Compiler Term-Project

Implementation of a SLR parser

for a simplified C programming language

[Team 16]

소프트웨어학부 20220881 한지강

소프트웨어학부 20220399 김희정

목차

1. Non-Ambiguous CFG

1) Ambiguous 제거 과정

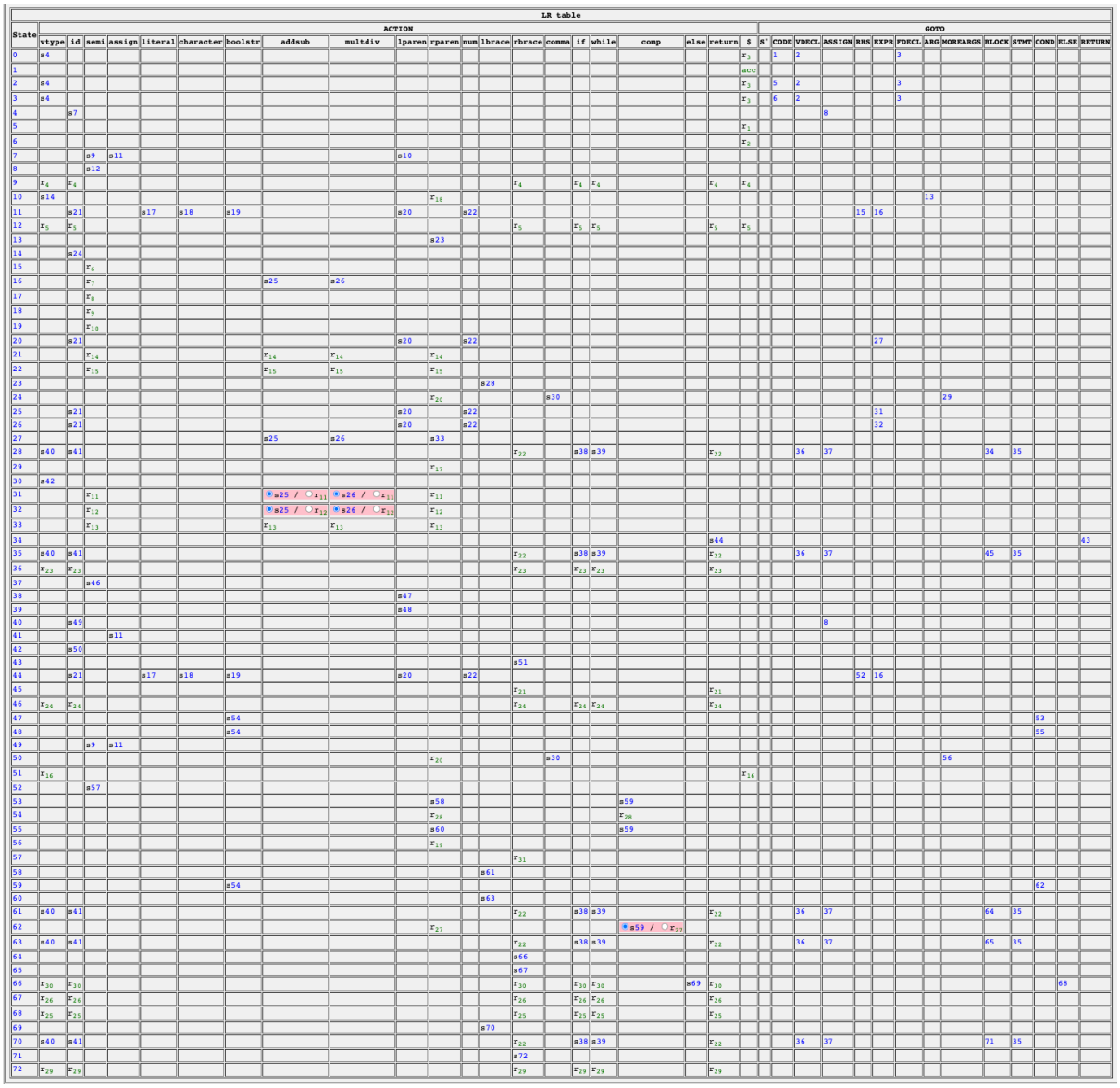
1	CODE → VDECL CODE | FDECL CODE | ϵ
2	VDECL → vtype id semi | vtype ASSIGN semi
3	ASSIGN → id assign RHS
4	RHS → EXPR | literal | character | boolstr
5	EXPR → EXPR addsub EXPR | EXPR multdiv EXPR
6	EXPR → lparen EXPR rparen | id | num
7	FDECL → vtype id lparen ARG rparen lbrace BLOCK RETURN rbrace
8	ARG → vtype id MOREARGS | ϵ
9	MOREARGS → comma vtype id MOREARGS | ϵ
10	BLOCK → STMT BLOCK | ϵ
11	STMT → VDECL | ASSIGN semi
12	STMT → if lparen COND rparen lbrace BLOCK rbrace ELSE
13	STMT → while lparen COND rparen lbrace BLOCK rbrace
14	COND → COND comp COND | boolstr
15	ELSE → else lbrace BLOCK rbrace | ϵ
16	RETURN → return RHS semi

[표 1] 주어진 CFG(Ambiguous 존재)

기존 CFG에서는 [그림 1]에서 보이는 것처럼 Shift-Reduce Conflict가 발생한다. 이 Conflict가 발생하게 되는 원인은 Ambiguous CFG이기 때문인데, 아래 2개의 CFG로 인해 multiple parse tree가 생성된다.

