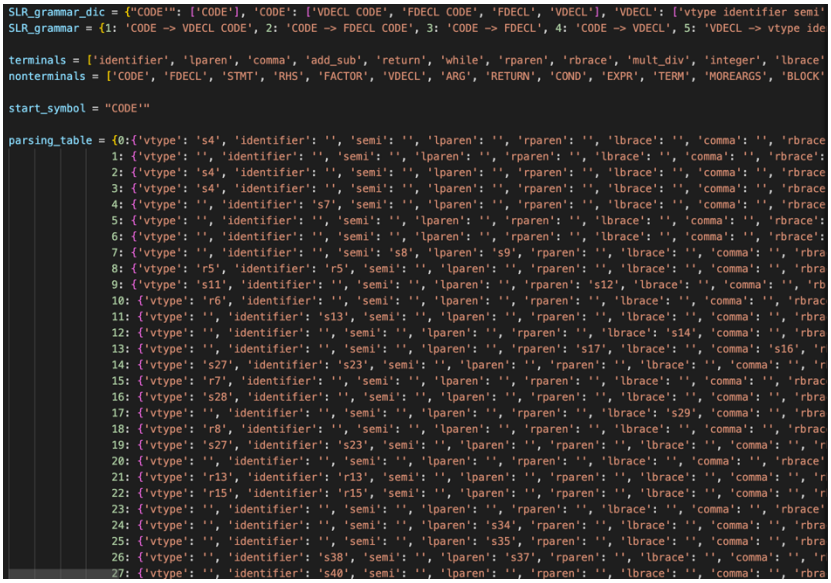
Student Number : 20200123
Student Name : Hwang seo-jin
Team : Team 29 (1 인)

1. **My SLR parsing table**

LR table

State	ACTION																			GOTO													
	vtype	identifier	semi	lparen	rparen	lbrace	comma	rbrace	else	assign	if	while	literal	addsub	multdiv	integer	comparison	return	$	CODE'	CODE	VDECL	FDECL	ARG	MOREARGS	BLOCK	STMT	RHS	EXPR	TERM	FACTOR	COND	RETURN
0	s4																				1	2	3											
1																			acc														
2	s4																		r_4		5	2	3										
3	s4																		r_3		6	2	3										
4		s7																															
5																			r_1														
6																			r_2														
7			s8	s9																													
8	r_5	r_5						r_5			r_5	r_5						r_5	r_5														
9	s11				s12																			10									
10	r_6																		r_6														
11		s13																															
12						s14																											
13					s17		s16																		15								
14	s27	s23						s21			s24	s25						s26				22					18	19					20
15	r_7																		r_7														
16	s28																																
17						s29																											
18	r_8																		r_8														
19	s27	s23						s21			s24	s25						s26				22					30	19					20
20								s31																									
21	r_{13}	r_{13}						r_{13}	s32		r_{13}	r_{13}						r_{13}	r_{13}														
22	r_{15}	r_{15}						r_{15}			r_{15}	r_{15}						r_{15}															
23										s33																							
24				s34																													
25				s35																													
26		s38		s37												s39																	
27		s40																															
28		s41																															
29	s27	s23						s21			s24	s25						s26				22					42	19					20
30	r_{11}	r_{11}						r_{11}			r_{11}	r_{11}						r_{11}	r_{11}														
31	r_{12}	r_{12}						r_{12}			r_{12}	r_{12}						r_{12}	r_{12}														
32						s43																											
33		s38		s37									s46			s39												44	45	47	48		
34		s38		s37												s39															50	49	
35		s38		s37												s39															50	51	
36			s52																														
37		s38		s37												s39													53	47	48		
38			r_{26}		r_{26}									r_{26}	r_{26}		r_{26}																
39			r_{27}		r_{27}									r_{27}	r_{27}		r_{27}																
40			s8																														
41					s17		s16																		54								
42	r_{10}																		r_{10}														
43	s27	s23						s21			s24	s25						s26				22					55	19					20
44			s56																														
45			r_{19}																														
46			r_{20}																														
47			r_{22}		r_{22}									s57																			
48			r_{24}		r_{24}									r_{24}	s58																		
49					s59																												
50																	s60																
51					s61																												
52								r_{29}																									
53					s62																												
54	r_9																		r_9														
55	r_{14}	r_{14}						r_{14}			r_{14}	r_{14}						r_{14}	r_{14}														
56	r_{16}	r_{16}						r_{16}			r_{16}	r_{16}						r_{16}															
57		s38		s37												s39													63	47	48		
58		s38		s37												s39														64	48		
59						s65																											
60		s38		s37												s39															66		
61						s67																											
62			r_{25}		r_{25}									r_{25}	r_{25}		r_{25}																
63			r_{21}		r_{21}																												
64			r_{23}		r_{23}									r_{23}																			
65	s27	s23						s21			s24	s25						s26				22					68	19					20
66					r_{28}																												
67	s27	s23						s21			s24	s25						s26				22					69	19					20
68	r_{17}	r_{17}						r_{17}			r_{17}	r_{17}						r_{17}															
69	r_{18}	r_{18}						r_{18}			r_{18}	r_{18}																					

