value로 들어가야 하므로 string에 더해준다.

doneFlag가 True인 경우는, 뒤에서 설명하겠지만 이미 앞에서 ‘<’, ‘>’, ‘!’가 들어오고 뒤 글자가 ‘=’임을 확인한 이후이다. 그러므로 string에 buff1을 더해주고 tokenize 함수를 호출한다.

beforeIntFlag는 위와 같은 이유로 False, doneFlag는 해당 비교연산자 TOKEN이 종료되었으므로 False로 설정한다.

6) LETTER input

텍스트이(가) 표시된 사진

자동 생성된 설명

buff1에 알파벳 대소문자가 들어온 경우, 2가지 경우로 구분된다. type, keyword, id 토큰의 시작이거나 중간이거나. 그렇기 때문에 Flag가 False이면 Flag를 바꿔주고, 그게 아니라면 그냥 string에 더해주기만 한다. 만약 buff2(다음 문자)가 type, keyword, id 토큰의 구성요소가 아니라면, 지금까지 저장한 string으로 tokenize를 진행한다. 이후 Flag를 초기화해준다.

7) ZERO & DIGIT input

텍스트이(가) 표시된 사진

자동 생성된 설명

숫자가 들어온 경우는 4가지 경우로 구분된다. ID TOKEN의 숫자, INTEGER TOKEN의 숫자(0, 중간, 시작).

ID의 경우 첫 시작이 숫자로 시작할 수 없기 때문에 무조건 wordFlag가 True인 상태에서 들어오므로, 위와 같이 처리를 해준다. 만약 다음 글자가 ID에 포함되지 않는 글자라면, beforeIntFlag를 True로 해, 변수가 왔음을 표시한다.

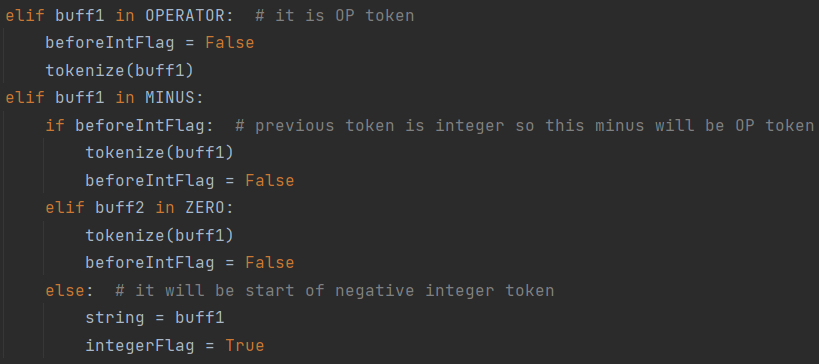
integerFlag가 True인 경우, 이미 숫자로 판단하고 있기 때문에 string에 더해준다.

Flag가 켜져 있지 않은데 0이 들어온 경우, INTEGER 0이므로 그 자체로 바로 tokenize를 진행한다.

1~9의 숫자가 들어온 경우, INTEGER의 시작이므로 Flag를 True로 바꿔주고 저장을 시작한다.

0을 제외한 나머지 세 경우 buff1이 TOKEN value의 마지막 글자일 수 있으므로 buff2를 확인하고 tokenize 여부를 결정한다.

8) Operator & MINUS input



OPERATOR(+, \*, /)의 경우, 그 자체로 TOKEN Value이므로 바로 tokenize를 진행한다.

‘-‘의 경우, OP일수도, 음수일수도 있기 때문에 확인을 한다. beforeIntFlag가 True인 경우, 앞 토큰이 숫자였기 때문에 OP로 사용됨을 알 수 있다. 그렇지 않으면 integer로 인식하고 설정하게 된다. 만약 다음 토큰이 0인 경우, 숫자의 시작이 아니라 OP 이므로 따로 처리를 해준다.

9) Compare Operation & Other characters input

텍스트이(가) 표시된 사진

자동 생성된 설명

OTHERS(소괄호, 중괄호, 쉼표, 세미콜론)의 경우, 한 글자로 TOKEN Value이기 때문에 바로 tokenize를 진행한다.

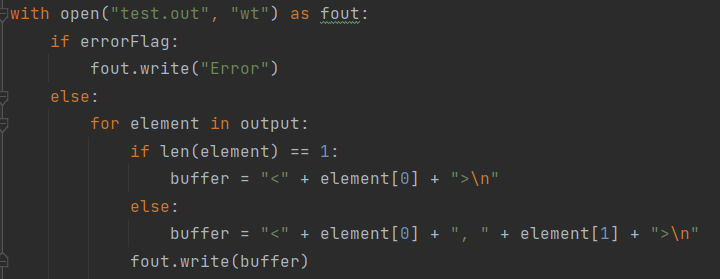
Compare Operation의 경우, 1글자, 2글자로 구분되는데, buff1에 ‘<’, ‘>’, ‘!’, ‘=’이 입력되는 경우, string에 우선 저장을 하게 된다. 이후 buff2에 ‘=’이 입력되는 경우, doneFlag를 True로 바꾸게 된다. 그렇지 않으면 현재 글자 하나만으로 tokenize를 진행한다.

10) next character input



buff2의 값을 buff1에 옮기게 되면, 다음 글자를 현재 글자로 바꾸게 된다는 의미이다. buff2는 새로 입력 받게 된다.

11) Output



test.out 파일에 토큰화한 결과를 출력하게 된다.

만약 errorFlag가 True라면 “Error”를 출력한다.

element 길이가 1인 경우, 즉 토큰에 value가 없는 경우, token만 출력하게 되고, 그렇지 않으면 token과 함께 value를 출력하게 된다.

(2) Implementation

1)

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명 <input> <output>

2)

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명 <input> <output>

3)

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트, 안테나이(가) 표시된 사진

자동 생성된 설명 <input> <output>

4)

<input> <output>

텍스트이(가) 표시된 사진

자동 생성된 설명