05: EXPR → EXPR addsub EXPR | EXPR multdiv EXPR

14: COND → COND comp COND | boolstr



LR table

State	ACTION																			GOTO														
	vtype	id	semi	assign	literal	character	boolstr	addsub	multdiv	lparen	rparen	num	lbrace	rbrace	comma	if	while	comp	else	return	$	S'	CODE	VDECL	ASSIGN	RHS	EXPR	FDECL	ARG	MOREARGS	BLOCK	STMT	COND	ELSE	RETURN
0	s4																				r3		1	2				3							
1																					acc														
2	s4																				r3		5	2				3							
3	s4																				r3		6	2				3							
4		s7																							8										
5																					r1														
6																					r2														
7			s9	s11						s10																									
8			s12																																
9	r4	r4												r4		r4	r4			r4	r4														
10	s14										r18																		13						
11		s21			s17	s18	s19			s20		s22														15	16								
12	r5	r5												r5		r5	r5			r5	r5														
13											s23																								
14		s24																																	
15			r6																																
16			r7					s25	s26																										
17			r8																																
18			r9																																
19			r10																																
20		s21								s20		s22															27								
21			r14					r14	r14		r14																								
22			r15					r15	r15		r15																								
23													s28																						
24											r20				s30														29						
25		s21								s20		s22															31								
26		s21								s20		s22															32								
27								s25	s26		s33																								
28	s40	s41												r22		s38	s39			r22				36	37							34	35			
29											r17																								
30	s42																																		
31			r11					s25 / r11	s26 / r11		r11																										
32			r12					s25 / r12	s26 / r12		r12																										
33			r13					r13	r13		r13																									
34																				s44															s43
35	s40	s41												r22		s38	s39			r22				36	37							45	35			
36	r23	r23												r23		r23	r23			r23																
37			s46																																	
38										s47																									
39										s48																									
40		s49																							8										
41				s11																															
42		s50																																	
43														r51																					
44		s21			s17	s18	s19			s20		s22														52	16								
45														r21						r21															
46	r24	r24												r24		r24	r24			r24															
47							s54																											53	
48							s54																											55	
49			s9	s11																															
50											r20				s30														56						
51	r16																				r16														
52			s57																																
53											s58							s59																	
54											r28							r28																	
55											s60							s59																	
56											r19																								
57														r31																					
58													s61																						
59							s54																											62	
60													s63																						
61	s40	s41												r22		s38	s39			r22				36	37							64	35			
62											r27							s59 / r27																	
63	s40	s41												r22		s38	s39			r22				36	37							65	35			
64														s66																					
65														s67																					
66	r30	r30												r30		r30	r30		s69	r30														68	
67	r26	r26												r26		r26	r26			r26															
68	r25	r25												r25		r25	r25			r25															
69													s70																						
70	s40	s41												r22		s38	s39			r22				36	37							71	35			
71														s72																					
72	r29	r29												r29		r29	r29			r29															

[그림 1] 기존 CFG에 대한 SLR parsing table

Ambiguity를 제거하기 위해 정해져 있는 방법은 없지만, 우선순위를 정함으로써 제거할 수 있다. 우리 팀은 새로운 non-terminal EXPR', EXPR'', COND'을 만들고 이들의 우선순위에 따라 [표 2]의 Non-Ambious CFG를 만들었다.

2) Ambiguous 를 제거한 최종 CFG

1	CODE → VDECL CODE | FDECL CODE | ϵ
2	VDECL → vtype id semi | vtype ASSIGN semi
3	ASSIGN → id assign RHS
4	RHS → EXPR | literal | character | boolstr
5	EXPR → EXPR addsub EXPR′ | EXPR′
6	EXPR′ → EXPR′ multdiv EXPR′′ | EXPR′′
7	EXPR′′ → lparen EXPR rparen | id | num
8	FDECL → vtype id lparen ARG rparen lbrace BLOCK RETURN rbrace
9	ARG → vtype id MOREARGS | ϵ
10	MOREARGS → comma vtype id MOREARGS | ϵ
11	BLOCK → STMT BLOCK | ϵ
12	STMT → VDECL | ASSIGN semi
13	STMT → if lparen COND rparen lbrace BLOCK rbrace ELSE
14	STMT → while lparen COND rparen lbrace BLOCK rbrace
15	COND → COND comp COND′ | COND′
16	COND′ → boolstr
17	ELSE → else lbrace BLOCK rbrace | ϵ
18	RETURN → return RHS semi

[표 2] 수정한 CFG(Ambiguous 제거)