2. ALL about how my syntax analyzer works



```
SLR_grammar_dic = {"CODE'": ['CODE'], 'CODE': ['VDECL CODE', 'FDECL CODE', 'FDECL', 'VDECL'], 'VDECL': ['vtype identifier semi'
SLR_grammar = {1: 'CODE -> VDECL CODE', 2: 'CODE -> FDECL CODE', 3: 'CODE -> FDECL', 4: 'CODE -> VDECL', 5: 'VDECL -> vtype ide

terminals = ['identifier', 'lparen', 'comma', 'add_sub', 'return', 'while', 'rparen', 'rbrace', 'mult_div', 'integer', 'lbrace'
nonterminals = ['CODE', 'FDECL', 'STMT', 'RHS', 'FACTOR', 'VDECL', 'ARG', 'RETURN', 'COND', 'EXPR', 'TERM', 'MOREARGS', 'BLOCK'

start_symbol = "CODE'"

parsing_table = {0:{'vtype': 's4', 'identifier': '', 'semi': '', 'lparen': '', 'rparen': '', 'lbrace': '', 'comma': '', 'rbrace
                 1: {'vtype': '', 'identifier': '', 'semi': '', 'lparen': '', 'rparen': '', 'lbrace': '', 'comma': '', 'rbrace':
                 2: {'vtype': 's4', 'identifier': '', 'semi': '', 'lparen': '', 'rparen': '', 'lbrace': '', 'comma': '', 'rbrace
                 3: {'vtype': 's4', 'identifier': '', 'semi': '', 'lparen': '', 'rparen': '', 'lbrace': '', 'comma': '', 'rbrace
                 4: {'vtype': '', 'identifier': 's7', 'semi': '', 'lparen': '', 'rparen': '', 'lbrace': '', 'comma': '', 'rbrace
                 5: {'vtype': '', 'identifier': '', 'semi': '', 'lparen': '', 'rparen': '', 'lbrace': '', 'comma': '', 'rbrace':
                 6: {'vtype': '', 'identifier': '', 'semi': '', 'lparen': '', 'rparen': '', 'lbrace': '', 'comma': '', 'rbrace':
                 7: {'vtype': '', 'identifier': '', 'semi': 's8', 'lparen': 's9', 'rparen': '', 'lbrace': '', 'comma': '', 'rbra
                 8: {'vtype': 'r5', 'identifier': 'r5', 'semi': '', 'lparen': '', 'rparen': '', 'lbrace': '', 'comma': '', 'rbra
                 9: {'vtype': 's11', 'identifier': '', 'semi': '', 'lparen': '', 'rparen': 's12', 'lbrace': '', 'comma': '', 'rb
                 10: {'vtype': 'r6', 'identifier': '', 'semi': '', 'lparen': '', 'rparen': '', 'lbrace': '', 'comma': '', 'rbrac
                 11: {'vtype': '', 'identifier': 's13', 'semi': '', 'lparen': '', 'rparen': '', 'lbrace': '', 'comma': '', 'rbra
                 12: {'vtype': '', 'identifier': '', 'semi': '', 'lparen': '', 'rparen': '', 'lbrace': 's14', 'comma': '', 'rbra
                 13: {'vtype': '', 'identifier': '', 'semi': '', 'lparen': '', 'rparen': 's17', 'lbrace': '', 'comma': 's16', 'r
                 14: {'vtype': 's27', 'identifier': 's23', 'semi': '', 'lparen': '', 'rparen': '', 'lbrace': '', 'comma': '', 'r
                 15: {'vtype': 'r7', 'identifier': '', 'semi': '', 'lparen': '', 'rparen': '', 'lbrace': '', 'comma': '', 'rbrac
                 16: {'vtype': 's28', 'identifier': '', 'semi': '', 'lparen': '', 'rparen': '', 'lbrace': '', 'comma': '', 'rbra
                 17: {'vtype': '', 'identifier': '', 'semi': '', 'lparen': '', 'rparen': '', 'lbrace': 's29', 'comma': '', 'rbra
                 18: {'vtype': 'r8', 'identifier': '', 'semi': '', 'lparen': '', 'rparen': '', 'lbrace': '', 'comma': '', 'rbrac
                 19: {'vtype': 's27', 'identifier': 's23', 'semi': '', 'lparen': '', 'rparen': '', 'lbrace': '', 'comma': '', 'r
                 20: {'vtype': '', 'identifier': '', 'semi': '', 'lparen': '', 'rparen': '', 'lbrace': '', 'comma': '', 'rbrace'
                 21: {'vtype': 'r13', 'identifier': 'r13', 'semi': '', 'lparen': '', 'rparen': '', 'lbrace': '', 'comma': '', 'r
                 22: {'vtype': 'r15', 'identifier': 'r15', 'semi': '', 'lparen': '', 'rparen': '', 'lbrace': '', 'comma': '', 'r
                 23: {'vtype': '', 'identifier': '', 'semi': '', 'lparen': '', 'rparen': '', 'lbrace': '', 'comma': '', 'rbrace'
                 24: {'vtype': '', 'identifier': '', 'semi': '', 'lparen': 's34', 'rparen': '', 'lbrace': '', 'comma': '', 'rbra
                 25: {'vtype': '', 'identifier': '', 'semi': '', 'lparen': 's35', 'rparen': '', 'lbrace': '', 'comma': '', 'rbra
                 26: {'vtype': '', 'identifier': 's38', 'semi': '', 'lparen': 's37', 'rparen': '', 'lbrace': '', 'comma': '', 'r
                 27: {'vtype': '', 'identifier': 's40', 'semi': '', 'lparen': '', 'rparen': '', 'lbrace': '', 'comma': '', 'rbra
```

