**Compiler**

**Term Project 2 Report**

컴파일러 01분반

소프트웨어학부

20202475 이동훈

20206802 임도연

Contents

1. CFG

2. SLR parsing table

3. lexicla.py code 수정

4. Program explanation & implementation

1. CFG

**00: S → CODE**

01: CODE → VDECL CODE | FDECL CODE | ϵ

02: VDECL → vtype id semi

03: FDECL → vtype id lparen ARG rparen lbrace BLOCK RETURN rbrace

04: ARG → vtype id MOREARGS | ϵ

05: MOREARGS → comma vtype id MOREARGS | ϵ

06: BLOCK → STMT BLOCK | ϵ

07: STMT → VDECL | id assign RHS semi

08: STMT → if lparen COND rparen lbrace BLOCK rbrace else lbrace BLOCK rbrace

09: STMT → while lparen COND rparen lbrace BLOCK rbrace

10: RHS → EXPR | literal

11: EXPR → TERM addsub EXPR | TERM

12: TERM → FACTOR multdiv TERM | FACTOR

13: FACTOR → lparen EXPR rparen | id | num

14: COND → FACTOR comp FACTOR

15: RETURN → return FACTOR semi

**CFG 수정된 부분**

- CFG의 Start를 ‘S → CODE’로 수정

2. SLR parsing table

S → CODE

CODE → VDECL CODE

CODE → FDECL CODE

CODE → ϵ

VDECL → vtype id semi

FDECL → vtype id lparen ARG rparen lbrace BLOCK RETURN rbrace

ARG → vtype id MOREARGS

ARG → ϵ

MOREARGS → comma vtype id MOREARGS

MOREARGS → ϵ

BLOCK → STMT BLOCK

BLOCK → ϵ

STMT → VDECL

STMT → id assign RHS semi

STMT → if lparen COND rparen lbrace BLOCK rbrace else lbrace BLOCK rbrace

STMT → while lparen COND rparen lbrace BLOCK rbrace

RHS → EXPR

RHS → literal

