

# README

## simple-java-parser

2023-1 Compiler Assignment: Making Parser for Simple Java

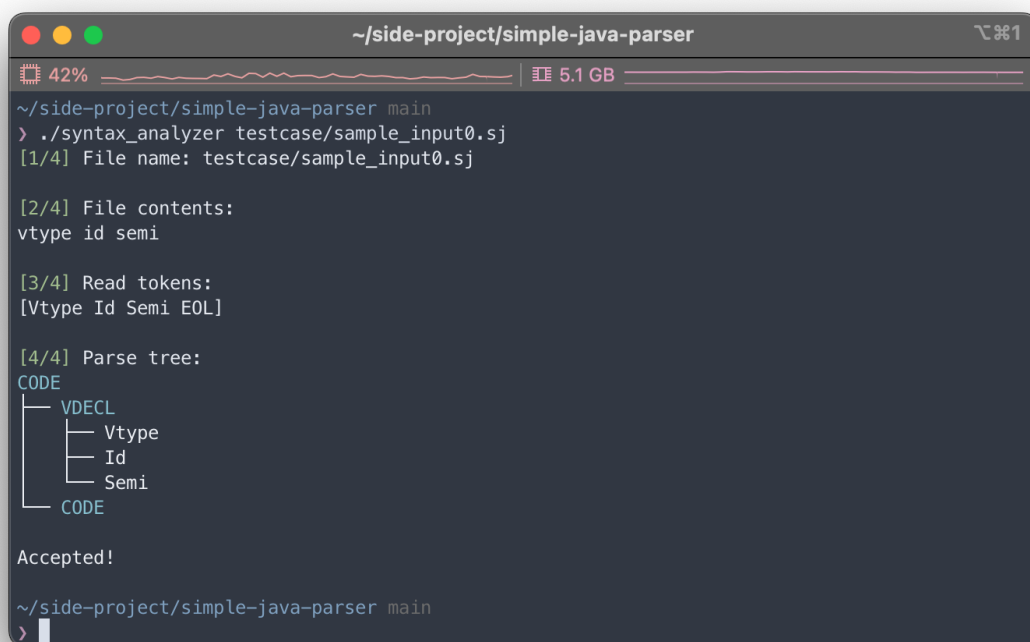
team\_id: 67

author: 20203458 조영호

## 실행

Ubuntu 20.04.6 LTS x86\_64에서 컴파일 되었습니다.

```
$ ./syntax_analyzer testcase/sample_input0.sj
```



```
~/side-project/simple-java-parser main
> ./syntax_analyzer testcase/sample_input0.sj
[1/4] File name: testcase/sample_input0.sj

[2/4] File contents:
vtype id semi

[3/4] Read tokens:
[Vtype Id Semi EOL]

[4/4] Parse tree:
CODE
├── VDECL
│   ├── Vtype
│   ├── Id
│   └── Semi
└── CODE

Accepted!

~/side-project/simple-java-parser main
>
```

## build

```
# rust 설치 (https://www.rust-lang.org/learn/get-started)
$ curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
# 컴파일
```

```
$ cargo build --release
$ cp target/release/simple-java-parser syntax_analyzer
# 실행
$ ./syntax_analyzer testcase/sample_input0.sj
```

## 수정된 CFG

```
00: CODE' -> CODE
01: CODE -> VDECL CODE
02: CODE -> FDECL CODE
03: CODE -> CDECL CODE
04: CODE -> ''
05: VDECL -> vtype id semi
06: VDECL -> vtype ASSIGN semi
07: ASSIGN -> id assign RHS
08: RHS -> EXPR
09: RHS -> literal
10: RHS -> character
11: RHS -> boolstr
12: EXPR -> EXPR addsub EXPR'
13: EXPR -> EXPR'
14: EXPR' -> EXPR' multdiv EXPR''
15: EXPR' -> EXPR''
16: EXPR'' -> lparen EXPR rparen
17: EXPR'' -> id
18: EXPR'' -> num
19: FDECL -> vtype id lparen ARG rparen lbrace BLOCK RETURN rbrace
20: ARG -> vtype id MOREARGS
21: ARG -> ''
22: MOREARGS -> comma vtype id MOREARGS
23: MOREARGS -> ''
24: BLOCK -> STMT BLOCK
25: BLOCK -> ''
26: STMT -> VDECL
27: STMT -> ASSIGN semi
28: STMT -> if lparen COND rparen lbrace BLOCK rbrace ELSE
29: STMT -> while lparen COND rparen lbrace BLOCK rbrace
30: COND -> COND comp boolstr
31: COND -> boolstr
32: ELSE -> else lbrace BLOCK rbrace
33: ELSE -> ''
34: RETURN -> return RHS semi
35: CDECL -> class id lbrace ODECL rbrace
36: ODECL -> VDECL ODECL
37: ODECL -> FDECL ODECL
38: ODECL -> ''
```

