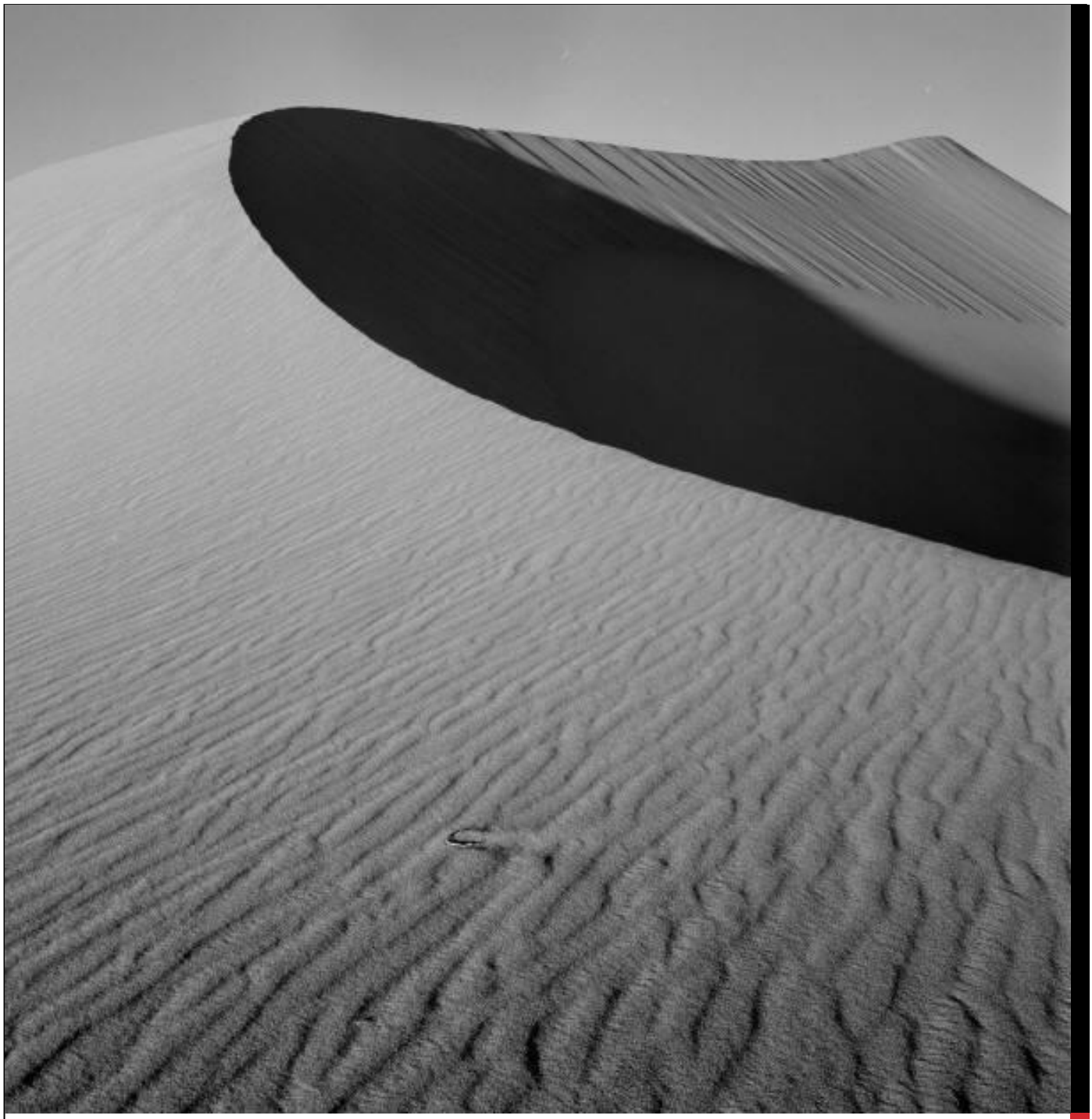

[TEAM 9]SYNTAX ANALYZER

소프트웨어공학과 20173483 김휘진

1. **Solve Ambiguous**

EXPR -> EXPR addsub EXPR | EXPR multdiv EXPR

이유 : addsub 와 multdiv 와의 우선순위가 지정되지 않는다.

