Compiler Term Project2

이름: 노현진

학번: 20183784

[CFG G]

1. CODE 🡪 DECL
2. DECL 🡪 VDECL CODE
3. DECL 🡪 FDECL CODE
4. DECL 🡪 ϵ
5. VDECL -> vtype id semi
6. FDECL -> vtype id lparen ARG rparen lbrace BLOCK RETURN rbrace
7. ARG -> vtype id MOREARGS
8. ARG -> ϵ
9. MOREARGS -> comma vtype id MOREARGS
10. MOREARGS -> ϵ
11. BLOCK -> STMT BLOCK
12. BLOCK -> ϵ
13. STMT -> VDECL
14. STMT -> id assign RHS semi
15. STMT -> if lparen COND rparen lbrace BLOCK rbrace else lbrace BLOCK rbrace
16. STMT -> while lparen COND rparen lbrace BLOCK rbrace
17. RHS -> EXPR
18. RHS -> literal
19. EXPR -> TERM addsub EXPR
20. EXPR -> TERM
21. TERM -> FACTOR multidiv TERM
22. TERM -> FACTOR
23. FACTOR -> lparen EXPR rparen
24. FACTOR -> id
25. FACTOR -> num
26. COND -> FACTOR comp FACTOR
27. RETURN -> return FACTOR semi

CFG G의 변경점은 다음과 같습니다.

변경 전:

1. CODE 🡪 VDECL CODE
2. CODE 🡪 FDECL CODE
3. CODE 🡪 ϵ

변경 후:

1. CODE 🡪 DECL
2. DECL 🡪 VDECL CODE
3. DECL 🡪 FDECL CODE
4. DECL 🡪 ϵ

변경 이유: SLR paring table을 <http://jsmachines.sourceforge.net/machines/slr.html>

website에서 생성하려고 할 때 start symbol인 CODE가 여러 개의 CFG을 갖고 있으면 부정확한 SLR parsing table이 생성되는 문제가 존재하였습니다. 이 문제를 해결하기 위해 DECL이란 새로운 symbol을 추가하였습니다.