## 변경사항 1 (CODE)

최상단인 `CODE` 를 가리키는 `CODE' -> CODE` 를 추가하였습니다.

```
CODE -> VDECL CODE
CODE -> FDECL CODE
CODE -> CDECL CODE
CODE -> ''
```

->

```
CODE' -> CODE
CODE -> VDECL CODE
CODE -> FDECL CODE
CODE -> CDECL CODE
CODE -> ''
```

## 변경사항 2 (COND)

`COND comp COND comp COND`의 ambiguous를 해결하기 위해 수정하였습니다.

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

->

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

## 변경사항 3 (EXPR)

`addsub`, `multdiv`, `lparen`, `rparen`, `id`, `num`의 우선순위에 대한 ambiguous를 해결하기 위해 수정하였습니다.

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

->

```
EXPR -> EXPR addsub EXPR'
EXPR -> EXPR'
EXPR' -> EXPR' multdiv EXPR''
```

```

EXPR' -> EXPR'
EXPR' -> lparen EXPR rparen
EXPR' -> id
EXPR' -> num

```

## parsing table

[img/parsing\\_table.jpg](#) 에서 원본 사진을 보실 수 있습니다.

[illegible]

## 동작 과정

이 parser는 네가지 단계에 거쳐 parsing tree를 생성합니다.

Step 1. 인자로부터 파일 이름 가져오기

Step 2. 파일 읽기

Step 3. String을 whitespace으로 나누어 token 인식하기

Step 4. 인식된 토큰을 parsing하여 parsing tree 생성하기

```
fn main() {
    // 1. 입력받은 파일 이름이 있다면 filename에 저장합니다.
    let filename = match env::args().nth(1) {
        (생략...)
    };

    // 2. 파일 읽기에 성공하면 raw_contents에 저장합니다.
    let raw_contents = match fs::read_to_string(filename) {
        (생략...)
    };

    // 3. token을 인식하는데 성공하면 tokens의 배열을 tokens에 저장합니다.
    // token_reader::reader_token(...) 함수는 src/token_reader.rs 에 작성되어 있습니다.
    let tokens = match token_reader::read_tokens(&raw_contents) {
        (생략...)
    };

    // 4. 입력된 tokens를 parsing하고 parse tree를 생성한 후 출력합니다.
    // parser::parse(...) 함수는 src/parser/mod.rs 에 작성되어 있습니다.
    match parser::parse(tokens) {
        (생략...)
    };
}
```

## Step 3 주요 struct, enum, procedure

### read\_tokens (In src/token\_reader.rs)

String을 white space로 나누어 문자열 비교를 통해 Token의 배열을 return하는 함수입니다.

```
pub fn read_tokens(contents: &String) -> Result<Tokens, UnknownTokenError> {
    let mut tokens = VecDeque::new();

    for word in contents.split_whitespace() {
        let token = match word {
            "vtype" => Token::Vtype,
            "num" => Token::Num,
            "character" => Token::Character,
            (생략...)
        };
        tokens.push_back(token);
    }
}
```

```

        // token 인식을 실패하면 UnknownTokenError에 정보를 담아 return합니다.
        unknown_token => return Err(UnknownTokenError(unknown_token)),
    };
    tokens.push_back(Terminal(token));
}

// token을 모두 인식하였다면 마지막에 EOL token을 추가하고 return합니다.
tokens.push_back(Terminal(Token::EOL));
Ok(Tokens(tokens))
}

```

## Token (In src/token\_reader.rs)

terminal과 non-terminal, EOL을 나타내는 enum입니다.

```

pub enum Token {
    // terminals
    Vtype,      // for the types of variables and function
    Num,        // for signed integers
    Character,  // for a single character
    (생략...)

    // for EOL
    EOL,

    // non-terminals
    CODE,
    CODE_,
    VDECL,
    (생략...)
}

```

## Node (In src/parser/mod.rs)

terminal과 non-terminal을 나타내며 동시에 tree의 node를 나타내는 enum입니다.

read\_tokens에서는 `Terminal(Token)` 만 사용되며 `NonTerminal(Token, Vec<Node>)` 는 Step 4 parse함수에서 사용됩니다.

```

pub enum Node {
    Terminal(Token),
    NonTerminal(Token, Vec<Node>),
}

```

## Tokens (In src/parser/formatting.rs)

Node의 배열을 나타내는 struct입니다.