EXPR → TERM addsub EXPR

EXPR → TERM

TERM → FACTOR multdiv TERM

TERM → FACTOR

FACTOR → lparen EXPR rparen

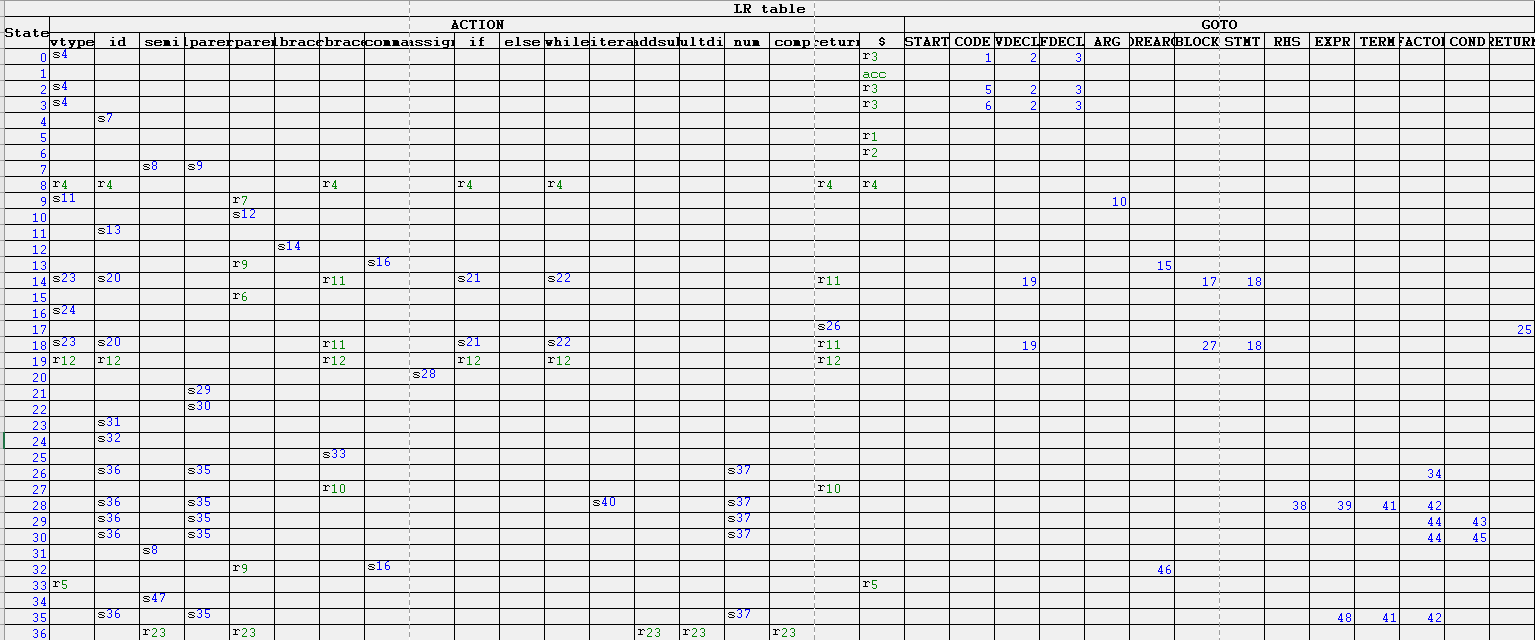
FACTOR → id

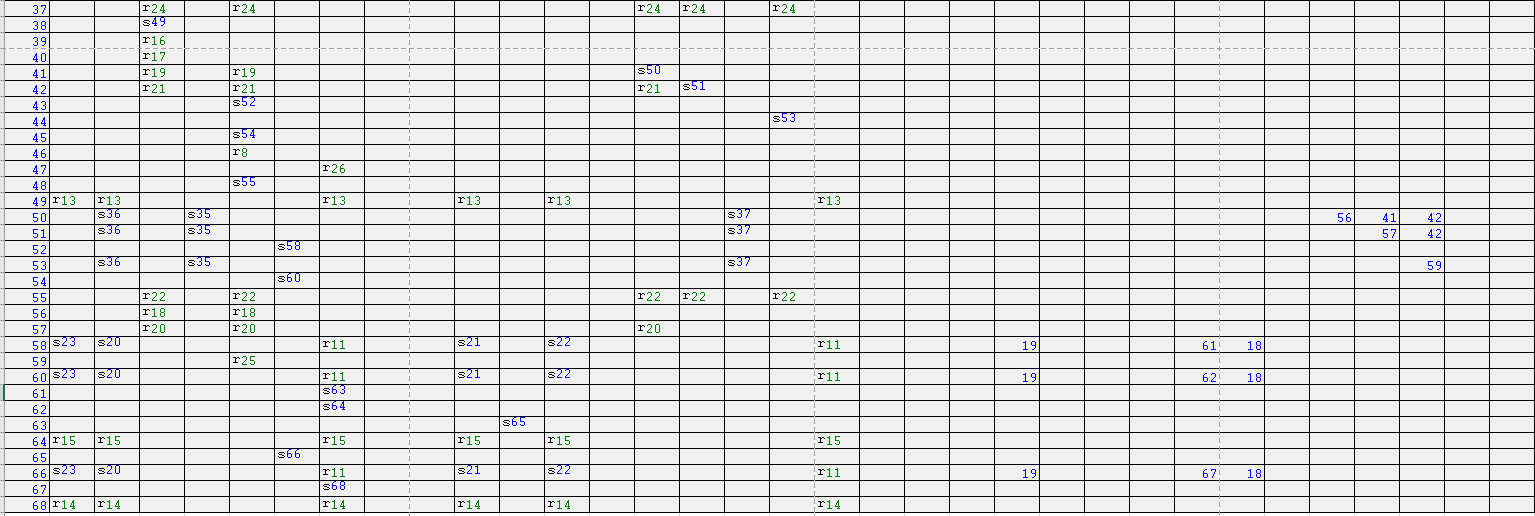
FACTOR → num

COND → FACTOR comp FACTOR

RETURN → return FACTOR semi

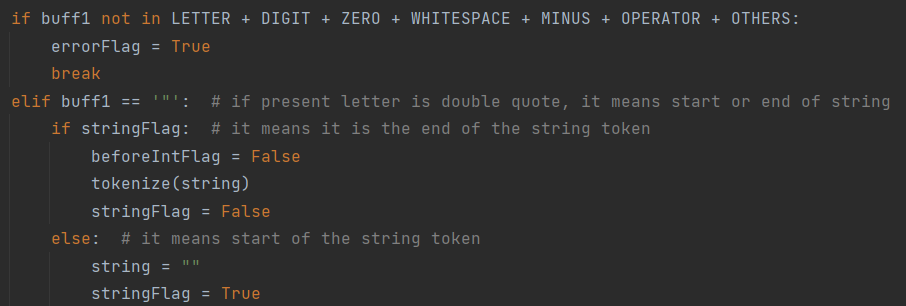
**<SLR Parsing Table>**



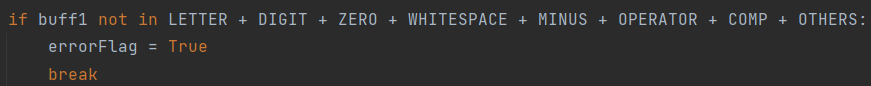


3. lexical.py code 수정

**원래 lexical.py 코드**



**수정된 lexical.py 코드**

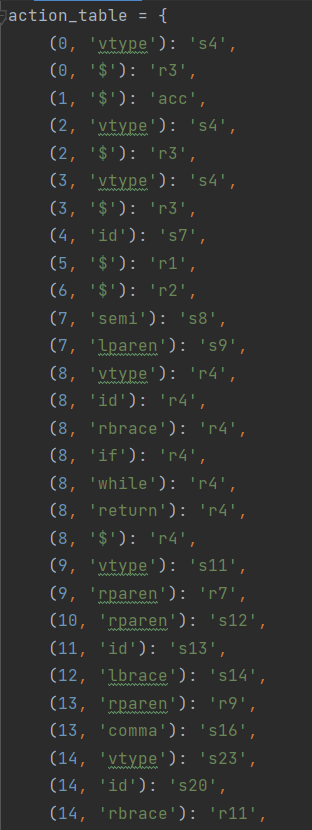


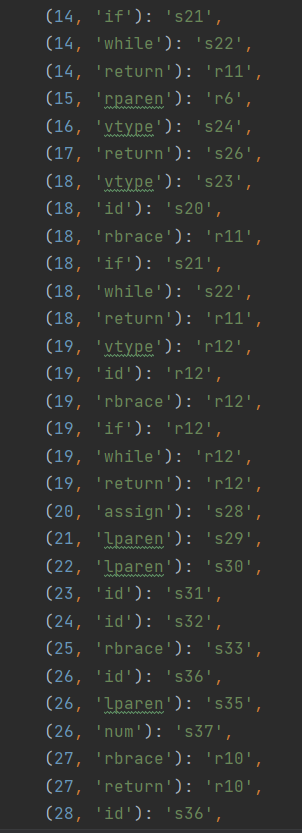
원래 lexical.py 코드에서 error를 탐지하는 코드에서 COMP(‘<’, ‘>’, ‘=’, ‘!’)를 범위에 넣지 않아서 이를 error라고 인식하는 결과가 도출되었다. 그래서 수정된 코드에서는 COMP를 추가해주어 이는 error라고 인식하지 못하게 했다.