2. SLR parsing table

LR table

ACTION

State	vtype	id	semi	assign	literal	character	boolstr	addsub	multdiv	lparen	rparen	num	lbrace	rbrace	comma	if	while	comp	else	return	$
0	s4																				r_3
1																					acc
2	s4																				r_3
3	s4																				r_3
4		s7																			
5																					r_1
6																					r_2
7			s9	s11						s10											
8			s12																		
9	r_4	r_4												r_4		r_4	r_4			r_4	r_4
10	s14										r_{20}										
11		s23			s17	s18	s19			s22		s24									
12	r_5	r_5												r_5		r_5	r_5			r_5	r_5
13											s25										
14		s26																			
15			r_6																		
16			r_7					s27													
17			r_8																		
18			r_9																		
19			r_{10}																		
20			r_{12}					r_{12}	s28		r_{12}										
21			r_{14}					r_{14}	r_{14}		r_{14}										
22		s23								s22		s24									
23			r_{16}					r_{16}	r_{16}		r_{16}										
24			r_{17}					r_{17}	r_{17}		r_{17}										
25													s30								
26											r_{22}				s32						
27		s23								s22		s24									
28		s23								s22		s24									
29								s27			s35										
30	s42	s43												r_{24}		s40	s41			r_{24}	
31											r_{19}										
32	s44																				
33			r_{11}					r_{11}	s28		r_{11}										
34			r_{13}					r_{13}	r_{13}		r_{13}										
35			r_{15}					r_{15}	r_{15}		r_{15}										
36																				s46	
37	s42	s43												r_{24}		s40	s41			r_{24}	
38	r_{25}	r_{25}												r_{25}		r_{25}	r_{25}			r_{25}	
39			s48																		
40										s49											
41										s50											
42		s51																			
43				s11																	
44		s52																			
45														s53							
46		s23			s17	s18	s19			s22		s24									
47														r_{23}						r_{23}	
48	r_{26}	r_{26}												r_{26}		r_{26}	r_{26}			r_{26}	
49							s57														
50							s57														
51			s9	s11																	
52											r_{22}				s32						
53	r_{18}																				r_{18}
54			s60																		
55											s61							s62			
56											r_{30}							r_{30}			
57											r_{31}							r_{31}			
58											s63							s62			
59											r_{21}										
60														r_{34}							
61													s64								
62							s57														
63													s66								
64	s42	s43												r_{24}		s40	s41			r_{24}	
65											r_{29}							r_{29}			
66	s42	s43												r_{24}		s40	s41			r_{24}	
67														s69							
68														s70							
69	r_{33}	r_{33}												r_{33}		r_{33}	r_{33}		s72	r_{33}	
70	r_{28}	r_{28}												r_{28}		r_{28}	r_{28}			r_{28}	
71	r_{27}	r_{27}												r_{27}		r_{27}	r_{27}			r_{27}	
72													s73								
73	s42	s43												r_{24}		s40	s41			r_{24}	
74														s75							
75	r_{32}	r_{32}												r_{32}		r_{32}	r_{32}			r_{32}	

GOTO

State	S'	CODE	VDECL	ASSIGN	RHS	EXPR	EXPR'	EXPR''	FDECL	ARG	MOREARGS	BLOCK	STMT	COND	COND'	ELSE	RETURN
0		1	2						3								
1																	
2		5	2						3								
3		6	2						3								
4				8													
5																	
6																	
7																	
8																	
9																	
10										13							
11					15	16	20	21									
12																	
13																	
14																	
15																	
16																	
17																	
18																	
19																	
20																	
21																	
22						29	20	21									
23																	
24																	
25																	
26											31						
27							33	21									
28								34									
29																	
30			38	39								36	37				
31																	
32																	
33																	
34																	
35																	
36																	45
37			38	39								47	37				
38																	
39																	
40																	
41																	
42				8													
43																	
44																	
45																	
46					54	16	20	21									
47																	
48																	
49														55	56		
50														58	56		
51																	
52											59						
53																	
54																	
55																	
56																	
57																	
58																	
59																	
60																	
61																	
62															65		
63																	
64			38	39								67	37				
65																	
66			38	39								68	37				
67																	
68																	
69																71	
70																	
71																	
72																	
73			38	39								74	37				
74																	
75																	

[그림 2] Non-Ambiguous로 만든 SLR parsing table

3. 코드 동작 설명

1) SLR_grammar 및 SLR_parsing_table

Non-Ambiguous CFG 를 SLR grammar 형식으로 바꾸어 [그림 3]과 같이 list 로 구현하였다. list 를 선택한 이유는 stack 에 있는 state 를 pop 하는 과정에서 A->α에 대해 |α|만큼 pop 하기 때문이다.

```
# SLR grammar
SLR_grammar = [
    ("S'", ["CODE"]),
    ("CODE", ["VDECL", "CODE"]),
    ("CODE", ["FDECL", "CODE"]),
    ("CODE", ["epsilon"]),
    ("VDECL", ["vtype", "id", "semi"]),
    ("VDECL", ["vtype", "ASSIGN", "semi"]),
    ("ASSIGN", ["id", "assign", "RHS"]),
    ("RHS", ["EXPR"]),
    ("RHS", ["literal"]),
    ("RHS", ["character"]),
    ("RHS", ["boolstr"]),
    ("EXPR", ["EXPR", "addsub", "EXPR'"]),
    ("EXPR", ["EXPR'"]),
    ("EXPR'", ["EXPR'", "multdiv", "EXPR''"]),
    ("EXPR'", ["EXPR''"]),
    ("EXPR''", ["lparen", "EXPR", "rparen"]),
    ("EXPR''", ["id"]),
    ("EXPR''", ["num"]),
    ("FDECL", ["vtype", "id", "lparen", "ARG", "rparen", "lbrace", "BLOCK", "RETURN", "rbrace"]),
    ("ARG", ["vtype", "id", "MOREARGS"]),
    ("ARG", ["epsilon"]),
    ("MOREARGS", ["comma", "vtype", "id", "MOREARGS"]),
    ("MOREARGS", ["epsilon"]),
    ("BLOCK", ["STMT", "BLOCK"]),
    ("BLOCK", ["epsilon"]),
    ("STMT", ["VDECL"]),
    ("STMT", ["ASSIGN", "semi"]),
    ("STMT", ["if", "lparen", "COND", "rparen", "lbrace", "BLOCK", "rbrace", "ELSE"]),
    ("STMT", ["while", "lparen", "COND", "rparen", "lbrace", "BLOCK", "rbrace"]),
    ("COND", ["COND", "comp", "COND'"]),
    ("COND", ["COND'"]),
    ("COND'", ["boolstr"]),
    ("ELSE", ["else", "lbrace", "BLOCK", "rbrace"]),
    ("ELSE", ["epsilon"]),
    ("RETURN", ["return", "RHS", "semi"])
]
```