여기서 VecDeque는 double-ended queue를 표현하는 Rust의 내장 collection입니다.

```
pub struct Tokens(pub VecDeque<Node>);
```

## Step 4 주요 struct, enum, procedure

### parse (In src/parser/mod.rs)

parsing table과 reduction table을 참조하여 parsing하며 tree를 생성하는 함수입니다.

parsing table에 해당하는 rule(shift, reduce, goto, accepted)이 있다면 그것을 실행하고, 없다면 ParsingError를 return합니다.

```
pub fn parse(tokens: Tokens) -> Result<Tree, ParsingError> {
    let mut tokens = tokens.0;

    // parsing_table::get_parsing_table()은 src/parser/parsing_table.rs에 정의되어 있습니다.
    // hard coding된 parsing table을 return하는 함수입니다.
    let parsing_table = parsing_table::get_parsing_table();

    // parsing_table::get_reduction_table()은 reduction을 할 때 필요한 CFG를 return하는
    // 함수이며 나머지는 parsing_table::get_parsing_table() 함수와 같습니다.
    let reduction_table = parsing_table::get_reduction_table();

    // stack을 생성한 후 state 0을 넣습니다.
    let mut stack = vec![StackItem::from(0, None)];

    loop {
        // parsing table을 검색하기 위해 현재 state와 next token 정보를 가져옵니다.
        let current_state = stack.last().unwrap().state;
        let next_token = match tokens[0] {
            Terminal(token) => token,
            NonTerminal(token, _) => token,
        };

        // parsing_table에 현재 state와 next token에 맞는 rule이 있다면
        // 해당 rule을 처리합니다.
        match parsing_table[current_state].get(&next_token) {
            Some(behavior) => match behavior {
                // 찾아진 rule에 따라 parsing을 진행합니다.
                // shift와 goto를 하나의 함수로 구현하여 구현을 단순화 하였습니다.
                // Shift, Reduce, Goto, Accepted 는 src/parser/parsing_table.rs에 Table
                Element로
                // 정의되어 있으며, parsing table의 rule을 나타냅니다.
                Shift(next_state) => shift_goto(&mut tokens, &mut stack, *next_state),
                Reduce(reduction_index) => reduce(&mut tokens, &mut stack, reduction_t
                able[*reduction_index]),
                Goto(next_state) => shift_goto(&mut tokens, &mut stack, *next_state),
                Accepted => break,
            },
            None => {
                // 현재 state와 next token에 맞는 rule이 없다면 부가 정보를 담은 Error를
```

```

        // return합니다.
        return Err(ParsingError(parsing_table[current_state].keys().cloned().collect(), next_token));
    },
};

}

// parsing이 완료되면 stack의 최상단에 저장된 node를 Tree로 wrapping하여 return합니다.
Ok(Tree(stack.pop().unwrap().tree.unwrap()))
}

```

## shift\_goto (In src/parser/mod.rs)

shift와 goto를 실행하는 함수입니다.

```

fn shift_goto(tokens: &mut VecDeque<Node>, stack: &mut Vec<StackItem>, next_state: usize) {
    // tokens에서 token을 pop하여
    let next_token = tokens.pop_front().unwrap();
    // next state와 함께 stack에 push합니다.
    stack.push(StackItem::from(next_state, Some(next_token)));
}

```

## reduce (In src/parser/mod.rs)

reduce를 실행하는 함수입니다.

```

fn reduce(tokens: &mut VecDeque<Node>, stack: &mut Vec<StackItem>, reduction: Reduction) {
    // 해당 reduction rule에 맞는 수 만큼 stack에서 pop하여
    // children으로 push합니다.
    let mut children: Vec<Node> = vec![];
    for _ in 0..reduction.right {
        children.push(stack.pop().unwrap().tree.unwrap());
    }
    children.reverse();

    // children과 함께 non-terminal(reduction.left)을 tokens에 push합니다.
    // 이렇게 push된 non-terminal은 shift와 goto가 함께 구현된 shift_goto를 거쳐
    // 알맞게 stack으로 다시 push됩니다.
    tokens.push_front(NonTerminal(reduction.left, children));
}

```

## StackItem (In src/parser/mod.rs)

stack에 state뿐만 아니라 parsing중인 node를 함께 저장합니다.



```
struct StackItem {
    state: usize,
    tree: Option<Node>,
}
```

## Tree (In src/parser/formatting.rs)

Node를 wrapping하는 struct입니다.

```
pub struct Tree(pub Node);
```

## get\_reduction\_table (In src/parser/parsing\_table.rs)

reduction rule을 hard coding한 함수입니다.