Defined **SLR_grammar_dic** and **SLR_grammar** from CFG, **terminals** & **nonterminals**, **start_symbol.**
Most importantly, implemented **parsing_table** based on my SLR parsing table at before page of this documentation.

```
# input (using lexical_analyzer output!)
lexical_file = open('lexical_test.out','r')
# input lexemes for pointing out where error has occured
lexeme_file = open('lexical_lexeme.out', 'r')

# ouput (syntax analyzer output : whether the grammar is accepted or not)
syntax_file = open('syntax_test.out','w')

# lexemes for pointing out where error has occured
lexemes = lexeme_file.read()
# file input uses as a string.
lexical_output = lexical_file.read().lower()

token_stream = (lexical_output + " $").split()
lexeme_stream = (lexemes).split()

# for counting steps
step_num = 0
# stack neeed for parsing
syntax_stack = ['0']

pointer = 0
token = token_stream[pointer]
```

Open files. Because we are using lexical_analyzer.py's output for input, we open 'lexical_test.out', which only includes tokens.
Also for pointing out where the error has occurred, I opened 'lexical_lexeme.out' which contains only lexemes in the same order with 'lexical_test.out'
And I **defined and initialized some values needed later on**.

```python
# Method to find First set.
def FIRST_SET(nonterminal):
    # if it is terminal, just return!
    if nonterminal in terminals:
        return {nonterminal}

    first = set([])

    first_list = [nonterminal]

    for next_symbol in SLR_grammar_dic[nonterminal]:
        for (i, next) in enumerate(next_symbol.split()):
            length = len(next_symbol)

            if next not in first_list:
                # check next's FIRST SET
                symbol_first = FIRST_SET(next)

                for temp in symbol_first:
                    if temp in terminals:
                        first.add(temp)

                if '^' not in symbol_first:
                    break
            else:
```

```python
# Method to find FOLLOW set.
def FOLLOW_SET(nonterminal):

    follow = set([])
    follow_list = [nonterminal]

    # starting symbol add $
    if nonterminal == start_symbol:
        follow.add('$')

    for (before, next_symbols) in SLR_grammar_dic.items():
        for next in next_symbols:
            next = next.split()

            if nonterminal in next[:-1]:
                index_search = next.index(nonterminal) + 1

                first = FIRST_SET(next[index_search])
                follow = follow | (first - set('^'))

                if '^' in first:
                    if before not in follow_list:
                        follow = follow | FOLLOW_SET(before)

            elif nonterminal in next[-1]:
                if before not in follow_list:
```

Defined FIRST_SET function, the method to find the **FIRST SET** of the symbol and **FOLLOW_SET function**, the method to find **FOLLOW SET**.

```python
# SLR Parser to check whether the Grammar is accepted or not.
while 1:
    try:
        step_num += 1
        state = int(syntax_stack[-1])

        # If there is no Symbols in parsing table, print ERROR & break
        if token not in parsing_table[state].keys():
            error_message = "\nERROR occured at : " + str(step_num) + "\n"
            print("\nREJECTED! THERE IS NO MATCHING SYMBOL NAMED" + token + "\n")
            print(error_message)
            break

        # If all grammar accepted, write and print accept message & break
        elif parsing_table[state][token] == "acc":
            accept_message = "\nACCEPTED! GRAMMATICALLY CORRECT\n"
            syntax_file.write(accept_message)
            print(accept_message)
            break

        # GOTO
        elif parsing_table[state][token][0] == "r":
            next_state = int(parsing_table[state][token][1:])
            grammar = SLR_grammar[next_state].split()

            if grammar[-1] != '^':
                after_index = grammar.index('->') + 1
                boundary_len = 2 * len(grammar[after_index:])
```

THE MAIN KEY PART! **Implemented SLR Parser part to check whether the Grammar is accepted or not**!

- If there is no corresponding symbol in parsing table, we print ERROR that there is no matched symbol.
- If all token's grammar are accepted, we write and print the accept message.

```
seojin@hwangseojin-ui-MacBookPro lexical_analyzer % python3 syntax_analyzer.py

ACCEPTED! GRAMMATICALLY CORRECT
```

- Apart from this, handled GOTO, and ACTION
- If IndexError occurs, it means grammatical error! (no corresponding symbol in particular state in parsing table) So we report ERROR and explain why and where(At ?th lexeme, lexeme name) the Error occurred.

```
REJECTED!

GRAMMATICAL ERROR found at : 12th lexeme(token)'('
```

3. Test input files and outputs which I used

- **1) First**
  - **Input : Test.c**

```
int operationfunc(int a, int b)
{
    d = 0;

    while (b<=1){
        a = 1;
    }

    return 2;
}
```

  - **Output :**

```
ACCEPTED! GRAMMATICALLY CORRECT
```

- **2) Second**
  - **Input :** Test2.c

```
int operationfunc1(int a, int b)
{
    print("Hello World");

    char c = (char)a - b;
    int d = -2134;
    d = 0;
}
```

  - **Output :** rejected because in CFG that we are using, there is no grammar that defines '(' comes after the identifier **!**

```
REJECTED!

GRAMMATICAL ERROR found at : 12th lexeme(token)'('
```

- **3) Third**
  - **Input** : Test3.c

```
int operationfunc2(int a, int b)
{
    char c = (char)a - b;
    int d = -2134;
    d = 0;

    while (b<=1){
        a = 1;
    }

    return 2;
}
```

  - **Output :** rejected because in CFG that we are using, there is no grammar that defines '*vtype identifier assign*' **!**

```
REJECTED!

GRAMMATICAL ERROR found at : 13th lexeme(token)'='
```

Just In case, I attach the whole capture of the SLR Parser site(where I made parsing table) next page!