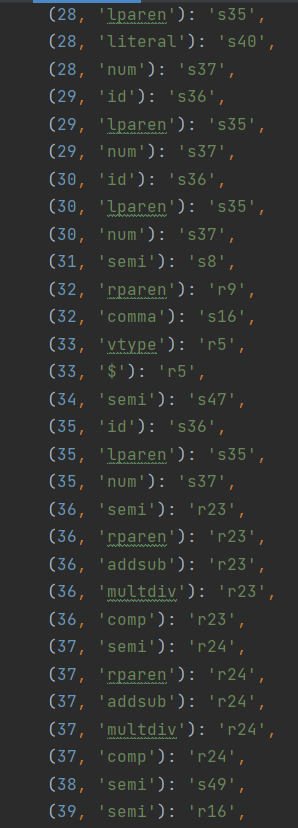
4. Program explanation & implementation

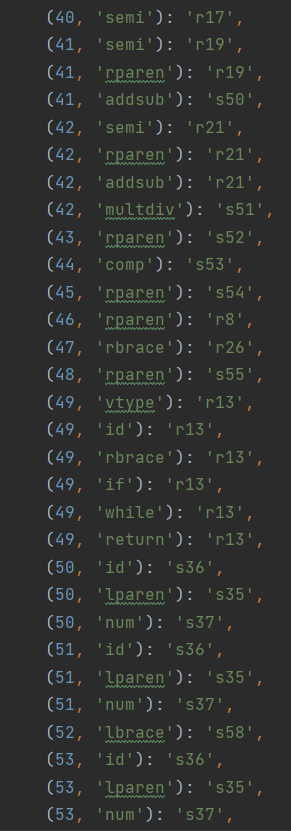
(1) Program Explanation

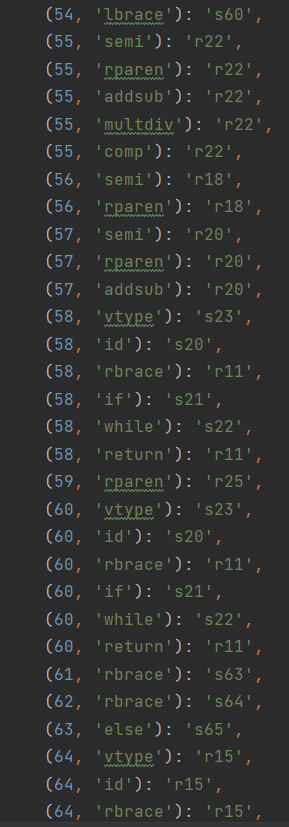
**1) SLR parsing table의 Action table 정의**

