**Compiler Term Project: SLR Parser**

|  |  |
| --- | --- |
| 과목명 | 컴파일러 |
| 교수명 | 김효수 |
| 학번 | 20192958 / 20192971 |
| 이름 | 설지환 / 최동욱 |
| 학부 | 소프트웨어학부 |

**중앙대학교**

**소프트웨어대학**

**Simplified Java의 SLR Parser 구현**

**실행 방법:**

Linux / macOS 환경의 경우 pandas 패키지, anytree 라이브러리을 설치해 주어야 한다. pip를 이용하여 패키지를 설치하며, 해당 설치 명령어는 다음과 같다.

pip install pandas / pip install anytree

Python이 인터프리터 방식이기 때문에 바이너리 파일은 따로 생성하지 않았으며, 다음 명령어로 SLR Parser를 실행할 수 있다.

python simple\_java\_parser.py / python simple\_java\_parser.py input.txt

전자의 경우 simple\_java\_parser.py 내에 정의된 기본 sequence에 대해 파싱을 진행하며, 후자의 경우 input.txt 내에 정의된 sequence에 대해 파싱을 진행한다. 파싱을 진행하며 accept되는 경우에는 생성된 Parsing tree를 console에 출력해 줌과 동시에 output.txt라는 파일 내에, 존재하지 않는다면 simple\_java\_parser.py와 동일한 디렉토리 내에 생성해 준 후 저장하며, reject되는 경우에는 파싱 과정에서 최초로 에러가 발생한 token을 기본 형식에 맞추어 동일하게 console과 file에 출력한다. 이 때, argument value의 개수가 1 또는 2인 경우가 아니면 NameError가 발생하고, input 파일이 존재하지 않는 경우 FileNotFoundError가 발생하기 때문에 해당 에러에 대해 ‘Invalid argument’ 출력이라는 예외처리를 수행해 준다.

**수정된 CFG와 Parsing Table:**

**(a) CFG**

S -> 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 -> TERM EXPR2

EXPR2 -> addsub TERM EXPR2

EXPR2 -> ''

TERM -> FACTOR TERM2

TERM2 -> multdiv FACTOR TERM2

TERM2 -> ''

FACTOR -> lparen EXPR rparen

FACTOR -> id

FACTOR -> 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

COND -> COND2 comp COND

COND -> boolstr

COND2 -> boolstr

ELSE -> else lbrace BLOCK rbrace

ELSE -> ''

RETURN -> return RHS semi

CDECL -> class id lbrace ODECL rbrace

ODECL -> VDECL ODECL

ODECL -> FDECL ODECL

ODECL -> ''

**(b) Parsing Table**

****

**CFG의 모호성 제거:**

**(a) EXPR의 모호성**

기존 EXPR CFG:

EXPR → EXPR addsub EXPR | EXPR multdiv EXPR

EXPR → lparen EXPR rparen | id | num

수정 후 EXPR CFG:

EXPR → TERM EXPR2

EXPR2 → addsub TERM EXPR2

EXPR2 → ’ ’

TERM → FACTOR TERM2

TERM2 → multdiv FACTOR TERM2

TERM2 → ‘ ‘

FACTOR → lparen EXPR rparen

FACTOR → id

FACTOR → num

🡪 RHS에 EXPR이 두 개 존재

Sol: EXPR과 EXPR2로 분리

🡪 left recursive

🡪 addsub와 multidiv의 우선순위 확립

Sol: 위의 두 가지를 해결하기 위해, TERM, FACTOR라는 새로운 Non-Terminal을 도입했음

우선순위의 해결을 위해, addsub는 EXPR2에서, multdiv는 TERM2에서 parsing 될 수 있도록 함

**(b) COND의 모호성**

기존 COND CFG:

COND → COND comp COND | boolstr

수정 후 COND CFG:

COND → COND2 comp COND

COND → boolstr

COND2 → boolstr

🡪 left recursive

🡪 똑같은 COND가 2개 존재

Sol: 이 두 가지의 해결을 위해 COND, COND2로 분리

**Syntax Analyzer가 Token sequence를 인식하는 과정의 구현:**

(a) Call CFG, Table 🡪 (b) Parser instance 생성 🡪 (c) Parse (Left Viable Check 🡪 Parsing) (반복)

**(a) Call CFG, Table**

cfg.txt와 table.csv로부터, parsing에 사용할 CFG와 SLR Table의 정보를 불러온다.