SLR grammar ('' is ε):

```
(0)  CODE' -> CODE
(1)  CODE -> VDECL
(2)  VDECL -> vtype identifier
     semi
(3)  FDECL -> vtype identifier
     lparen ARG
(4)  ARG -> vtype identifier
     MOREARGS
(5)  ARG -> rparen lbrace BLOCK
(6)  MOREARGS -> comma vtype
     identifier MOREARGS
(7)  MOREARGS -> rparen lbrace
     BLOCK
(8)  BLOCK -> STMT BLOCK
(9)  BLOCK -> RETURN rbrace
(10) BLOCK -> rbrace
(11) BLOCK -> rbrace else lbrace
     BLOCK
```

FIRST / FOLLOW table

Nonterminal	FIRST	FOLLOW
CODE'	{vtype}	{$}
CODE	{vtype}	{$}
VDECL	{vtype}	{vtype,$,rbrace,identifier,if,while,return}
FDECL	{vtype}	{vtype,$}
ARG	{vtype,rparen}	{vtype,$}
MOREARGS	{comma,rparen}	{vtype,$}
BLOCK	{rbrace,vtype,identifier,if,while,return}	{vtype,$,rbrace,identifier,if,while,return}
STMT	{vtype,identifier,if,while}	{rbrace,vtype,identifier,if,while,return}
RHS	{literal,lparen,identifier,integer}	{semi}
EXPR	{lparen,identifier,integer}	{semi,rparen}
TERM	{lparen,identifier,integer}	{addsub,semi,rparen}
FACTOR	{lparen,identifier,integer}	{multdiv,addsub,semi,comparison,rparen}
COND	{lparen,identifier,integer}	{rparen}
RETURN	{return}	{rbrace}