[그림 3] SLR grammar

SLR parsing table은 [그림 4]와 같이 Dictonary로 구현하였다. SLR parsing은 stack의 top에 있는 state와 right substring의 leftmost terminal을 읽어 해당하는 action을 수행함으로써 이루어진다. 이를 위해 state를 키로 하고, Dictionary를 값으로 갖는 Dictionary를 만들고, 내부 Dictionary는 token을 키로 하고 해당하는 action을 값으로 갖도록 구성하였다.

```
# SLR parsing table
SLR_parsing_table = {
    0: {'vtype': 's4', '$': 'r3', 'CODE': 1, 'VDECL': 2, 'FDECL': 3},
    1: {'$': 'acc'},
    2: {'vtype': 's4', '$': 'r3', 'CODE': 5, 'VDECL': 2, 'FDECL': 3},
    3: {'vtype': 's4', '$': 'r3', 'CODE': 6, 'VDECL': 2, 'FDECL': 3},
    4: {'id': 's7', 'ASSIGN': 8},
    5: {'$': 'r1'},
    6: {'$': 'r2'},
    7: {'semi': 's9', 'assign': 's11', 'lparen': 's10'},
    8: {'semi': 's12'},
    9: {'vtype': 'r4', 'id': 'r4', 'rbrace': 'r4', 'if': 'r4', 'while': 'r4', 'return': 'r4', '$': 'r4'},
    10: {'vtype': 's14', 'rparen': 'r20', 'ARG': 13},
    11: {'id': 's23', 'literal': 's17', 'character': 's18', 'boolstr': 's19', 'lparen': 's22', 'num': 's24', 'RHS': 15, 'EXPR': 16, 'EXPR'': 20, 'EXPR''': 21},
    12: {'vtype': 'r5', 'id': 'r5', 'rbrace': 'r5', 'if': 'r5', 'while': 'r5', 'return': 'r5', '$': 'r5'},
    13: {'rparen': 's25'},
    14: {'id': 's26'},
    15: {'semi': 'r6'},
    16: {'semi': 'r7', 'addsub': 's27'},
    17: {'semi': 'r8'},
    18: {'semi': 'r9'},
    19: {'semi': 'r10'},
    20: {'semi': 'r12', 'addsub': 'r12', 'multdiv': 's28', 'rparen': 'r12'},
    21: {'semi': 'r14', 'addsub': 'r14', 'multdiv': 'r14', 'rparen': 'r14'},
    22: {'id': 's23', 'lparen': 's22', 'num': 's24', 'EXPR': 29, 'EXPR'': 20, 'EXPR''': 21},
    23: {'semi': 'r16', 'addsub': 'r16', 'multdiv': 'r16', 'rparen': 'r16'},
    24: {'semi': 'r17', 'addsub': 'r17', 'multdiv': 'r17', 'rparen': 'r17'},
    25: {'lbrace': 's30'},
    26: {'rparen': 'r22', 'comma': 's32', 'MOREARGS': 31},
    27: {'id': 's23', 'lparen': 's22', 'num': 's24', 'EXPR'': 33, 'EXPR''': 21},
    28: {'id': 's23', 'lparen': 's22', 'num': 's24', 'EXPR''': 34},
    29: {'addsub': 's27', 'rparen': 's35'},
    30: {'vtype': 's42', 'id': 's43', 'rbrace': 'r24', 'if': 's40', 'while': 's41', 'return': 'r24', 'VDECL': 38, 'ASSIGN': 39, 'BLOCK': 36, 'STMT': 37},
    31: {'rparen': 'r19'},
    32: {'vtype': 's44'},
    33: {'semi': 'r11', 'addsub': 'r11', 'multdiv': 's28', 'rparen': 'r11'},
    34: {'semi': 'r13', 'addsub': 'r13', 'multdiv': 'r13', 'rparen': 'r13'},
    35: {'semi': 'r15', 'addsub': 'r15', 'multdiv': 'r15', 'rparen': 'r15'},