텍스트, 폰트, 플래시 메모리, 디자인이(가) 표시된 사진

자동 생성된 설명

1. **Call CFG**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

* + Description

cfg\_name (str)을 parameter로 받아 해당하는 경로 내의 txt 파일을 읽어, 처리를 거쳐 dictionary를 반환하는 함수. “LHS -> RHS” 형식으로 이뤄져 있는 각 CFG들은 index 별로 ‘index: [LHS, RHS]’ 형식으로 dictionary cfg에 저장된다.   
Ex. {0: ['S', 'CODE'], 1: ['CODE', 'VDECL CODE'], … }

1. **Call SLR Table**

텍스트, 스크린샷, 소프트웨어, 폰트이(가) 표시된 사진

자동 생성된 설명

* + Description

table\_name (str)을 parameter로 받아 해당 경로 내의 csv 파일을 읽어, 처리를 거쳐 dictionary를 반환하는 함수. Table 형태의 데이터를 읽어, (row, column, value)의 형태로 변환한 후 row를 key로 하고, value로는 {column: value} dictionary를 갖는 dictionay table을 반환한다.

Ex. {0: {'State': 0, 'vtype': 's5', 'class': 's6', '$': 'r4', 'CODE': 1, 'VDECL': 2, 'FDECL': 3, 'CDECL': 4}, … }

**(b) Parser instance 생성**

Terminal로부터 sequence를 입력 받아 parsing 해주는 class

**텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명**

* + Description
    - self.table : call\_table()을 통해 불러온 table dictionary를 받아 저장
    - self.cfg\_table : call\_cfg()를 통해 불러온 cfg dictionary를 받아 저장
    - self.tokens : tokenizer 함수를 통해, terminal로 입력 받은 sequence를 tokenize해 저장함
      * tokenizer

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

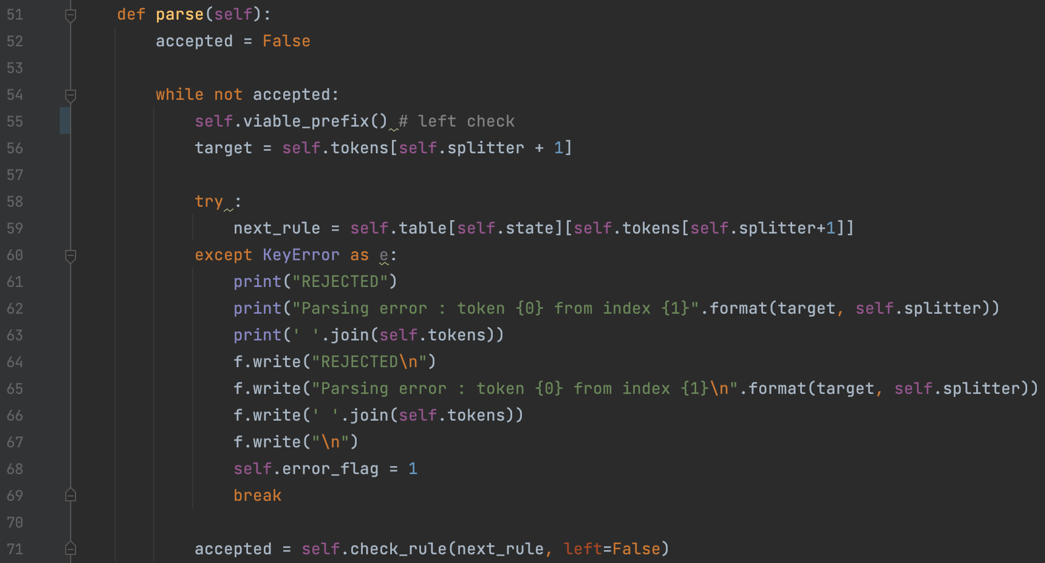
* + - * + Description

Sequence를 공백 기준으로 split한 list 앞에 splitter를 나타내는 “|”, list 뒤에는 EOS를 나타내는 “$”를 append