CFG에 따라 총 39개의 rule이 작성되어있습니다.

```
pub fn get_reduction_table() -> Vec<Reduction> {
    let mut table = vec![];

    table.push(Reduction::from(CODE_, 1)); // 0
    table.push(Reduction::from(CODE, 2));
    table.push(Reduction::from(CODE, 2));
    table.push(Reduction::from(CODE, 2));
    (생략...)

    table
}
```

## Reduction (In src/parser/parsing\_table.rs)

하나의 reduction rule을 나타내는 struct입니다.

left는 좌항의 non-terminal을 나타내고, right는 우항의 non-terminal, terminal의 수를 나타냅니다.

right는 reduce함수에서 stack에서 pop을 하는 횟수를 확인하는 데 사용됩니다.

```
CODE -> '' // Reduction { left: CODE, right: 0 }
VDECL -> vtype id semi // Reduction { left: VDECL, right: 3 }
```

```
pub struct Reduction {
    pub left: Token,
```

```
pub right: usize,
}
```

## get\_parsing\_table (In src/parser/parsing\_table.rs)

이 함수에는 parsing table이 hard coding되어있습니다.

```
pub fn get_parsing_table() -> Vec<HashMap<Token, TableElement>> {
    let mut table = vec![];

    // for state 0
    let mut hashmap = HashMap::new();
    hashmap.insert(Vtype, Shift(5));
    hashmap.insert(Class, Shift(6));
    hashmap.insert(EOL, Reduce(4));
    (생략...)
    table.push(hashmap);

    (생략...)

    table
}
```

## TableElement (In src/parser/parsing\_table.rs)

parsing table의 element를 나타내는 요소입니다.

```
pub enum TableElement {
    Shift(usize),
    Reduce(usize),
    Goto(usize),
    Accepted,
}
```

## test case

### case 0

```
// In testcase/sample_input0.sj
vtype id semi
```

```
$ ./syntax_analyzer testcase/sample_input0.sj
[1/4] File name: testcase/sample_input0.sj

[2/4] File contents:
```

```
vtype id semi

[3/4] Read tokens:
[Vtype Id Semi EOL]

[4/4] Parse tree:
CODE
├─ VDECL
│   ├─ Vtype
│   │   └─ Id
│   │       └─ Semi
└─ CODE

Accepted!
```

## case 1

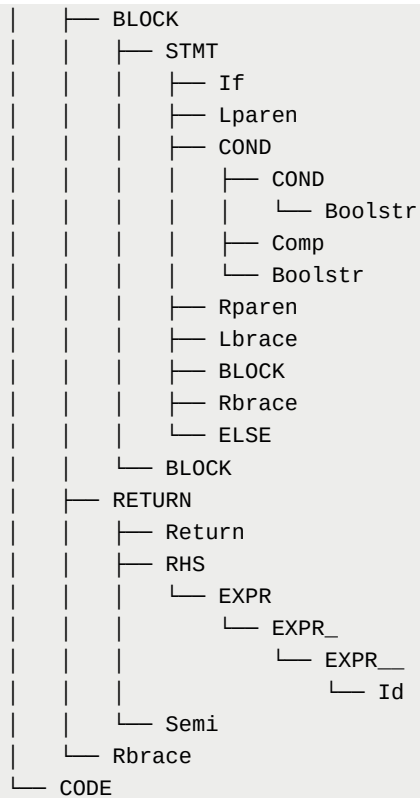
```
// In testcase/sample_input1.sj
vtype id semi
vtype id lparen rparen lbrace
    if lparen boolstr comp boolstr rparen lbrace
        rbrace
return id semi rbrace
```

```
$ ./syntax_analyzer testcase/sample_input1.sj
[1/4] File name: testcase/sample_input1.sj

[2/4] File contents:
vtype id semi
vtype id lparen rparen lbrace
    if lparen boolstr comp boolstr rparen lbrace
        rbrace
return id semi rbrace

[3/4] Read tokens:
[Vtype Id Semi Vtype Id Lparen Rparen Lbrace If Lparen Boolstr Comp Boolstr Rparen Lbr
ace Rbrace Return Id Semi Rbrace EOL]

[4/4] Parse tree:
CODE
├─ VDECL
│   ├─ Vtype
│   │   └─ Id
│   │       └─ Semi
└─ CODE
    ├─ FDECL
    │   ├─ Vtype
    │   │   └─ Id
    │   │       └─ Lparen
    │   │           └─ ARG
    │   │               └─ Rparen
    │   │                   └─ Lbrace
```