    36: {'return': 's46', 'RETURN': 45},
    37: {'vtype': 's42', 'id': 's43', 'rbrace': 'r24', 'if': 's40', 'while': 's41', 'return': 'r24', 'VDECL': 38, 'ASSIGN': 39, 'BLOCK': 47, 'STMT': 37},
    38: {'vtype': 'r25', 'id': 'r25', 'rbrace': 'r25', 'if': 'r25', 'while': 'r25', 'return': 'r25'},
    39: {'semi': 's48'},
    40: {'lparen': 's49'},
    41: {'lparen': 's50'},
    42: {'id': 's51', 'ASSIGN': 8},
    43: {'assign': 's11'},
    44: {'id': 's52'},
    45: {'rbrace': 's53'},
    46: {'id': 's23', 'literal': 's17', 'character': 's18', 'boolstr': 's19', 'lparen': 's22', 'num': 's24', 'RHS': 54, 'EXPR': 16, 'EXPR'': 20, 'EXPR''': 21},
    47: {'rbrace': 'r23', 'return': 'r23'},
    48: {'vtype': 'r26', 'id': 'r26', 'rbrace': 'r26', 'if': 'r26', 'while': 'r26', 'return': 'r26'},
    49: {'boolstr': 's57', 'COND': 55, 'COND'': 56},
    50: {'boolstr': 's57', 'COND': 58, 'COND'': 56},
    51: {'semi': 's9', 'assign': 's11'},
    52: {'rparen': 'r22', 'comma': 's32', 'MOREARGS': 59},
    53: {'vtype': 'r18', '$': 'r18'},
    54: {'semi': 's60'},
    55: {'rparen': 's61', 'comp': 's62', 'COND': 62, 'COND'': 57},
    56: {'rparen': 'r30', 'comp': 'r30'},
    57: {'rparen': 'r31', 'comp': 'r31'},
    58: {'rparen': 's63', 'comp': 's62', 'COND': 62, 'COND'': 57},
    59: {'rparen': 'r21'},
    60: {'rbrace': 'r34'},
    61: {'lbrace': 's64'},
    62: {'boolstr': 's57', 'COND'': 65},
    63: {'lbrace': 's66'},
    64: {'vtype': 's42', 'id': 's43', 'rbrace': 'r24', 'if': 's40', 'while': 's41', 'return': 'r24', 'VDECL': 38, 'ASSIGN': 39, 'BLOCK': 67, 'STMT': 37},
    65: {'rparen': 'r29', 'return': 'r29'},
    66: {'vtype': 's42', 'id': 's43', 'rbrace': 'r24', 'if': 's40', 'while': 's41', 'return': 'r24', 'VDECL': 38, 'ASSIGN': 39, 'BLOCK': 68, 'STMT': 37},
    67: {'rbrace': 's69'},
    68: {'rbrace': 's70'},
    69: {'vtype': 'r33', 'id': 'r33', 'rbrace': 'r33', 'if': 'r33', 'while': 'r33', 'else': 's72', 'return': 'r33', 'ELSE': 71},
    70: {'vtype': 'r28', 'id': 'r28', 'rbrace': 'r28', 'if': 'r28', 'while': 'r28', 'return': 'r28'},
    71: {'vtype': 'r27', 'id': 'r27', 'rbrace': 'r27', 'if': 'r27', 'while': 'r27', 'return': 'r27'},
    72: {'lbrace': 's73'},
    73: {'vtype': 's42', 'id': 's43', 'rbrace': 'r24', 'if': 's40', 'while': 's41', 'return': 'r24', 'VDECL': 38, 'ASSIGN': 39, 'BLOCK': 74, 'STMT': 37},
    74: {'rbrace': 's75'},
    75: {'vtype': 'r32', 'id': 'r32', 'rbrace': 'r32', 'if': 'r32', 'while': 'r32', 'return': 'r32'}
}
```