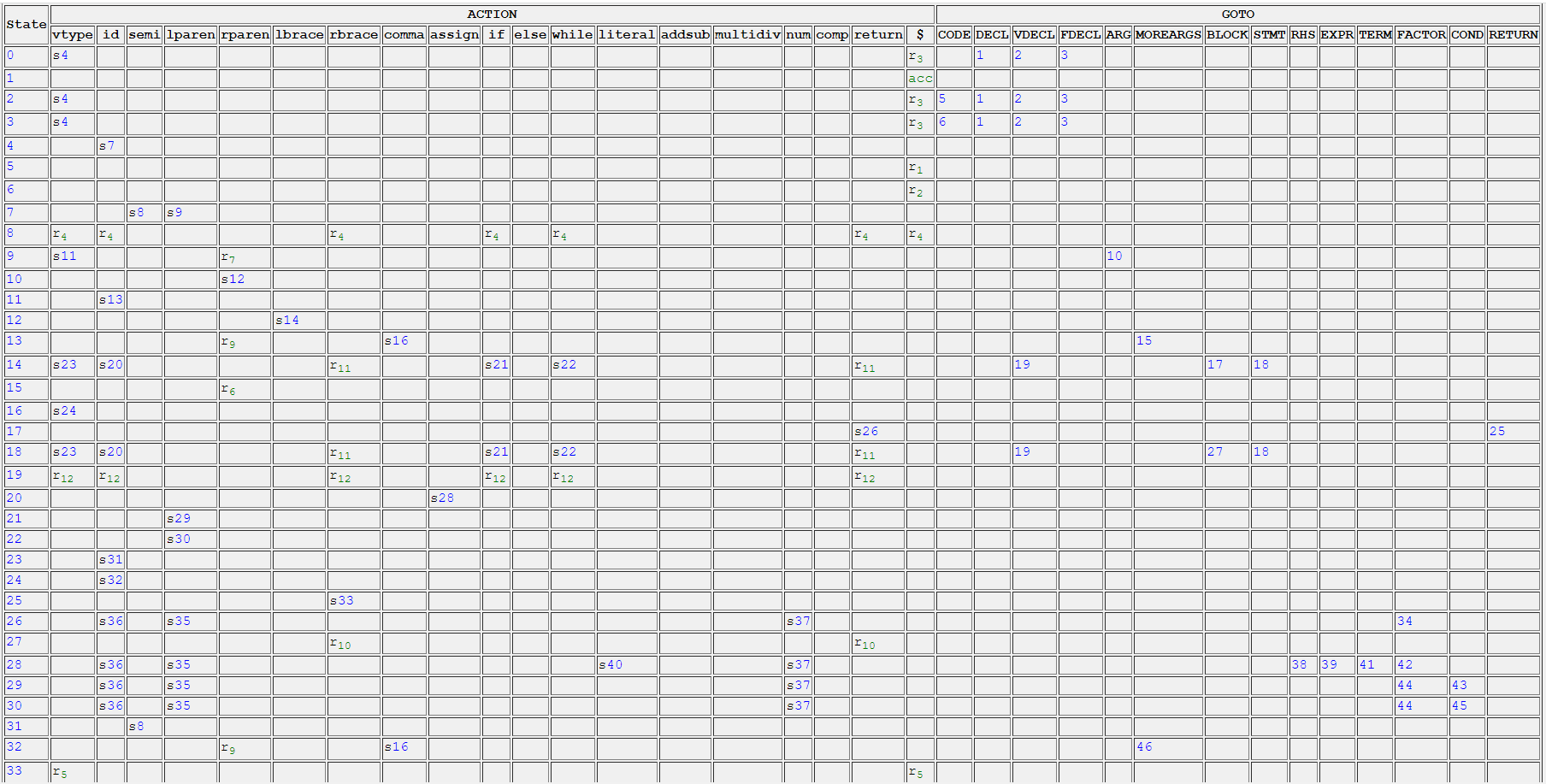
* Terminals:
* vtype for the types of variables and functions. In my program, the token name is [VTYPE].
* num for signed integer. In my program, the token name is [INTEGER].
* literal for literal strings. In my program, the token name is [STRING].
* id for the identifiers of variables and functions. In my program, the token name is [ID].
* if, else, while, and return for if, else, while, and return statements respectively. In my program, the tokens names are [IF], [ELSE], [WHILE], and [return] respectively.
* addsub for + and – arithmetic operators. In my program, the token name is [ADDSUB].
* multidiv for \* and / arithmetic operators. In my program, the token name is [MULTIDIV].
* assign for assignment operators. In my program, the token name is [ASSIGN].
* comp for comparison operators. In my program, the token name is [COMPARISON].
* semi and comma for semicolons and commas respectively. In my program, the token names are [SEMI] and [COMMA] respectively.
* lparen, rparen, lbrace, and rbrace for (, ), {, and } resprectively. In my program, the token names are [LPAREN], [RPAREN], [LBRACE], and [RBRACE] respectively.