## case 2

```
// In testcase/sample_input2.sj
class id lbrace
  vtype id semi
  vtype id assign id addsub lparen num multdiv id rparen semi

  vtype id lparen rparen lbrace
    id assign num multdiv num semi
    while lparen boolstr comp boolstr comp boolstr rparen lbrace
      id assign literal semi
      id assign boolstr semi
    rbrace
  return id semi
rbrace

vtype id lparen vtype id comma vtype id rparen lbrace
  if lparen boolstr rparen lbrace
    rbrace
  return num addsub id semi
rbrace

rbrace
```

```
$ ./syntax_analyzer testcase/sample_input2.sj
```

```
[1/4] File name: testcase/sample_input2.sj
```

```
[2/4] File contents:
```

```
class id lbrace
  vtype id semi
  vtype id assign id addsub lparen num multdiv id rparen semi

  vtype id lparen rparen lbrace
    id assign num multdiv num semi
    while lparen boolstr comp boolstr comp boolstr rparen lbrace
      id assign literal semi
      id assign boolstr semi
    rbrace
    return id semi
  rbrace

  vtype id lparen vtype id comma vtype id rparen lbrace
    if lparen boolstr rparen lbrace
      rbrace
      return num addsub id semi
    rbrace

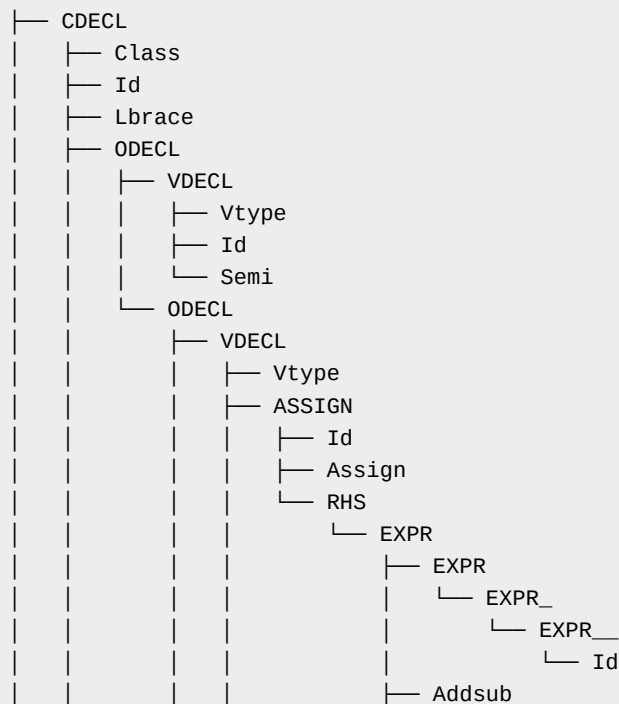
rbrace
```