SLR closure table

Goto	Kernel	State	Closure
	{CODE' -> .CODE}	0	{CODE' -> .CODE; CODE -> .VDECL CODE; CODE -> .FDECL CODE; CODE -> .FDECL; CODE -> .VDECL; VDECL -> .vtype identifier semi; FDECL -> .vtype identifier lparen ARG}
goto(0, CODE)	{CODE' -> CODE.}	1	{CODE' -> CODE.}
goto(0, VDECL)	{CODE -> VDECL.CODE; CODE -> VDECL.}	2	{CODE -> VDECL.CODE; CODE -> VDECL.; CODE -> .VDECL CODE; CODE -> .FDECL CODE; CODE -> .FDECL; CODE -> .VDECL; VDECL -> .vtype identifier semi; FDECL -> .vtype identifier lparen ARG}
goto(0, FDECL)	{CODE -> FDECL.CODE; CODE -> FDECL.}	3	{CODE -> FDECL.CODE; CODE -> FDECL.; CODE -> .VDECL CODE; CODE -> .FDECL CODE; CODE -> .FDECL; CODE -> .VDECL; VDECL -> .vtype identifier semi; FDECL -> .vtype identifier lparen ARG}
goto(0, vtype)	{VDECL -> vtype.identifier semi; FDECL -> vtype.identifier lparen ARG}	4	{VDECL -> vtype.identifier semi; FDECL -> vtype.identifier lparen ARG}
goto(2, CODE)	{CODE -> VDECL CODE.}	5	{CODE -> VDECL CODE.}
goto(2, VDECL)	{CODE -> VDECL.CODE; CODE -> VDECL.}	2	
goto(2, FDECL)	{CODE -> FDECL.CODE; CODE -> FDECL.}	3	
goto(2, vtype)	{VDECL -> vtype.identifier semi; FDECL -> vtype.identifier lparen ARG}	4	
goto(3, CODE)	{CODE -> FDECL CODE.}	6	{CODE -> FDECL CODE.}
goto(3, VDECL)	{CODE -> VDECL.CODE; CODE -> VDECL.}	2	
goto(3, FDECL)	{CODE -> FDECL.CODE; CODE -> FDECL.}	3	
goto(3, vtype)	{VDECL -> vtype.identifier semi; FDECL -> vtype.identifier lparen ARG}	4	
goto(4, identifier)	{VDECL -> vtype identifier.semi; FDECL -> vtype identifier.lparen ARG}	7	{VDECL -> vtype identifier.semi; FDECL -> vtype identifier.lparen ARG}
goto(7, semi)	{VDECL -> vtype identifier semi.}	8	{VDECL -> vtype identifier semi.}
goto(7, lparen)	{FDECL -> vtype identifier lparen.ARG}	9	{FDECL -> vtype identifier lparen.ARG; ARG -> .vtype identifier MOREARGS; ARG -> .rparen lbrace BLOCK}
goto(9, ARG)	{FDECL -> vtype identifier lparen ARG.}	10	{FDECL -> vtype identifier lparen ARG.}
goto(9, vtype)	{ARG -> vtype.identifier MOREARGS}	11	{ARG -> vtype.identifier MOREARGS}
goto(9, rparen)	{ARG -> rparen.lbrace BLOCK}	12	{ARG -> rparen.lbrace BLOCK}
goto(11, identifier)	{ARG -> vtype identifier.MOREARGS}	13	{ARG -> vtype identifier.MOREARGS; MOREARGS -> .comma vtype identifier MOREARGS; MOREARGS -> .rparen lbrace BLOCK}
goto(12, lbrace)	{ARG -> rparen lbrace.BLOCK}	14	{ARG -> rparen lbrace.BLOCK; BLOCK -> .STMT BLOCK; BLOCK -> .RETURN rbrace; BLOCK -> .rbrace; BLOCK -> .rbrace else lbrace BLOCK; STMT -> .VDECL; STMT -> .identifier assign RHS semi; STMT -> .if lparen COND rparen lbrace BLOCK; STMT -> .while lparen COND rparen lbrace BLOCK; RETURN -> .return FACTOR semi; VDECL -> .vtype identifier semi}
goto(13, MOREARGS)	{ARG -> vtype identifier MOREARGS.}	15	{ARG -> vtype identifier MOREARGS.}
goto(13, comma)	{MOREARGS -> comma.vtype identifier MOREARGS}	16	{MOREARGS -> comma.vtype identifier MOREARGS}
goto(13, rparen)	{MOREARGS -> rparen.lbrace BLOCK}	17	{MOREARGS -> rparen.lbrace BLOCK}
goto(14, BLOCK)	{ARG -> rparen lbrace BLOCK.}	18	{ARG -> rparen lbrace BLOCK.}
goto(14, STMT)	{BLOCK -> STMT.BLOCK}	19	{BLOCK -> STMT.BLOCK; BLOCK -> .STMT BLOCK; BLOCK -> .RETURN rbrace; BLOCK -> .rbrace; BLOCK -> .rbrace else lbrace BLOCK; STMT -> .VDECL; STMT -> .identifier assign RHS semi; STMT -> .if lparen COND rparen lbrace BLOCK; STMT -> .while lparen COND rparen lbrace BLOCK; RETURN -> .return FACTOR semi; VDECL -> .vtype identifier semi}
goto(14, RETURN)	{BLOCK -> RETURN.rbrace}	20	{BLOCK -> RETURN.rbrace}
goto(14, rbrace)	{BLOCK -> rbrace.; BLOCK -> rbrace.else lbrace BLOCK}	21	{BLOCK -> rbrace.; BLOCK -> rbrace.else lbrace BLOCK}
goto(14, VDECL)	{STMT -> VDECL.}	22	{STMT -> VDECL.}
goto(14, identifier)	{STMT -> identifier.assign RHS semi}	23	{STMT -> identifier.assign RHS semi}
goto(14, if)	{STMT -> if.lparen COND rparen lbrace BLOCK}	24	{STMT -> if.lparen COND rparen lbrace BLOCK}
goto(14, while)	{STMT -> while.lparen COND rparen lbrace BLOCK}	25	{STMT -> while.lparen COND rparen lbrace BLOCK}
goto(14, return)	{RETURN -> return.FACTOR semi}	26	{RETURN -> return.FACTOR semi; FACTOR -> .lparen EXPR rparen; FACTOR -> .identifier; FACTOR -> .integer}
goto(14, vtype)	{VDECL -> vtype.identifier semi}	27	{VDECL -> vtype.identifier semi}
goto(16, vtype)	{MOREARGS -> comma vtype.identifier MOREARGS}	28	{MOREARGS -> comma vtype.identifier MOREARGS}
goto(17, lbrace)	{MOREARGS -> rparen lbrace.BLOCK}	29	{MOREARGS -> rparen lbrace.BLOCK; BLOCK -> .STMT BLOCK; BLOCK -> .RETURN rbrace; BLOCK -> .rbrace; BLOCK -> .rbrace else lbrace BLOCK; STMT -> .VDECL; STMT -> .identifier assign RHS semi; STMT -> .if lparen COND rparen lbrace BLOCK; STMT -> .while lparen COND rparen lbrace BLOCK; RETURN -> .return FACTOR semi; VDECL -> .vtype identifier semi}
goto(19, BLOCK)	{BLOCK -> STMT BLOCK.}	30	{BLOCK -> STMT BLOCK.}