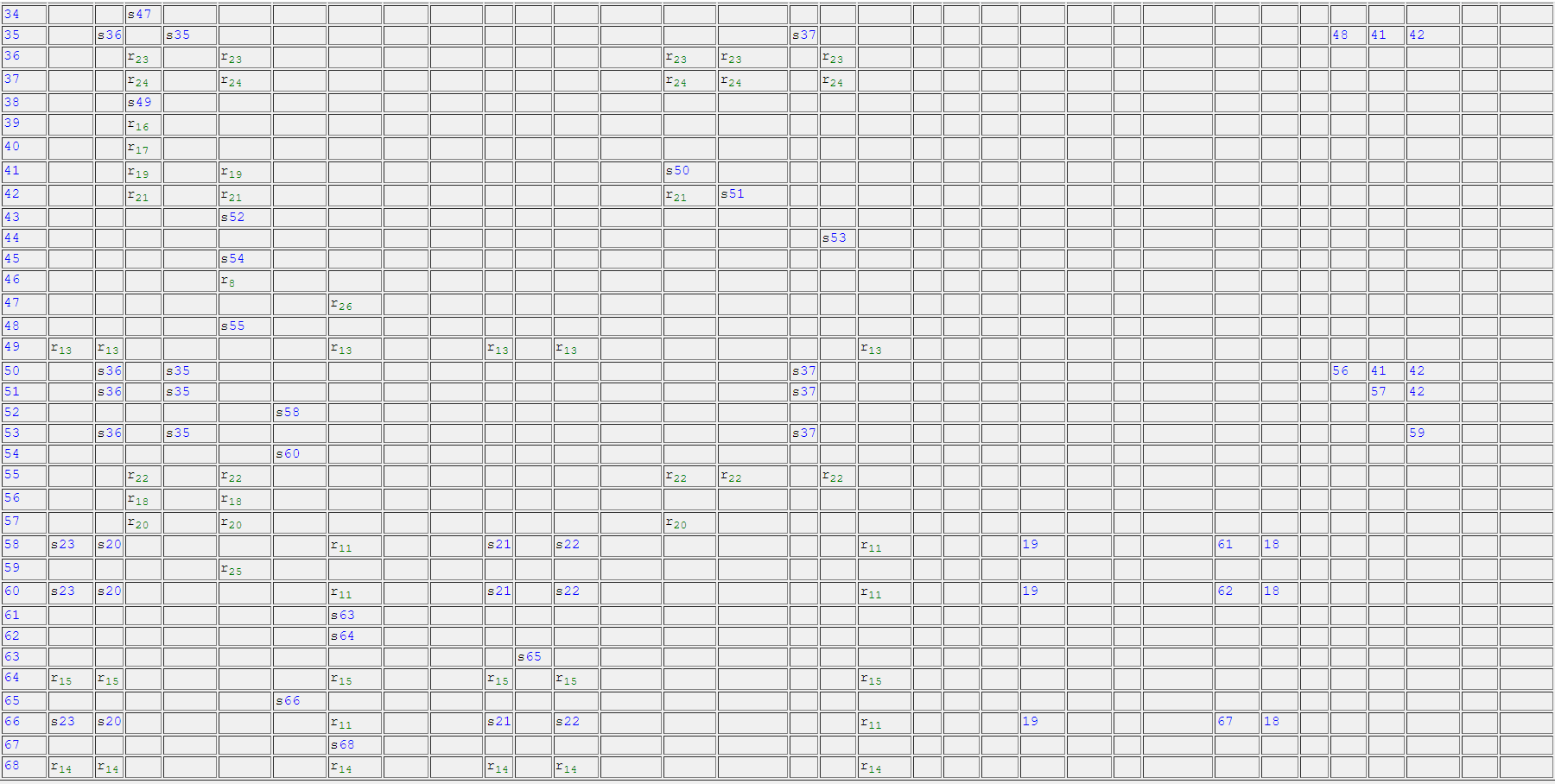
위에서 token name은 lexical analysis을 통해 생성된 token의 이름을 나타냅니다. [return]의 경우 Non-terminal [RETURN]과 겹치는 문제를 해결하고자 소문자로 지정하였습니다.

* Non-terminals:
* CODE, DECL, VDECL, FDECL, ARG, MOREARGS, BLOCK, STMT, RHS, EXPR, TERM, FACTOR, COND, RETURN
* Start Symbol: CODE
* Descriptions:
* The given CFG F is not ambiguous and non-left recursive.
* Source codes include zero or more declarations of functions and variables.
* Variables are always declared without initialization.
* Functions can have zero or more input arguments.
* Function blocks include zero or more statements.
* There are four type of statements: variable declarations, assignment operations, if-else statements, and while statements.
* if-else statements without else are not allowed.
* The right hand size of assignment operations can be classified into two types: arithmetic operations (expressions) and literal strings.
* Arithmetic operations are the combinations of +, -, \*, / operators.

[My SLR parsing table]

위의 CFG G을 이용해 생성한 SLR parsing table은 다음과 같습니다.