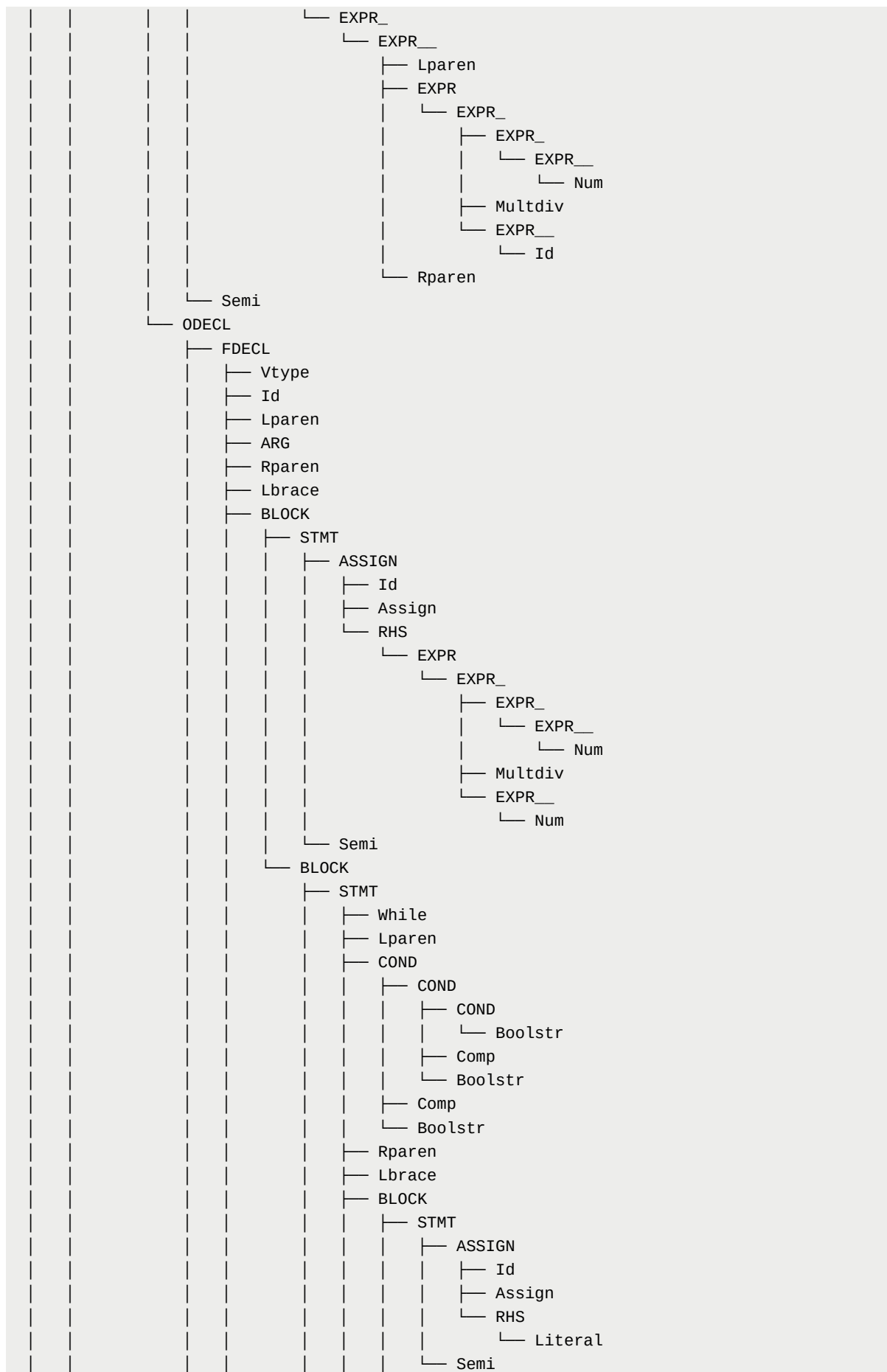
```
[3/4] Read tokens:
```

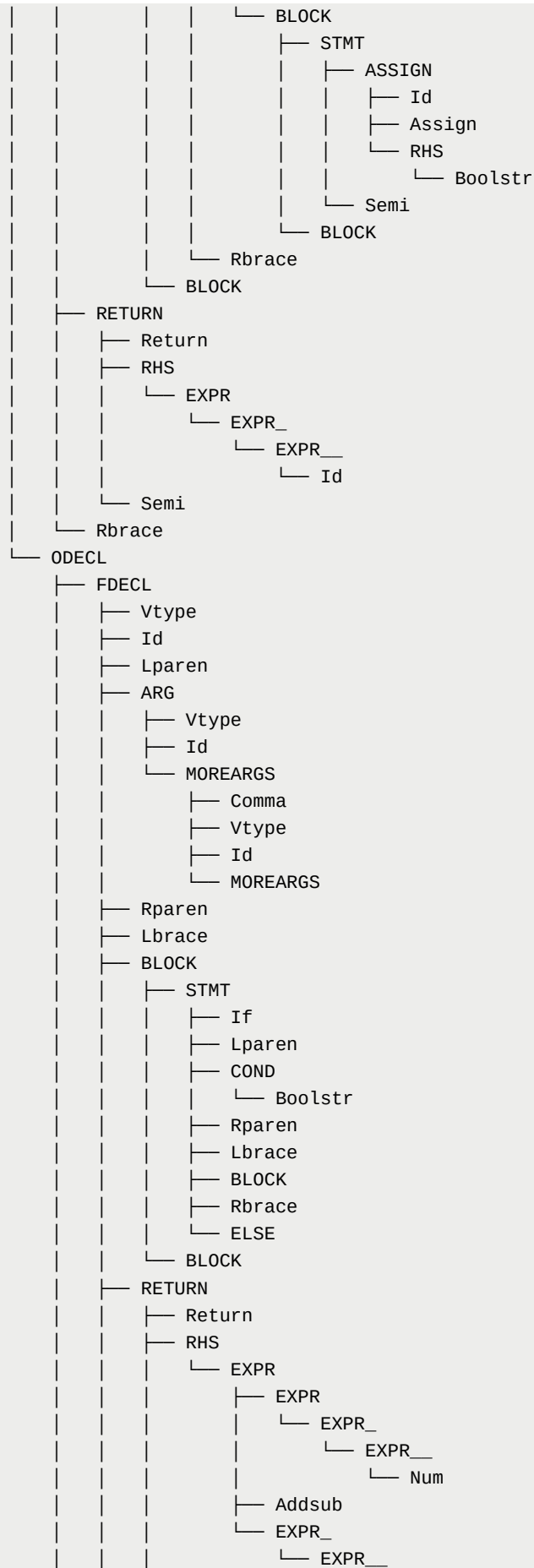
```
[Class Id Lbrace Vtype Id Semi Vtype Id Assign Id Addsub Lparen Num Multdiv Id Rparen
Semi Vtype Id Lparen Rparen Lbrace Id Assign Num Multdiv Num Semi While Lparen Boolstr
r Comp Boolstr Comp Boolstr Rparen Lbrace Id Assign Literal Semi Id Assign Boolstr Sem
i Rbrace Return Id Semi Rbrace Vtype Id Lparen Vtype Id Comma Vtype Id Rparen Lbrace I
f Lparen Boolstr Rparen Lbrace Rbrace Return Num Addsub Id Semi Rbrace Rbrace EOL]
```