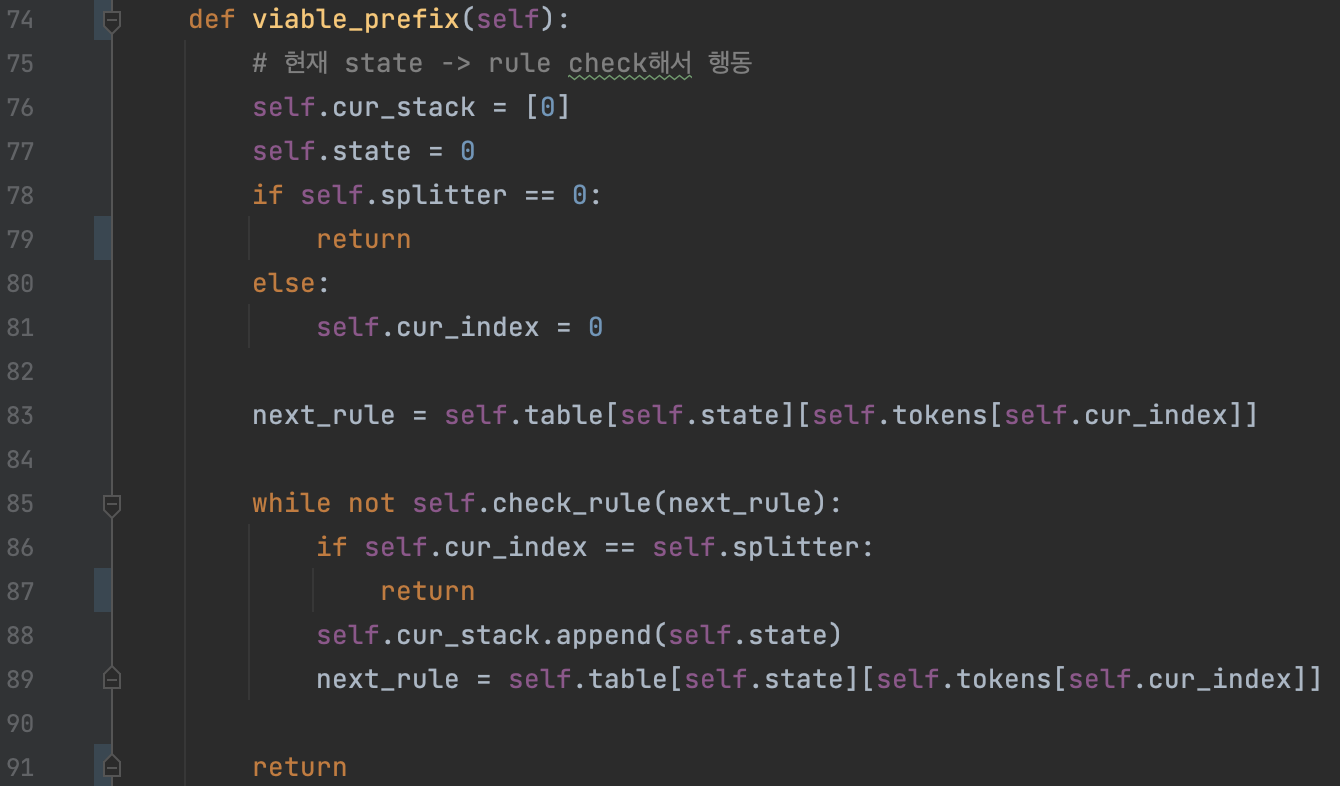
* + - self.splitter : parsing을 해 나가면서, “|” (splitter)를 이동시킬 것인데, 그 때 list 속 splitter의 index를 나타냄
    - self.state : SLR parsing table을 통해 parsing을 진행할 때의 state 값을 저장
    - self.cur\_index : Left Viable Check시에 self.tokens을 iter할 때의 index를 의미
    - self.cur\_stack : reduce 시에 활용하기 위해, left viable check 시의 state들을 저장하는 stack
    - self.state\_stack : tree 생성에 사용할 linked list stack
    - self.error\_flag : error가 발생했을 때의 line을 담는 flag