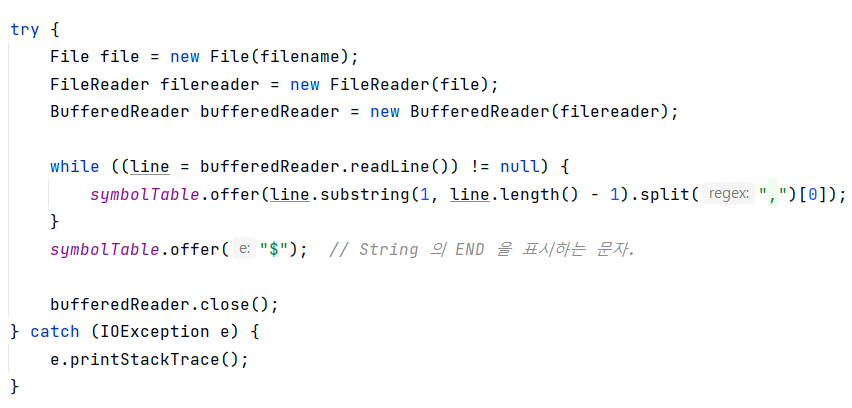


[How my syntax analyzer works for validating token sequences]

all about how your syntax analyzer works for validating token sequences (for example, overall procedures, implementation details like algorithms and data structures, working examples, and so on)

이 Syntax analyzer은 전반적으로 다음과 같은 과정을 수행한다.

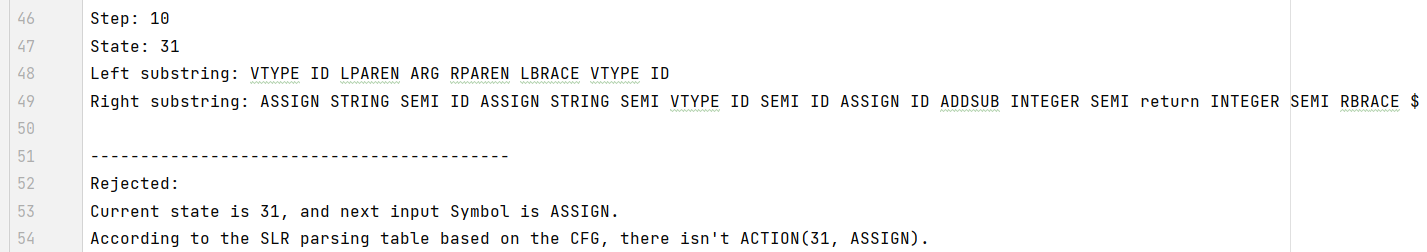
1. Lexical analysis을 통해 생성된 Symbol table(Output.txt)에서 token 정보들을 읽어 queue 타입의 symbolTable 변수에 저장한다. 여기서 symbolTable은 parsing 과정에서의 right substring에 해당한다. 모든 token 정보들을 저장했으면 마지막으로 “$” 문자를 저장하여 sequence of tokens의 끝을 표시해둔다.



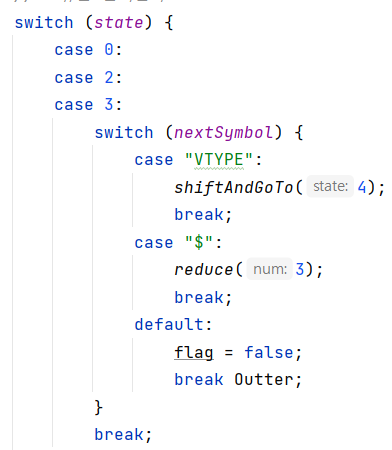
1. state들의 정보를 담은 Stack 타입의 변수 states에 state의 초깃값으로 0을 push한다. 그리고 next input symbol을 확인한다.



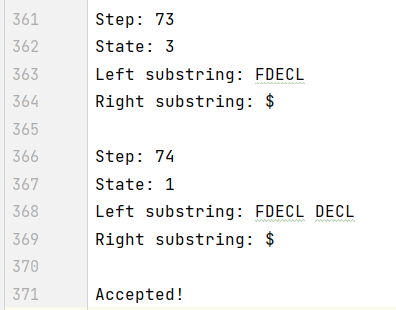
1. 현재 state와 next symbol에 따라 shift & goto 과정을 거칠지, 아니면 reduce 과정을 거칠지를 결정한다.
2. 만약 현재 state에서 next symbol을 읽었을 때 해당하는 ACTION이 존재하지 않는 경우 parsing 과정을 멈추고 “rejected”을 출력한다. 그리고 error가 발생한 위치와 이유를 출력한다.