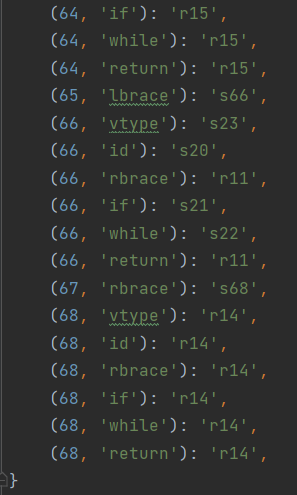




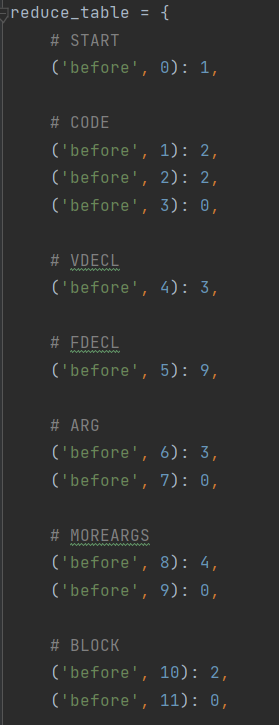


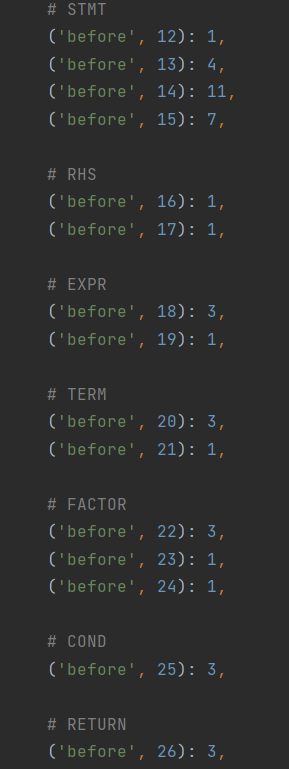


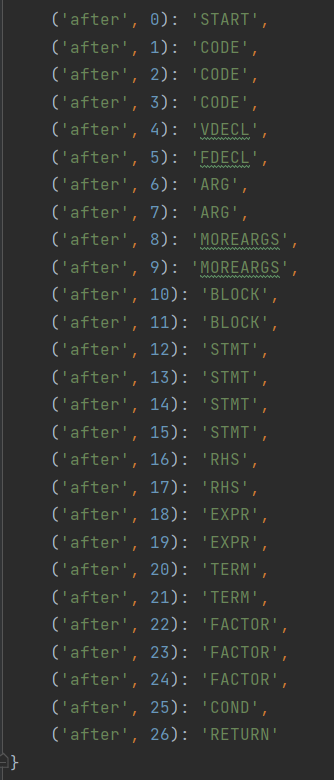




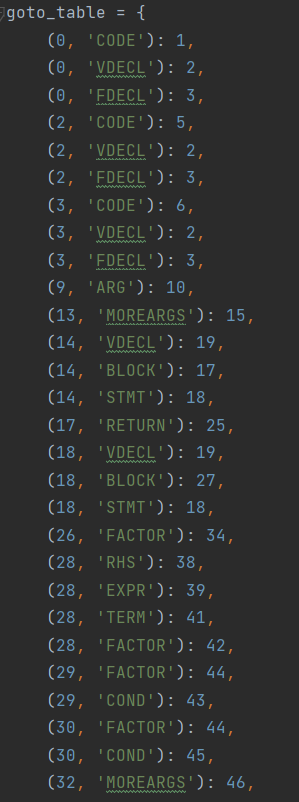
**2) Action table 중 reduction table 정의**

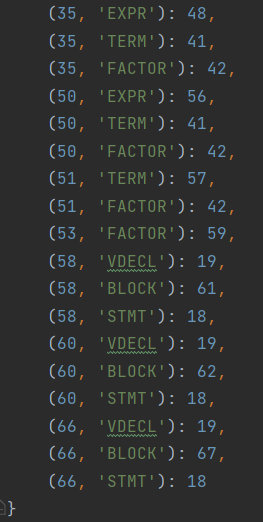






**3) SLR parsing table의 Goto table 정의**





**4) lexical.py 코드 읽어오기**

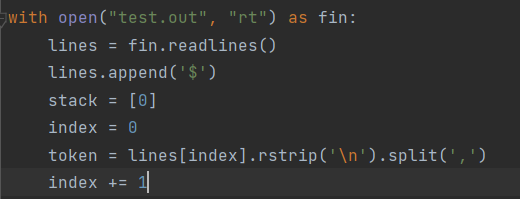


**5) doneFlag 정의**



doneFlag – ‘$’ 들어올 경우 모든 과정이 끝났음을 의미하는 flag

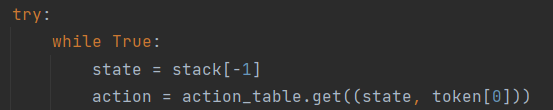
**6) test.out 파일을 읽어오기**



‘test.out’ 파일을 읽어온 후 각 줄을 리스트에 저장한 후 마지막에 ‘$’를 추가한다.

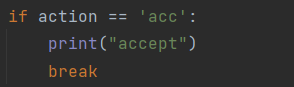
Stack과 index값을 초기화 해준 후 각 lines의 ‘\n’을 제거하고 ‘,’ 기준으로 split해서 token에 저장하고 index 값을 1씩 증가해준다.