**(c) Parse**

Parse 과정은 Left Viable Check와 parse(make decision)로 구성됨. 이 과정은 accepted 될 때 까지 일어나며, 중간에 error가 발생하면 exception 처리에 의해 error report 생성됨



1. **Left Viable Prefix**



* + Description

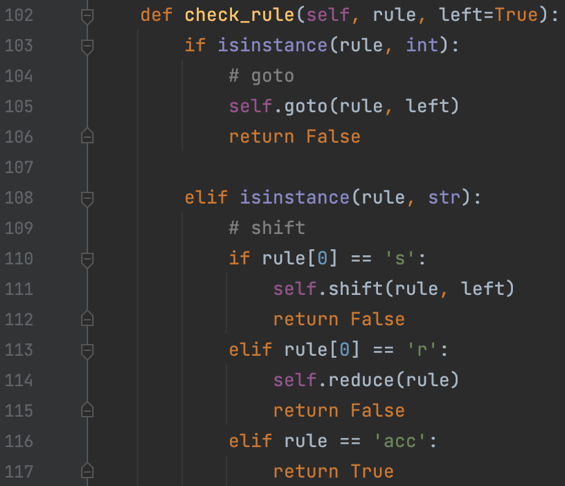
Left substr을 대상으로 viable prefix parse

ⓐ self.cur\_stack, self.state 초기화

ⓑ Left substr이 empty인 경우는 바로 True, 아니라면 self.cur\_index를 0으로 초기화해 left viable prefix check 시작

Ⓒ cur\_index와 state를 기반으로 SLR table에서 적용할 rule을 가져와 next\_rule에 저장

ⓓ self.check\_rule(next\_rule)을 통해 rule에 맞게 상태 및 토큰 Index 변경



* + - Description

- parameter로 받은 rule을 기준으로 decision을 내림

- Self.table의 담겨 있는 rule엔 Shift, Reduce, GOTO, acc 네 가지가 존재함

- Shift: ‘s’ + int   
- Reduce: ‘r’ + int   
- GOTO: int  
- acc: ‘acc’

- rule이 네 가지 중에 존재한다는 가정 하에 진행

- Left parameter는 Left Viable Prefix check 과정 중임을 나타냄. Left Viable Prefix check 도중엔 splitter를 앞으로 옮기는 forward()가 일어나지 않게 함

ⓔ self.cur\_index가 self.splitter와 같아지면, Left substr check가 끝난 것이므로 True 반환

1. **Parse (Make decision)**

Left Viable Prefix check를 통해 update된 self.state를 기반으로 실제 parsing을 진행함.

**ⓐ Shift**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

* + Description

Shift를 구현한 함수. Left viable prefix check의 경우는 self.cur\_index만 증가 시키고, 실제 parsing의 경우는 self.forward()를 통해 splitter를 이동시킴