[그림 4] SLR parsing table

2) 실행 부분

먼저 input file을 command로 받아 처리하고, 읽기 모드로 열도록 구현하였다. input file을 통해 입력받은 input tokens를 초기화하는 과정을 거쳐 parsing한다. 그 결과 Accept되면 parse tree를 출력하고, Reject되면 error message를 출력하도록 하였다.

```
# input file 입력으로 받아서 처리
input_file = sys.argv[1]
with open(input_file, 'r') as file:
    input = file.read()

# parser 초기화
parser = SLRParser(SLR_parsing_table, SLR_grammar)
input_tokens = input.split()
parser.initialize(input_tokens)

# parser 실행 및 결과 출력
if parser.parse():
    print("\n-----Generate Parse Tree-----")
    parser.print_parse_tree(parser.parse_tree)
else:
    print("Reject!\n")
    print(parser.get_error_message())
```

[그림 5] 실행 부분

3) ParseNode 클래스

parse tree를 생성하기 위한 클래스이다. parsing과정에서 나오는 node(non-terminal과 terminal)을 저장해둔다.

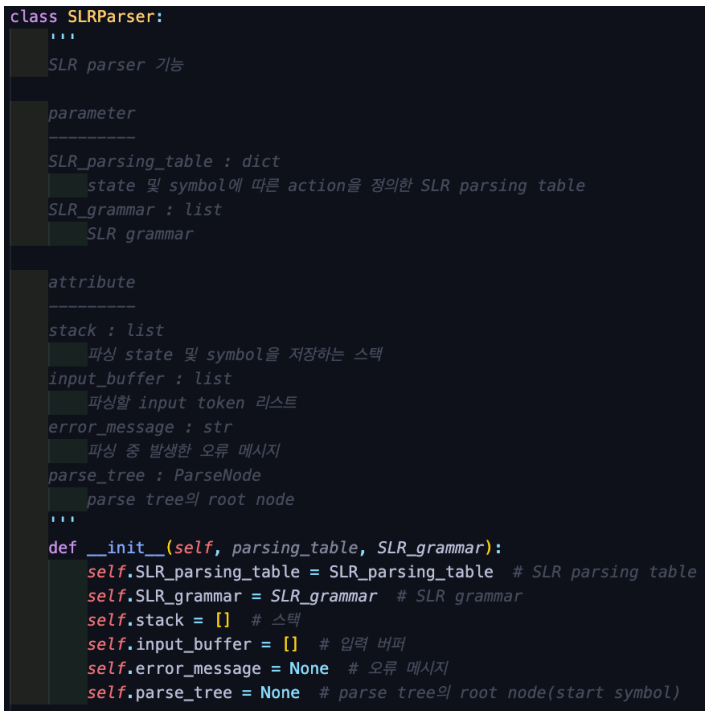
```
class ParseNode:
    '''
    parse tree의 각 node(non-terminal & terminal) 표현
    '''
    def __init__(self, symbol, children=None):
        self.symbol = symbol
        self.children = children if children else []
```

[그림 6] ParseNode 클래스

4) SLRParser 클래스

i) __init__ 함수

SLRParser 클래스에서 객체를 초기화하는 함수이다. LIFO 구조를 가진 stack 을 이용하기 위해 [그림 7]과 같이 구성하였다.

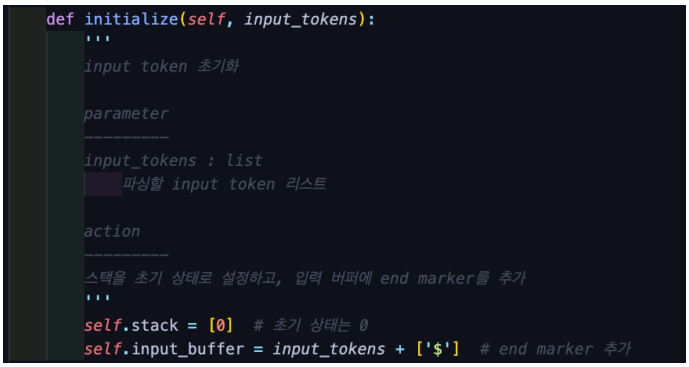


```
class SLRParser:
    '''
    SLR parser 기능

    parameter
    ---------
    SLR_parsing_table : dict
        state 및 symbol에 따른 action을 정의한 SLR parsing table
    SLR_grammar : list
        SLR grammar

    attribute
    ---------
    stack : list
        파싱 state 및 symbol을 저장하는 스택
    input_buffer : list
        파싱할 input token 리스트
    error_message : str
        파싱 중 발생한 오류 메시지
    parse_tree : ParseNode
        parse tree의 root node
    '''
    def __init__(self, parsing_table, SLR_grammar):
        self.SLR_parsing_table = SLR_parsing_table  # SLR parsing table
        self.SLR_grammar = SLR_grammar  # SLR grammar
        self.stack = []  # 스택
        self.input_buffer = []  # 입력 버퍼
        self.error_message = None  # 오류 메시지
        self.parse_tree = None  # parse tree의 root node(start symbol)
```

[그림 7] __init__ 함수

ii) initialize 함수

SLRparser를 초기화하는 부분이다. stack의 top은 0으로 설정하고, input token의 끝에 end marker를 추가하여 parsing 과정의 끝을 인식할 수 있도록 한다.



```
def initialize(self, input_tokens):
    '''
    input token 초기화

    parameter
    ---------
    input_tokens : list
        파싱할 input token 리스트

    action
    ---------
    스택을 초기 상태로 설정하고, 입력 버퍼에 end marker를 추가
    '''
    self.stack = [0]  # 초기 상태는 0
    self.input_buffer = input_tokens + ['$']  # end marker 추가
```