**7) action\_table을 통한 action 값**



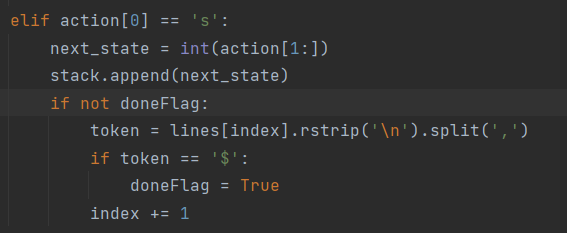
가장 위에 있는 stack 값을 state에 저장하고 state값과 token[0] 값에 맞는 action 값을 action\_table에서 찾아 action 변수에 저장한다.

**8) action 값이 ‘acc’인 경우**



action 값이 ‘acc’인 경우 “accept”를 출력하고 while 문을 나가고 끝이 난다.

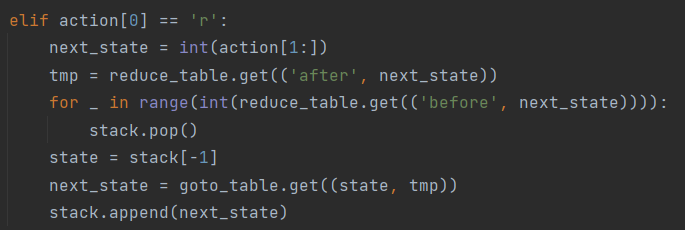
**9) action이 shift인 경우**



action값의 처음이 ‘s’로 시작할 경우 shift를 의미하기 때문에 다음에 오는 숫자를 next\_state 변수에 저장 후 이를 stack에 추가한다.

doneFlag가 켜져 있지 않다면 다시 lines를 분리 후 token에 저장하고 그 token의 값이 ‘$’라면 끝났음을 의미하기 때문에 doneFlag를 켜준다.

**10) action이 reduce인 경우**

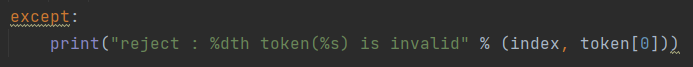


action값의 처음이 ‘r’로 시작하는 경우 이는 reduction을 의미한다. 다음 숫자를 next\_state에 저장한 후 reduce\_table에서 (‘after’, next\_state) 값을 찾아 tmp에 저장한다.

reduce\_table(‘before’, next\_state) 값의 경우 stack에서 제거해준 후 다시 state은 stack의 제일 위 숫자로 정의한다.

다시 next\_state의 값의 경우 goto\_table에서 값을 찾아와 stack에 추가한다.

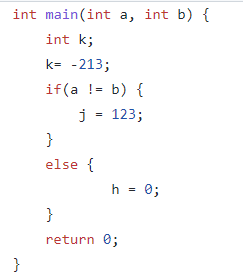
**11) invalid한 token이 들어왔을 경우**



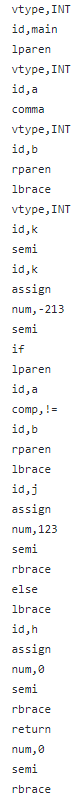
유효하지 않은 token의 경우 예외 처리를 통해 빠져나오게 된다.

(2) Implementation

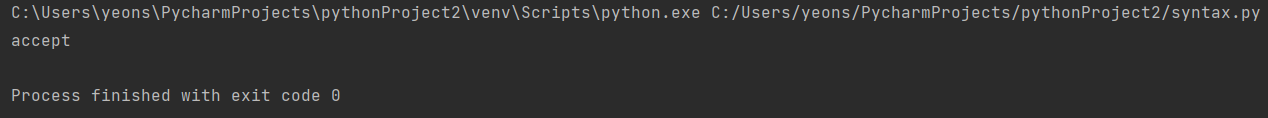
**test.c**

****

**test.out**

****

**syntax.py result**

****