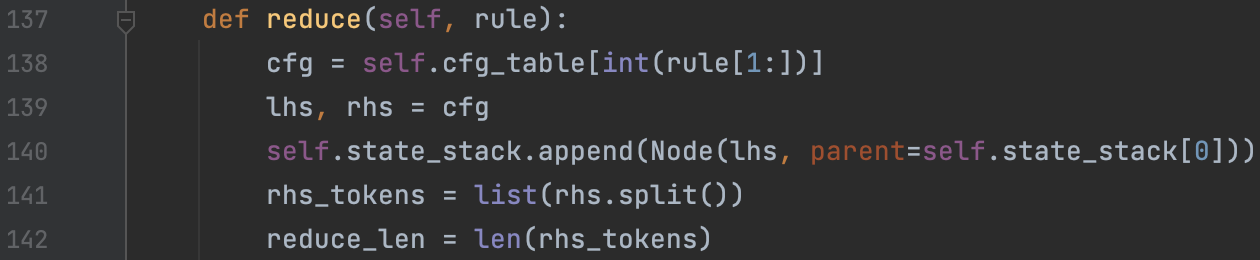
* + - Forward

텍스트, 스크린샷, 폰트이(가) 표시된 사진

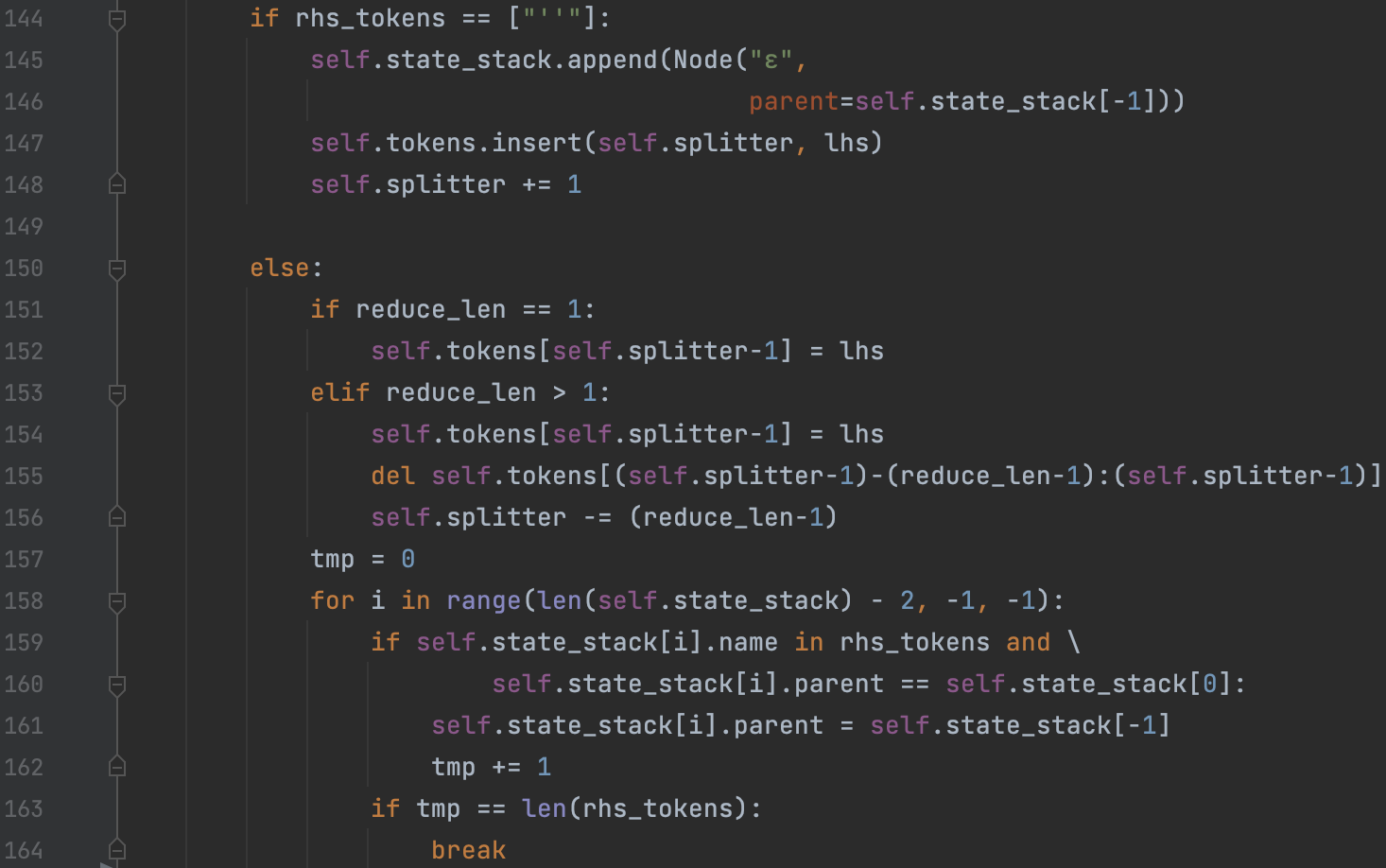
자동 생성된 설명

Splitter를 self.tokens에서 앞으로 한 칸 이동시키고, self.splitter 역시 1 증가시킴  
Tree 구성을 위해 state\_stack에도 추가해줌