[그림 8] initialize 함수

iii) parse 함수

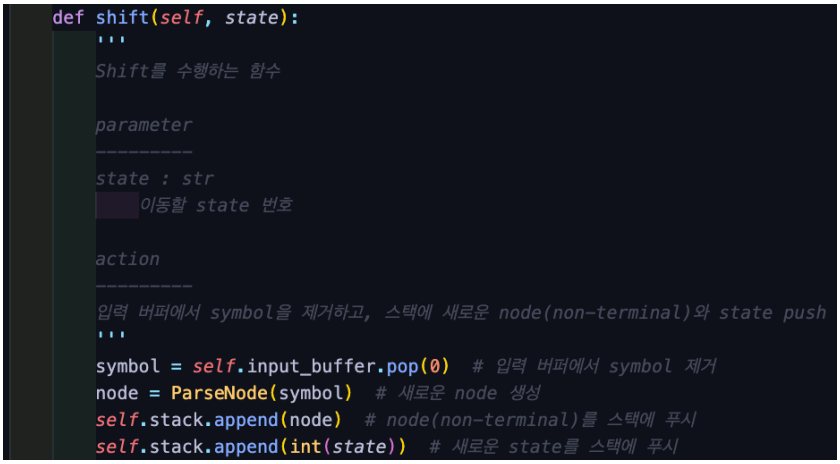
parsing 의 과정이 나타나있는 함수이다. stack 에 저장된 state 를 하나 pop 하여, input buffer 에 저장된 첫번째 symbol 을 가져온다. 그 후, state 와 symbol 에 해당하는 action 을 취하고, 이 과정을 반복한다. action 을 처리하는 과정은 [그림 8]과 같이 비어있을 경우 error 발생, s 로 시작하면 shift, r 로 시작하면 reduce, acc 인 경우 과정을 멈춘다.

```
def parse(self):
    '''
    입력을 파싱하여 syntax analyze를 수행하는 함수

    return
    ---------
    bool
        파싱 성공 시 True, 실패 시 False를 반환
    '''
    while True:
        state = self.stack[-1]  # 스택의 top에 있는 state를 가져옴
        symbol = self.input_buffer[0]  # 입력 버퍼의 첫 번째 symbol을 가져옴
        action = self.SLR_parsing_table[state].get(symbol)  # 현재 state와 symbol로 action 조회

        # action이 None인 경우 error 처리
        if action is None:
            self.error_message = f"Syntax Error: state {state}에서 symbol '{symbol}'에 대한 action 없음"
            return False

        # Shift인 경우
        if action.startswith('s'):
            self.shift(action[1:])
        # Reduce인 경우
        elif action.startswith('r'):
            if not self.reduce(action[1:], state, symbol):
                return False
        # Accept인 경우
        elif action == 'acc':
            print("Accept!")
            self.parse_tree = self.stack[1]  # parse tree의 root node(start symbol) 설정
            return True
        # 기타 예외 처리
        else:
            return False
```

[그림 9] parse 함수

iv) shift 함수

parse 함수에서 action 이 s 로 시작할 때 실행되는 함수이다. input buffer 에서 symbol 을 pop 하고, ParseNode 클래스로 새로운 node(non-terminal 과 terminal)을 생성한다. 이렇게 나온 node 를 stack 에 push 하고 shift 한 state 를 push 한다.



```
def shift(self, state):
    '''
    Shift를 수행하는 함수

    parameter
    ---------
    state : str
        이동할 state 번호

    action
    ----------
    입력 버퍼에서 symbol을 제거하고, 스택에 새로운 node(non-terminal)와 state push
    '''
    symbol = self.input_buffer.pop(0)  # 입력 버퍼에서 symbol 제거
    node = ParseNode(symbol)  # 새로운 node 생성
    self.stack.append(node)  # node(non-terminal)를 스택에 푸시
    self.stack.append(int(state))  # 새로운 state를 스택에 푸시
```

[그림 10] shift 함수

v) reduce 함수

parse 함수에서 action 이 r 로 시작할 때 실행되는 함수이다. r 뒤의 SLR grammar 규칙을 따라 A->α에서 |α|만큼 stack 에서 pop 하고 자식 node 로 추가한다. 새로운 node 를 생성하고, stack 에 node 를 push 한다. 그 후, next input 을 읽어 stack 에 push 한다.

```
def reduce(self, rule_number, current_state, current_symbol):
    '''
    Reduce를 수행하는 함수

    parameter
    ---------
    rule_number : str
        적용할 grammar 번호
    current_state : int
        현재 state
    current_symbol : str
        현재 symbol

    return
    ---------
    bool
        Reduce 성공 시 True, 실패 시 False를 반환

    action
    ---------
    grammar 규칙에 따라 스택에서 state와 symbol를 제거하고 새로운 node(non-terminal)를 생성하여 push
    '''
    rule_number = int(rule_number)
    lhs, rhs = self.SLR_grammar[rule_number]  # reduce 번호에 해당하는 rule을 가져옴

    children = []
    if rhs != ['epsilon']:
        for _ in range(len(rhs)):
            self.stack.pop()  # state pop
            children.insert(0, self.stack.pop())  # symbol 제거 및 자식 리스트에 추가

    node = ParseNode(lhs, children)  # 새로운 node 생성
    self.stack.append(node)  # node를 스택에 푸시

    state = self.stack[-2]  # 스택의 새로운 최상위 state를 가져옴
    new_state = self.SLR_parsing_table[state].get(lhs)  # 새로운 state 조회

    if new_state is None:
        return False

    self.stack.append(new_state)  # 새로운 state를 스택에 푸시
    return True
```

[그림 11] reduce 함수

vi) get_error_message 함수

객체에 저장된 error message 를 가져오는 함수이다.

```
    def get_error_message(self):
        return self.error_message
```

[그림 12] get_error_message 함수

vii) print_parse_tree 함수

parse tree 를 출력하는 함수이다. parsing 과정에서 저장된 node 들을 불러와 재귀적으로 parse tree 를 만든다.

```
    def print_parse_tree(self, node, level=0):
        '''
        parse tree를 출력하는 함수

        parameter
        ---------
        node : ParseNode
            출력할 parse tree의 root node(start symbol)
        level : int
            현재 출력 중인 tree의 깊이 (기본값: 0)

        action
        ---------
        parse tree의 각 node를 재귀적으로 출력합니다.
        '''
        print('  ' * (level * 2) + str(node.symbol))  # 현재 node 출력
        for child in node.children:  # 자식 node에 대해 재귀적으로 출력
            self.print_parse_tree(child, level + 1)
```