goto(19, STMT)	{BLOCK -> STMT.BLOCK}	19	
goto(19, RETURN)	{BLOCK -> RETURN.rbrace}	20	
goto(19, rbrace)	{BLOCK -> rbrace.; BLOCK -> rbrace.else lbrace BLOCK}	21	
goto(19, VDECL)	{STMT -> VDECL.}	22	
goto(19, identifier)	{STMT -> identifier.assign RHS semi}	23	
goto(19, if)	{STMT -> if.lparen COND rparen lbrace BLOCK}	24	
goto(19, while)	{STMT -> while.lparen COND rparen lbrace BLOCK}	25	
goto(19, return)	{RETURN -> return.FACTOR semi}	26	
goto(19, vtype)	{VDECL -> vtype.identifier semi}	27	
goto(20, rbrace)	{BLOCK -> RETURN rbrace.}	31	{BLOCK -> RETURN rbrace.}
goto(21, else)	{BLOCK -> rbrace else.lbrace BLOCK}	32	{BLOCK -> rbrace else.lbrace BLOCK}
goto(23, assign)	{STMT -> identifier assign.RHS semi}	33	{STMT -> identifier assign.RHS semi; RHS -> .EXPR; RHS -> .literal; EXPR -> .TERM addsub EXPR; EXPR -> .TERM; TERM -> .FACTOR multdiv TERM; TERM -> .FACTOR; FACTOR -> .lparen EXPR rparen; FACTOR -> .identifier; FACTOR -> .integer}
goto(24, lparen)	{STMT -> if lparen.COND rparen lbrace BLOCK}	34	{STMT -> if lparen.COND rparen lbrace BLOCK; COND -> .FACTOR comparison FACTOR; FACTOR -> .lparen EXPR rparen; FACTOR -> .identifier; FACTOR -> .integer}
goto(25, lparen)	{STMT -> while lparen.COND rparen lbrace BLOCK}	35	{STMT -> while lparen.COND rparen lbrace BLOCK; COND -> .FACTOR comparison FACTOR; FACTOR -> .lparen EXPR rparen; FACTOR -> .identifier; FACTOR -> .integer}
goto(26, FACTOR)	{RETURN -> return FACTOR.semi}	36	{RETURN -> return FACTOR.semi}
goto(26, lparen)	{FACTOR -> lparen.EXPR rparen}	37	{FACTOR -> lparen.EXPR rparen; EXPR -> .TERM addsub EXPR; EXPR -> .TERM; TERM -> .FACTOR multdiv TERM; TERM -> .FACTOR; FACTOR -> .lparen EXPR rparen; FACTOR -> .identifier; FACTOR -> .integer}
goto(26, identifier)	{FACTOR -> identifier.}	38	{FACTOR -> identifier.}
goto(26, integer)	{FACTOR -> integer.}	39	{FACTOR -> integer.}
goto(27, identifier)	{VDECL -> vtype identifier.semi}	40	{VDECL -> vtype identifier.semi}
goto(28, identifier)	{MOREARGS -> comma vtype identifier.MOREARGS}	41	{MOREARGS -> comma vtype identifier.MOREARGS; MOREARGS -> .comma vtype identifier MOREARGS; MOREARGS -> .rparen lbrace BLOCK}
goto(29, BLOCK)	{MOREARGS -> rparen lbrace BLOCK.}	42	{MOREARGS -> rparen lbrace BLOCK.}
goto(29, STMT)	{BLOCK -> STMT.BLOCK}	19	
goto(29, RETURN)	{BLOCK -> RETURN.rbrace}	20	
goto(29, rbrace)	{BLOCK -> rbrace.; BLOCK -> rbrace.else lbrace BLOCK}	21	
goto(29, VDECL)	{STMT -> VDECL.}	22	
goto(29, identifier)	{STMT -> identifier.assign RHS semi}	23	
goto(29, if)	{STMT -> if.lparen COND rparen lbrace BLOCK}	24	
goto(29, while)	{STMT -> while.lparen COND rparen lbrace BLOCK}	25	
goto(29, return)	{RETURN -> return.FACTOR semi}	26	
goto(29, vtype)	{VDECL -> vtype.identifier semi}	27	
goto(32, lbrace)	{BLOCK -> rbrace else lbrace.BLOCK}	43	{BLOCK -> rbrace else lbrace.BLOCK; BLOCK -> .STMT BLOCK; BLOCK -> .RETURN rbrace; BLOCK -> .rbrace; BLOCK -> .rbrace else lbrace BLOCK; STMT -> .VDECL; STMT -> .identifier assign RHS semi; STMT -> .if lparen COND rparen lbrace BLOCK; STMT -> .while lparen COND rparen lbrace BLOCK; RETURN -> .return FACTOR semi; VDECL -> .vtype identifier semi}
goto(33, RHS)	{STMT -> identifier assign RHS.semi}	44	{STMT -> identifier assign RHS.semi}
goto(33, EXPR)	{RHS -> EXPR.}	45	{RHS -> EXPR.}
goto(33, literal)	{RHS -> literal.}	46	{RHS -> literal.}
goto(33, TERM)	{EXPR -> TERM.addsub EXPR; EXPR -> TERM.}	47	{EXPR -> TERM.addsub EXPR; EXPR -> TERM.}
goto(33, FACTOR)	{TERM -> FACTOR.multdiv TERM; TERM -> FACTOR.}	48	{TERM -> FACTOR.multdiv TERM; TERM -> FACTOR.}
goto(33, lparen)	{FACTOR -> lparen.EXPR rparen}	37	
goto(33, identifier)	{FACTOR -> identifier.}	38	
goto(33, integer)	{FACTOR -> integer.}	39	
goto(34, COND)	{STMT -> if lparen COND.rparen lbrace BLOCK}	49	{STMT -> if lparen COND.rparen lbrace BLOCK}
goto(34, FACTOR)	{COND -> FACTOR.comparison FACTOR}	50	{COND -> FACTOR.comparison FACTOR}
goto(34, lparen)	{FACTOR -> lparen.EXPR rparen}	37	
goto(34, identifier)	{FACTOR -> identifier.}	38	
goto(34, integer)	{FACTOR -> integer.}	39	
goto(35, COND)	{STMT -> while lparen COND.rparen lbrace BLOCK}	51	{STMT -> while lparen COND.rparen lbrace BLOCK}
goto(35, FACTOR)	{COND -> FACTOR.comparison FACTOR}	50	
goto(35, lparen)	{FACTOR -> lparen.EXPR rparen}	37	
goto(35, identifier)	{FACTOR -> identifier.}	38	
goto(35, integer)	{FACTOR -> integer.}	39	
goto(36, semi)	{RETURN -> return FACTOR semi.}	52	{RETURN -> return FACTOR semi.}
goto(37, EXPR)	{FACTOR -> lparen EXPR.rparen}	53	{FACTOR -> lparen EXPR.rparen}
goto(37, TERM)	{EXPR -> TERM.addsub EXPR; EXPR -> TERM.}	47	
goto(37, FACTOR)	{TERM -> FACTOR.multdiv TERM; TERM -> FACTOR.}	48	
goto(37, lparen)	{FACTOR -> lparen.EXPR rparen}	37	
goto(37, identifier)	{FACTOR -> identifier.}	38	
goto(37, integer)	{FACTOR -> integer.}	39	
goto(40, semi)	{VDECL -> vtype identifier semi.}	8	