**ⓑ Reduce**

  
 Reduce - 1

Reduce를 구현한 함수. Left Viable Prefix Check의 경우는 ?



Reduce -2

Reduce 해야 할 token이 어떤 지에 따라 구분해줌. Token이 [“’’”]인 경우는 epsilon reduce 이므로 lhs를 self.tokens에 추가해줌

하나의 token이 reduce의 대상인 경우, 그대로 대체해주고, 여러 token이 reduce 대상인 경우는 대체 후, self.tokens에서 해당하는 index 만큼 제거

**Ⓒ GOTO**

**텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명**

check\_rule의 rule이 int로 들어오는 경우이므로 left인 경우는 cur\_index 만 증가시키고, left = False인 경우는 self.forward()를 호출해줌

**ⓓ accepted**

Accepted인 경우는 check\_rule에서 True를 반환해 accepted 되어 parsing을 멈추게 함

**Parsing tree 생성 과정의 구현:**

Parsing Tree를 시각화할 때 사용한 Python 라이브러리는 anytree인데, Parsing 과정에서 도출되는 Tree에 최적화되어 있으며, 매우 직관적이기 때문에 해당 라이브러리를 사용하게 되었다. Anytree의 Node()는 Node의 이름과 parent가 존재할 경우 이를 가리키는 변수를 주로 parameter로 받으며, 우리가 구현하고자 하는 Parsing Tree를 위해서는 해당 parameter로 충분하다.

먼저 SLR Parser 클래스인 Parser의 초기화 메소드에 Tree의 노드를 담을 stack을 선언해 준다. Tree 구조를 이용할 수 있기는 하지만, SLR Parser의 구조가 Left-to-right이고 Anytree의 라이브러리 특성상 tree를 빌드하는 로직을 구현하는 데에는 단순한 stack으로도 충분하다고 생각하여 노드를 가리키는 변수로 이루어진 state\_stack이라는 배열을 도입하였다. 초기값은 Parsing을 진행하는 데에 있어 Root node인 Starting non-terminal이 존재해야 하므로 S라는 이름을 가진 Node를 최초로 초기화해준다. SLR Parsing을 진행하며 state\_stack에 추가되어야 하는 경우는 해당 SLR Parser에서 각각 shift와 reduce를 수행하는 과정이며, Shift의 경우 오로지 forward가 발생하는 경우에만 append가 수행되며, forward는 오로지 Shift일 때에만 수행되기 때문에 shift가 아닌 forward 메소드에 동작을 수행하도록 하였다.

SLR Parser에서 Shift가 진행되는 경우 forward에서 state\_stack에 append가 일어난다. 이는 Shift하는 경우 splitter의 왼쪽을 기준으로 Rightmost token를 새로 읽어들이기 때문인데, 해당 토큰은 class 내의 tokens의 splitter 변수값을 index로 하는 구간에 저장되어 있어 해당 위치를 Node의 이름으로 둔다. 이 때, 해당 Node는 결국 root node를 ancestor로 받도록 수정되어야 하기 때문에 초기값은 구애를 받지 않으며, Root node의 child node에 해당하는 token을 받아들이는 경우의 편의성을 위하여 parent의 값은 root node (state\_stack[0])로 설정하여 준다.