[그림 13] print_parse_tree 함수

4. 테스트 결과

1) Accept 되는 경우

i) accept_input1.txt

```
kimheejung ~/study/compiler
 cat accept_input1.txt
vtype id semi vtype id lparen rparen lbrace if lparen boolstr comp boolstr rparen lbrace rbrace return literal semi rbrace%
```

```
kimheejung ~/study/compiler
 python SLRparser.py accept_input1.txt
Accept!

-----Generate Parse Tree-----
CODE
  VDECL
    vtype
    id
    semi
  CODE
    FDECL
      vtype
      id
      lparen
      ARG
      rparen
      lbrace
      BLOCK
        STMT
          if
          lparen
          COND
            COND
              COND'
                boolstr
            comp
            COND'
              boolstr
          rparen
          lbrace
          BLOCK
          rbrace
          ELSE
        BLOCK
      RETURN
        return
        RHS
          literal
        semi
      rbrace
    CODE
```

```
int a;
int func()
{
  if (b < c)
  {
  }
  return "hello";
}
```

ii) accept_input2.txt

```
 cat accept_input2.txt
vtype id lparen vtype id comma vtype id rparen lbrace while lparen boolstr comp boolstr rparen lbrace if lparen boolstr comp
 boolstr rparen lbrace vtype id semi id assign num addsub num multdiv lparen num addsub num rparen semi vtype id semi rbrace
 else lbrace vtype id semi rbrace rbrace id assign literal semi return character semi rbrace vtype id semi vtype id assign b
oolstr semi
```

```
kimheejung ~/study/compiler
 python SLRparser.py accept_input2.txt
Accept!

-----Generate Parse Tree-----
CODE
  FDECL
    vtype
    id
    lparen
    ARG
      vtype
      id
      MOREARGS
        comma
        vtype
        id
        MOREARGS
    rparen
    lbrace
    BLOCK
      STMT
        while
        lparen
        COND
          COND
            COND'
              boolstr
          comp
          COND'
            boolstr
        rparen
        lbrace
        BLOCK
          STMT
            if
            lparen
            COND
              COND
                COND'
                  boolstr
              comp
              COND'
                boolstr
            rparen
            lbrace
            BLOCK
              STMT
                VDECL
                  vtype
                  id
                  semi
              BLOCK
                STMT
                  ASSIGN
                    id
                    assign
                    RHS
                      EXPR
                        EXPR
                          EXPR'
                            EXPR''
                              num
                        addsub
                        EXPR'
                          EXPR'
                            EXPR''
                              num
                          multdiv
                          EXPR''
                            lparen
                            EXPR
                              EXPR
                                EXPR'
                                  EXPR''
                                    num
                              addsub
                              EXPR'
                                EXPR''
                                  num
                            rparen
                    semi
                BLOCK
                  STMT
                    VDECL
                      vtype
                      id
                      semi
                  BLOCK
            rbrace
            ELSE
              else
              lbrace
              BLOCK
                STMT
                  VDECL
                    vtype
                    id
                    semi
                BLOCK
            rbrace
        BLOCK
      rbrace
    BLOCK
      STMT
        ASSIGN
          id
          assign
          RHS
            literal
        semi
    BLOCK
  RETURN
    return
    RHS
      character
    semi
  rbrace
CODE
  VDECL
    vtype
    id
    semi
  CODE
    VDECL
      vtype
      ASSIGN
        id
        assign
        RHS
          boolstr
      semi
    CODE
```

```
int main(int x, int y) {
    while (a < b) {
        if (c > d) {
            int e;
            x = 1 + 2 * (3 + 4);
            int f;
        } else {
            int g;
        }
    }
    y = "hello";
    return 'c';
}
int h;
int i = true;
```

2) Reject되는 경우

i) reject_input1.txt

```
kimheejung ~/study/compiler
cat reject_input1.txt
vtype id semi vtype id lparen rparen lbrace if lparen boolstr comp boolstr rparen lbrace return literal semi rbrace
```

```
kimheejung ~/study/compiler
python SLRparser.py reject_input1.txt
Reject!

Syntax Error: state 67에서 symbol 'return'에 대한 action 없음
```

```
int a;
int func()
{
  if (b < c)
  {
  }없음
  return "hello";
}
```

해당 input에 대한 결과로 나온 error message처럼 SLR parsing table에 67번 state에 대해 return에 대한 action이 없음을 확인할 수 있다.

ii) reject_input2.txt

```
kimheejung ~/study/compiler
cat reject_input2.txt
vtype id lparen vtype id comma vtype id rparen lbrace while lparen boolstr comp boolstr rparen lbrace if lparen boolstr
 comp boolstr rparen lbrace vtype id semi id assign num addsub num multdiv lparen num addsub num rparen semi vtype id s
emi rbrace else lbrace vtype id rbrace rbrace id assign literal semi return character semi rbrace vtype id semi vtype i
d assign boolstr semi%
```

```
kimheejung ~/study/compiler
python SLRparser.py reject_input2.txt
Reject!

Syntax Error: state 51에서 symbol 'rbrace'에 대한 action 없음
```

```
int main(int x, int y) {
    while (a < b) {
        if (c > d) {
            int e;
            x = 1 + 2 * (3 + 4);
            int f;
        } else {
            int g;없음
        }
    }
    y = "hello";
    return 'c';
}
int h;
int i = true;
```

해당 input에 대한 결과로 나온 error message처럼 SLR parsing table에 51번 state에 대해 rbrace에 대한 action이 없음을 확인할 수 있다.