해결 : EXPRT 에서 addsub, EXPRF 에서 multdiv 를 지정하도록 한다.

```
EXPR -> EXPRT addsub EXPR
EXPR -> EXPRT
EXPRT -> EXPRF multdiv EXPRT
EXPRT -> EXPRF
EXPRF -> lparen EXPR rparen
EXPRF -> id
EXPRF -> num
```

COND -> COND comp COND

이유 : COND comp COND comp COND 일경우 tree 가 여러 개 지정가능하다.

해결 : 중복된 non-terminal 을 각각 생성해준다.

```
COND -> CONDL comp COND
COND -> CONDL
CONDL -> boolstr
```

2. 총 Grammer

```
CODE' -> CODE
CODE -> VDECL CODE
CODE -> FDECL CODE
CODE -> CDECL CODE
CODE -> ''
VDECL->vtype id semi
VDECL -> vtype ASSIGN semi
ASSIGN -> id assign RHS
RHS -> EXPR
RHS -> literal
RHS -> character
RHS -> boolstr
EXPR -> EXPRT addsub EXPR
EXPR -> EXPRT
EXPRT -> EXPRF multdiv EXPRT
EXPRT -> EXPRF
EXPRF -> lparen EXPR rparen
EXPRF -> id
EXPRF -> num
FDECL -
> vtype id lparen ARG rparen lbrace BLOCK RETURN rbrace
ARG -> vtype id MOREARGS
ARG -> ''
MOREARGS -> comma vtype id MOREARGS
MOREARGS -> ''
BLOCK -> STMT BLOCK
BLOCK -> ''
STMT -> VDECL
STMT -> ASSIGN semi
STMT -> if lparen COND rparen lbrace BLOCK rbrace ELSE
STMT -> while lparen COND rparen lbrace BLOCK rbrace ELSE
COND -> CONDL comp COND
COND -> CONDL
CONDL -> boolstr
```

```
ELSE -> else lbrace BLOCK rbrace
ELSE -> ''
RETURN -> return RHS semi
CDECL -> class id lbrace ODECL rbrace
ODECL -> VDECL ODECL
ODECL -> FDECL ODECL
ODECL -> ''
```

3. **SLR table**

http://jsmachines.sourceforge.net/machines/slr.html

4. **Implementation details**

Tokens -> lexical analyzer 의 return list

Tokens type : deque([("Non-Terminal", "Token string") , …])

Grammer -> [("RETURN" , ("return RHS semi"), …] 공백을 기준으로 나눔

LR_tabel[0]["vtype"] = "s5" , LR_tabel[0]["class"] = "s6" 의 형태로 List 에 각원소로 dict 이 저장된다.

Stack -> list 형식 state 와 token 이 저장된다. 맨처음은 [0]으로 초기화됨.

State -> stack top 이 int 일경우 state = stack top

Token = tokens[0][0] Tokens deque 의 맨처음 non-terminal 을 저장한다.

Action = LR table 에 state 와 token 에 맞는 값 , action = LR_tabel[state][token]

parsing 과정 :

while 문에서 stack top 과, tokens 의 맨처음 token 을 통해 action 을 찾는다.

Action 에 따라 행동을 지정한다.

- S 의 경우 token 을 stack 에 append, tokens 에서 popleft 하고,

State 를 stack 에 append 한다.

- R 의 경우 action 의 state 를 통해 grammer 의 reduce grammer 를 찾는다.

Reduce grammer 에 해당하는지 stack 의 non-terminal 을 확인한후,

마지막 non-terminal 까지 pop 한후, 목적 non-terminal 을 stack 에 append 한다.

- Goto 문일 경우, LR tabel 에 state 를 stack 에 추가한다.

각 Step 별 과정을 log 파일에 저장한다.

5. 실행

Syntex_analyzer.py 를 실행하면 lexical_analyzer, grammer, lr_table 이 import 되어 실행된다.

Input_string.txt 를 accept 하는지, reject 하는지 여부를 print 한다.

parsing 하는 과정은 syntex_analyzer_log/.log 파일에 저장된다.