Reduce가 진행되는 경우 기준이 되는 Rule에서의 lhs를 state\_stack에 append하는데, 이 때에도 역시 Shift와 동일하게 Parent를 root node로 둔다. 다음으로 reduce를 수행하는 과정에서는 결국 reduce된 rhs\_token에 해당하는 state\_stack 내의 node가 존재하는 경우, 그리고 동일한 token을 가진 state\_stack 내의 node가 원하지 않게 할당되는 것을 막기 위하여 조건을 만족하는 노드 중 가장 rightmost한 token에 대해서, 그리고 해당 token의 parent가 초기값인 root가 아닌 경우에만 parent를 state\_stack 내 lhs에 대응하는 nod로 설정해 준다. 이 때 state\_stack을 탐색하는 방향은 lhs가 아닌 가장 rightmost한 node에서 역방향으로 root 노드에 이르기까지이다.

이렇게 생성된 Tree를 그대로 출력하는 경우에는 Tree의 leaf node를 제외한 모든 node에 대해서 left child과 right child가 바뀌어 있는 상태로 존재하는데, Reduce를 진행하는 과정에서 state\_stack의 탐색 방향을 역순으로 하였기 때문이다. 따라서 실제로 Parsing tree를 출력하는 경우, 출력 방향을 단순히 역순으로 하는 것만으로 문제를 해결할 수 있으며, 최종적으로 ambiguity가 제거된 unique한 tree를 생성할 수 있다.

**Test input files, outputs 목록:**

**(a) input files (input1 ~ 3.txt) 내의 sequence**

1. vtype id lparen vtype id comma vtype id rparen lbrace return num semi rbrace

2. class id lbrace vtype id assign num semi vtype id lparen vtype id comma vtype id rparen lbrace return num semi rbrace rbrace

3. vtype id assign lparen num addsub num rparen multdiv num semi vtype id lparen vtype id rparen lbrace if lparen boolstr comp boolstr comp boolstr rparen lbrace id assign num semi rbrace return num semi rbrace

**(b) 해당 input에 대해 생성된 output tree**

1.

S

└── CODE

├── FDECL

│ ├── vtype

│ ├── id

│ ├── lparen

│ ├── ARG

│ │ ├── vtype

│ │ ├── id

│ │ └── MOREARGS

│ │ ├── comma

│ │ ├── vtype

│ │ ├── id

│ │ └── MOREARGS

│ │ └── ε

│ ├── rparen

│ ├── lbrace

│ ├── BLOCK

│ │ └── ε

│ ├── RETURN

│ │ ├── return

│ │ ├── RHS

│ │ │ └── EXPR

│ │ │ ├── TERM

│ │ │ │ ├── FACTOR

│ │ │ │ │ └── num

│ │ │ │ └── TERM2

│ │ │ │ └── ε

│ │ │ └── EXPR2

│ │ │ └── ε

│ │ └── semi

│ └── rbrace

└── CODE

└── ε

2.

S

└── CODE

├── CDECL

│ ├── class

│ ├── id

│ ├── lbrace

│ ├── ODECL

│ │ ├── VDECL

│ │ │ ├── vtype

│ │ │ ├── ASSIGN

│ │ │ │ ├── id

│ │ │ │ ├── assign

│ │ │ │ └── RHS

│ │ │ │ └── EXPR

│ │ │ │ ├── TERM

│ │ │ │ │ ├── FACTOR

│ │ │ │ │ │ └── num

│ │ │ │ │ └── TERM2

│ │ │ │ │ └── ε

│ │ │ │ └── EXPR2

│ │ │ │ └── ε

│ │ │ └── semi

│ │ └── ODECL

│ │ ├── FDECL

│ │ │ ├── vtype

│ │ │ ├── id

│ │ │ ├── lparen

│ │ │ ├── ARG

│ │ │ │ ├── vtype

│ │ │ │ ├── id

│ │ │ │ └── MOREARGS

│ │ │ │ ├── comma

│ │ │ │ ├── vtype

│ │ │ │ ├── id

│ │ │ │ └── MOREARGS

│ │ │ │ └── ε

│ │ │ ├── rparen

│ │ │ ├── lbrace

│ │ │ ├── BLOCK

│ │ │ │ └── ε

│ │ │ ├── RETURN

│ │ │ │ ├── return

│ │ │ │ ├── RHS

│ │ │ │ │ └── EXPR

│ │ │ │ │ ├── TERM

│ │ │ │ │ │ ├── FACTOR

│ │ │ │ │ │ │ └── num

│ │ │ │ │ │ └── TERM2

│ │ │ │ │ │ └── ε

│ │ │ │ │ └── EXPR2

│ │ │ │ │ └── ε

│ │ │ │ └── semi

│ │ │ └── rbrace

│ │ └── ODECL

│ │ └── ε

│ └── rbrace

└── CODE

└── ε

3.