goto(41, MOREARGS)	{MOREARGS -> comma vtype identifier MOREARGS.}	54	{MOREARGS -> comma vtype identifier MOREARGS.}
goto(41, comma)	{MOREARGS -> comma.vtype identifier MOREARGS}	16	
goto(41, rparen)	{MOREARGS -> rparen.lbrace BLOCK}	17	
goto(43, BLOCK)	{BLOCK -> rbrace else lbrace BLOCK.}	55	{BLOCK -> rbrace else lbrace BLOCK.}
goto(43, STMT)	{BLOCK -> STMT.BLOCK}	19	
goto(43, RETURN)	{BLOCK -> RETURN.rbrace}	20	
goto(43, rbrace)	{BLOCK -> rbrace.; BLOCK -> rbrace.else lbrace BLOCK}	21	
goto(43, VDECL)	{STMT -> VDECL.}	22	
goto(43, identifier)	{STMT -> identifier.assign RHS semi}	23	
goto(43, if)	{STMT -> if.lparen COND rparen lbrace BLOCK}	24	
goto(43, while)	{STMT -> while.lparen COND rparen lbrace BLOCK}	25	
goto(43, return)	{RETURN -> return.FACTOR semi}	26	
goto(43, vtype)	{VDECL -> vtype.identifier semi}	27	
goto(44, semi)	{STMT -> identifier assign RHS semi.}	56	{STMT -> identifier assign RHS semi.}
goto(47, addsub)	{EXPR -> TERM addsub.EXPR}	57	{EXPR -> TERM addsub.EXPR; EXPR -> .TERM addsub EXPR; EXPR -> .TERM; TERM -> .FACTOR multdiv TERM; TERM -> .FACTOR; FACTOR -> .lparen EXPR rparen; FACTOR -> .identifier; FACTOR -> .integer}
goto(48, multdiv)	{TERM -> FACTOR multdiv.TERM}	58	{TERM -> FACTOR multdiv.TERM; TERM -> .FACTOR multdiv TERM; TERM -> .FACTOR; FACTOR -> .lparen EXPR rparen; FACTOR -> .identifier; FACTOR -> .integer}
goto(49, rparen)	{STMT -> if lparen COND rparen.lbrace BLOCK}	59	{STMT -> if lparen COND rparen.lbrace BLOCK}
goto(50, comparison)	{COND -> FACTOR comparison.FACTOR}	60	{COND -> FACTOR comparison.FACTOR; FACTOR -> .lparen EXPR rparen; FACTOR -> .identifier; FACTOR -> .integer}
goto(51, rparen)	{STMT -> while lparen COND rparen.lbrace BLOCK}	61	{STMT -> while lparen COND rparen.lbrace BLOCK}
goto(53, rparen)	{FACTOR -> lparen EXPR rparen.}	62	{FACTOR -> lparen EXPR rparen.}
goto(57, EXPR)	{EXPR -> TERM addsub EXPR.}	63	{EXPR -> TERM addsub EXPR.}
goto(57, TERM)	{EXPR -> TERM.addsub EXPR; EXPR -> TERM.}	47	
goto(57, FACTOR)	{TERM -> FACTOR.multdiv TERM; TERM -> FACTOR.}	48	
goto(57, lparen)	{FACTOR -> lparen.EXPR rparen}	37	
goto(57, identifier)	{FACTOR -> identifier.}	38	
goto(57, integer)	{FACTOR -> integer.}	39	
goto(58, TERM)	{TERM -> FACTOR multdiv TERM.}	64	{TERM -> FACTOR multdiv TERM.}
goto(58, FACTOR)	{TERM -> FACTOR.multdiv TERM; TERM -> FACTOR.}	48	
goto(58, lparen)	{FACTOR -> lparen.EXPR rparen}	37	
goto(58, identifier)	{FACTOR -> identifier.}	38	
goto(58, integer)	{FACTOR -> integer.}	39	
goto(59, lbrace)	{STMT -> if lparen COND rparen lbrace.BLOCK}	65	{STMT -> if lparen COND rparen lbrace.BLOCK; BLOCK -> .STMT BLOCK; BLOCK -> .RETURN rbrace; BLOCK -> .rbrace; BLOCK -> .rbrace else lbrace BLOCK; STMT -> .VDECL; STMT -> .identifier assign RHS semi; STMT -> .if lparen COND rparen lbrace BLOCK; STMT -> .while lparen COND rparen lbrace BLOCK; RETURN -> .return FACTOR semi; VDECL -> .vtype identifier semi}
goto(60, FACTOR)	{COND -> FACTOR comparison FACTOR.}	66	{COND -> FACTOR comparison FACTOR.}
goto(60, lparen)	{FACTOR -> lparen.EXPR rparen}	37	
goto(60, identifier)	{FACTOR -> identifier.}	38	
goto(60, integer)	{FACTOR -> integer.}	39	
goto(61, lbrace)	{STMT -> while lparen COND rparen lbrace.BLOCK}	67	{STMT -> while lparen COND rparen lbrace.BLOCK; BLOCK -> .STMT BLOCK; BLOCK -> .RETURN rbrace; BLOCK -> .rbrace; BLOCK -> .rbrace else lbrace BLOCK; STMT -> .VDECL; STMT -> .identifier assign RHS semi; STMT -> .if lparen COND rparen lbrace BLOCK; STMT -> .while lparen COND rparen lbrace BLOCK; RETURN -> .return FACTOR semi; VDECL -> .vtype identifier semi}
goto(65, BLOCK)	{STMT -> if lparen COND rparen lbrace BLOCK.}	68	{STMT -> if lparen COND rparen lbrace BLOCK.}
goto(65, STMT)	{BLOCK -> STMT.BLOCK}	19	
goto(65, RETURN)	{BLOCK -> RETURN.rbrace}	20	
goto(65, rbrace)	{BLOCK -> rbrace.; BLOCK -> rbrace.else lbrace BLOCK}	21	
goto(65, VDECL)	{STMT -> VDECL.}	22	
goto(65, identifier)	{STMT -> identifier.assign RHS semi}	23	
goto(65, if)	{STMT -> if.lparen COND rparen lbrace BLOCK}	24	
goto(65, while)	{STMT -> while.lparen COND rparen lbrace BLOCK}	25	
goto(65, return)	{RETURN -> return.FACTOR semi}	26	
goto(65, vtype)	{VDECL -> vtype.identifier semi}	27	
goto(67, BLOCK)	{STMT -> while lparen COND rparen lbrace BLOCK.}	69	{STMT -> while lparen COND rparen lbrace BLOCK.}
goto(67, STMT)	{BLOCK -> STMT.BLOCK}	19	
goto(67, RETURN)	{BLOCK -> RETURN.rbrace}	20	
goto(67, rbrace)	{BLOCK -> rbrace.; BLOCK -> rbrace.else lbrace BLOCK}	21	
goto(67, VDECL)	{STMT -> VDECL.}	22	
goto(67, identifier)	{STMT -> identifier.assign RHS semi}	23	
goto(67, if)	{STMT -> if.lparen COND rparen lbrace BLOCK}	24	
goto(67, while)	{STMT -> while.lparen COND rparen lbrace BLOCK}	25	
goto(67, return)	{RETURN -> return.FACTOR semi}	26	
goto(67, vtype)	{VDECL -> vtype.identifier semi}	27	