1. Shift & goto 과정을 거치기로 결정했으면, 다음 state 값을 states 변수에 push하고 shift 과정을 거친다. Reduce 과정을 거치기로 결정했으면 CFG G에 해당하는 handle 값을 찾아 reduce하고 goto 과정을 거쳐서 다음 state 값을 states 변수에 push한다.



1. 이와 같은 과정을 거쳐서 parsing이 완료되면 “Accepted!”을 출력한다.

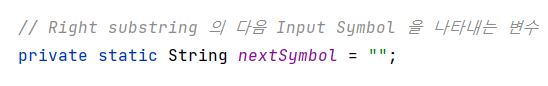


Syntax analysis을 위해 사용한 주요 변수와 메소드들에 대해 설명하자면 다음과 같다.

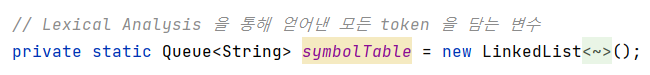
* State 변수: SLR Table의 현재 state을 나타내는 변수이다.



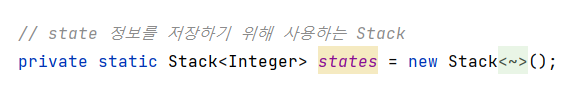
* nextSymbol 변수: Right substring의 다음 input symbol을 나타내는 변수.



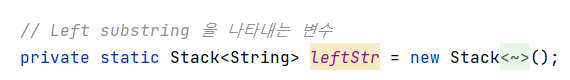
* symbolTable 변수: Lexical Analysis을 통해 얻어낸 sequence of tokens 정보를 담는 변수



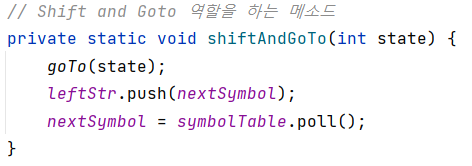
* states 변수: state 값들을 저장하는 변수



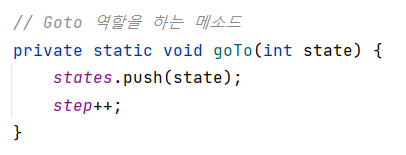
* leftStr 변수: parsing 과정에서 left substring에 해당하는 변수



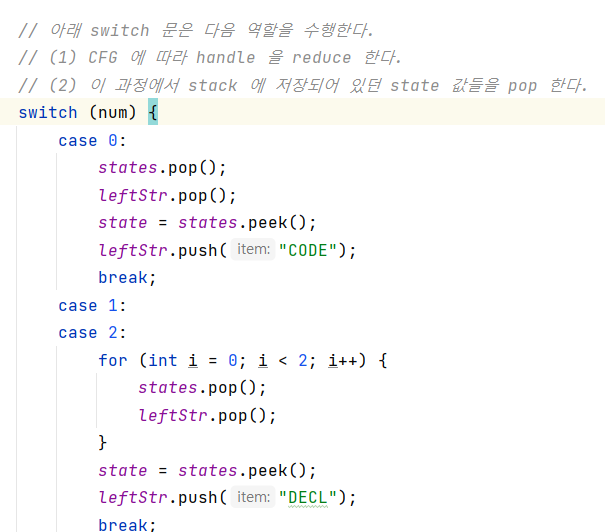
* shiftAndGoTo 메소드: shift and goto 역할을 하는 메소드이다.



* goTo 메소드: goto 역할을 하는 메소드로 shiftAndGoto 메소드의 일부로 사용되거나, reduce 과정에서의 goto 기능을 수행한다.



* reduce 메소드: reduce 역할을 하는 메소드이다. 이 메소드는 먼저 CFG에 따라 handle을 reduce한다. 그리고 이 과정에서 states 변수에 저장되어 있던 state 값들을 pop한다.



Reduce한 다음에는 현재 state와 reduce된 symbol에 따라 goto 과정을 거쳐 다음 state 값을 states 변수에 push한다.