S

└── CODE

├── VDECL

│ ├── vtype

│ ├── ASSIGN

│ │ ├── id

│ │ ├── assign

│ │ └── RHS

│ │ └── EXPR

│ │ ├── TERM

│ │ │ ├── FACTOR

│ │ │ │ ├── lparen

│ │ │ │ ├── EXPR

│ │ │ │ │ ├── TERM

│ │ │ │ │ │ ├── FACTOR

│ │ │ │ │ │ │ └── num

│ │ │ │ │ │ └── TERM2

│ │ │ │ │ │ └── ε

│ │ │ │ │ └── EXPR2

│ │ │ │ │ ├── addsub

│ │ │ │ │ ├── TERM

│ │ │ │ │ │ ├── FACTOR

│ │ │ │ │ │ │ └── num

│ │ │ │ │ │ └── TERM2

│ │ │ │ │ │ └── ε

│ │ │ │ │ └── EXPR2

│ │ │ │ │ └── ε

│ │ │ │ └── rparen

│ │ │ └── TERM2

│ │ │ ├── multdiv

│ │ │ ├── FACTOR

│ │ │ │ └── num

│ │ │ └── TERM2

│ │ │ └── ε

│ │ └── EXPR2

│ │ └── ε

│ └── semi

└── CODE

├── FDECL

│ ├── vtype

│ ├── id

│ ├── lparen

│ ├── ARG

│ │ ├── vtype

│ │ ├── id

│ │ └── MOREARGS

│ │ └── ε

│ ├── rparen

│ ├── lbrace

│ ├── BLOCK

│ │ ├── STMT

│ │ │ ├── if

│ │ │ ├── lparen

│ │ │ ├── COND

│ │ │ │ ├── COND2

│ │ │ │ │ └── boolstr

│ │ │ │ ├── comp

│ │ │ │ └── COND

│ │ │ │ ├── COND2

│ │ │ │ │ └── boolstr

│ │ │ │ ├── comp

│ │ │ │ └── COND

│ │ │ │ └── boolstr

│ │ │ ├── rparen

│ │ │ ├── lbrace

│ │ │ ├── BLOCK

│ │ │ │ ├── STMT

│ │ │ │ │ ├── ASSIGN

│ │ │ │ │ │ ├── id

│ │ │ │ │ │ ├── assign

│ │ │ │ │ │ └── RHS

│ │ │ │ │ │ └── EXPR

│ │ │ │ │ │ ├── TERM

│ │ │ │ │ │ │ ├── FACTOR

│ │ │ │ │ │ │ │ └── num

│ │ │ │ │ │ │ └── TERM2

│ │ │ │ │ │ │ └── ε

│ │ │ │ │ │ └── EXPR2

│ │ │ │ │ │ └── ε

│ │ │ │ │ └── semi

│ │ │ │ └── BLOCK

│ │ │ │ └── ε

│ │ │ ├── rbrace

│ │ │ └── ELSE

│ │ │ └── ε

│ │ └── BLOCK

│ │ └── ε

│ ├── RETURN

│ │ ├── return

│ │ ├── RHS

│ │ │ └── EXPR

│ │ │ ├── TERM

│ │ │ │ ├── FACTOR

│ │ │ │ │ └── num

│ │ │ │ └── TERM2

│ │ │ │ └── ε

│ │ │ └── EXPR2

│ │ │ └── ε

│ │ └── semi

│ └── rbrace

└── CODE

└── ε