LR table

State	ACTION																		GOTO										
	vtype	identifier	semi	lparen	rparen	lbrace	comma	rbrace	else	assign	if	while	literal	addsub	multdiv	integer	comparison	return	$	CODE'	CODE	VDECL	FDECL	ARG	MOREARGS	BLOCK	STMT	RHS	EXPR	TERM	FACTOR	COND	RETURN
0	s4																			1	2	3											
1																			acc														
2	s4																		r4	5	2	3											
3	s4																		r3	6	2	3											
4		s7																															
5																			r1														
6																			r2														
7			s8	s9																													
8	r5	r5						r5			r5	r5						r5	r5														
9	s11				s12																			10									
10	r6																		r6														
11		s13																															
12						s14																											
13					s17		s16																	15									
14	s27	s23						s21			s24	s25						s26			22				18	19							20
15	r7																		r7														
16	s28																																
17						s29																											
18	r8																		r8														
19	s27	s23						s21			s24	s25						s26			22				30	19							20
20								s31																									
21	r13	r13						r13	s32		r13	r13						r13	r13														
22	r15	r15						r15			r15	r15						r15															
23										s33																							
24				s34																													
25				s35																													
26		s38		s37												s39														36			
27		s40																															
28		s41																															
29	s27	s23						s21			s24	s25						s26			22				42	19							20
30	r11	r11						r11			r11	r11						r11	r11														
31	r12	r12						r12			r12	r12						r12	r12														
32						s43																											
33		s38		s37									s46			s39													44	45	47	48		
34		s38		s37												s39														50	49		
35		s38		s37												s39														50	51		
36			s52																														
37		s38		s37												s39													53	47	48		
38			r26		r26									r26	r26		r26																
39			r27		r27									r27	r27		r27																
40			s8																														
41					s17		s16																	54									
42	r10																		r10														
43	s27	s23						s21			s24	s25						s26			22				55	19							20
44			s56																														
45			r19																														
46			r20																														
47			r22		r22									s57																			
48			r24		r24									r24	s58																		
49					s59																												
50																	s60																
51					s61																												
52								r29																									
53					s62																												
54	r9																		r9														
55	r14	r14						r14			r14	r14						r14	r14														
56	r16	r16						r16			r16	r16						r16															
57		s38		s37												s39													63	47	48		
58		s38		s37												s39														64	48		
59						s65																											
60		s38		s37												s39														66			
61						s67																											
62			r25		r25									r25	r25		r25																
63			r21		r21																												
64			r23		r23									r23																				
65	s27	s23						s21			s24	s25						s26			22				68	19							20
66					r28																												
67	s27	s23						s21			s24	s25						s26			22				69	19							20
68	r17	r17						r17			r17	r17						r17															
69	r18	r18						r18			r18	r18						r18															

Input (tokens): id + id * id

Maximum number of steps: 100

parse

Trace

Step	Stack	Input	Action	Tree
1	0	id + id * id $		