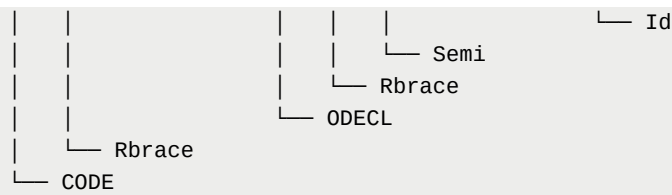
```
[4/4] Parse tree:
```

```
CODE
```









Accepted!

## case 3

```
// In testcase/sample_input3.sj
vtype id lparen rparen lbrace
  id assign id addsub id multdiv id addsub id multdiv id semi

  return id semi
rbrace
```

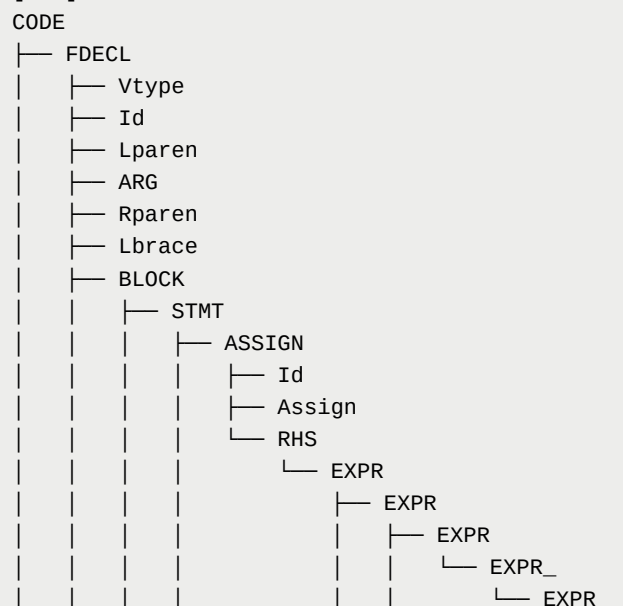
```
$ ./syntax_analyzer testcase/sample_input3.sj
[1/4] File name: testcase/sample_input3.sj

[2/4] File contents:
vtype id lparen rparen lbrace
  id assign id addsub id multdiv id addsub id multdiv id semi

  return id semi
rbrace

[3/4] Read tokens:
[Vtype Id Lparen Rparen Lbrace Id Assign Id Addsub Id Multdiv Id Addsub Id Multdiv Id
Semi Return Id Semi Rbrace EOL]
```

[4/4] Parse tree:



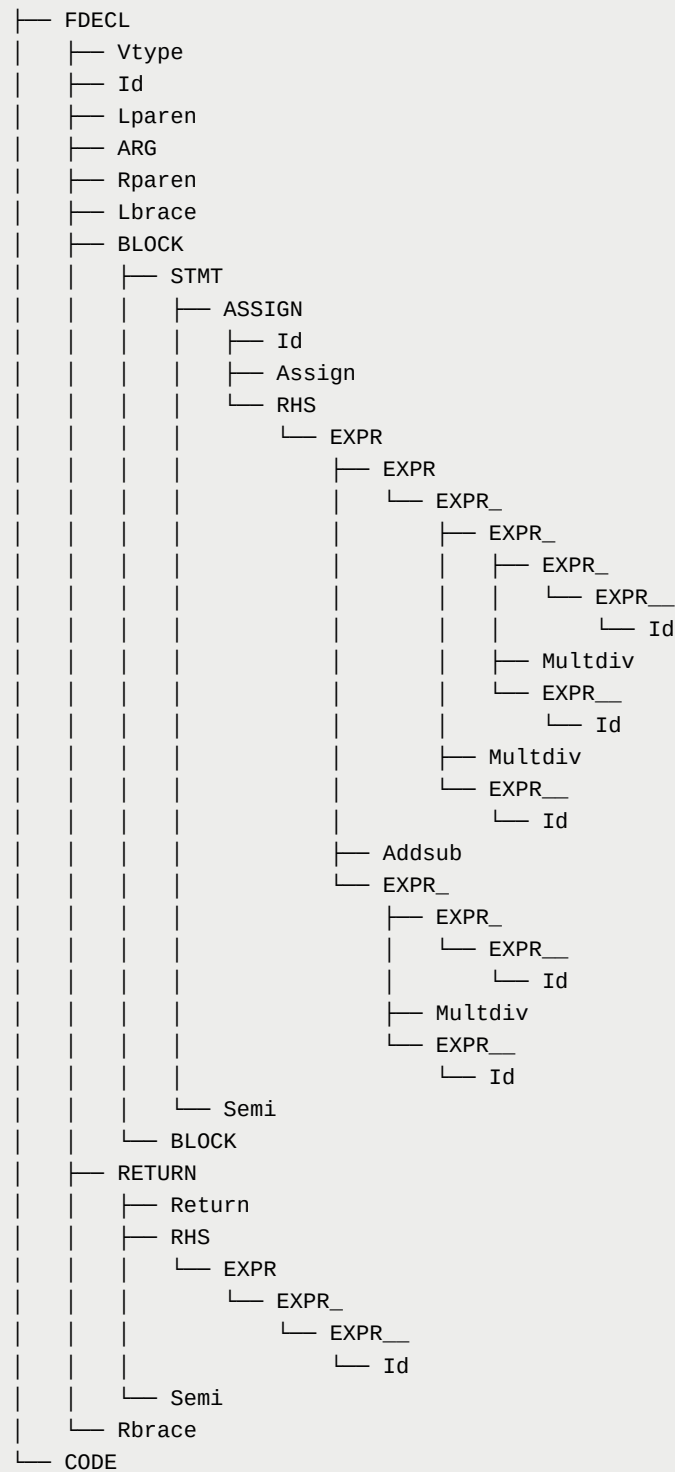




[Vtype Id Lparen Rparen Lbrace Id Assign Id Multdiv Id Multdiv Id Addsub Id Multdiv Id  
Semi Return Id Semi Rbrace EOL]

[4/4] Parse tree:

CODE



Accepted!

## fail case 0

```
// In testcase/sample_fail_input0.sj
CODE
```

```
$ ./syntax_analyzer testcase/sample_fail_input0.sj
[1/4] File name: testcase/sample_fail_input0.sj

[2/4] File contents:
CODE

[3/4] error: unknown token: CODE
```

## fail case 1

```
// In testcase/sample_fail_input1.sj
vtype id semi vtype id lparen rparen lbrace if lparen boolstr comp boolstr rparen lbrace rbrace
```

```
$ ./syntax_analyzer testcase/sample_fail_input1.sj
[1/4] File name: testcase/sample_fail_input1.sj

[2/4] File contents:
vtype id semi vtype id lparen rparen lbrace if lparen boolstr comp boolstr rparen lbrace rbrace

[3/4] Read tokens:
[Vtype Id Semi Vtype Id Lparen Rparen Lbrace If Lparen Boolstr Comp Boolstr Rparen Lbrace Rbrace EOL]

[4/4] error: parsing error
  expected: [ELSE, Vtype, Else, If, Id, While, Return, Rbrace]
  but found: EOL
```

## fail case 2

```
// In testcase/sample_fail_input2.sj
vtype id semi vtype id lparen rparen lbrace if lparen boolstr comp boolstr rparen lbrace return id semi rbrace
```

```
$ ./syntax_analyzer testcase/sample_fail_input2.sj
[1/4] File name: testcase/sample_fail_input2.sj

[2/4] File contents:
vtype id semi vtype id lparen rparen lbrace if lparen boolstr comp boolstr rparen lbrace
```

```
ce return id semi rbrace
```

```
[3/4] Read tokens:
```

```
[Vtype Id Semi Vtype Id Lparen Rparen Lbrace If Lparen Boolstr Comp Boolstr Rparen Lbrace Return Id Semi Rbrace EOL]
```

```
[4/4] error: parsing error
```

```
    expected: [Rbrace]
```

```
    but found: Return
```

## fail case 3

```
// In testcase/sample_fail_input3.sj
vtype id lparen rpalren lbrace
    return id semi
rbrace
```

```
$ ./syntax_analyzer testcase/sample_fail_input3.sj
```

```
[1/4] File name: testcase/sample_fail_input3.sj
```

```
[2/4] File contents:
```

```
vtype id lparen rpalren lbrace
```

```
    return id semi
```

```
rbrace
```

```
[3/4] error: unknown token: rpalren
```

## fail case 4

no input

```
$ ./syntax_analyzer
```

```
[1/4] error: no input file
```

## fail case 5

fail to read file

```
$ ./syntax_analyzer awkjnviwunalwdj
```

```
[1/4] File name: awkjnviwunalwdj
```

```
[2/4] error: something went wrong during reading file
```