**COMPILER**

**TERM PROJECT**

Junseok Park

20200267

Department of Software

1. Below is the CFG I used.

01 : S -> CODE

02 : CODE -> VDECL CODE' | FDECL CODE' | CDECL CODE' | ^

03 : CODE' -> VDECL CODE' | FDECL CODE' | CDECL CODE' | ^

04 : VDECL -> vtype id semi | vtype ASSIGN semi

05 : ASSIGN -> id assign RHS

06 : RHS -> EXPR | literal | character | boolstr

07 : EXPR -> TERM EXPR'

08 : EXPR' -> addsub TERM EXPR' | multdiv FACTOR EXPR' | ^

09 : TERM -> FACTOR TERM'

10 : TERM' -> multdiv FACTOR TERM' | ^

11 : FACTOR -> lparen EXPR rparen | id | num | boolstr

12 : FDECL -> vtype id lparen ARG rparen lbrace BLOCK RETURN rbrace | vtype id lparen ARG rparen

13 : ARG -> vtype id MOREARGS | ^

14 : MOREARGS -> comma vtype id MOREARGS | ^

15 : BLOCK -> STMT BLOCK | ^

16 : STMT -> VDECL | ASSIGN semi | if lparen COND rparen lbrace BLOCK rbrace ELSE | while lparen COND rparen lbrace BLOCK rbrace

17 : COND -> EXPR comp EXPR

18 : ELSE -> else lbrace BLOCK rbrace | ^

19 : RETURN -> return RHS semi | ^

20 : CDECL -> class id lbrace ODECL rbrace

21 : ODECL -> VDECL ODECL' | FDECL ODECL' | ^

22 : ODECL' -> VDECL ODECL' | FDECL ODECL' | ^

2. How to execute the program.

First, you need to build an environment to run programs. First, run "python -m venv venv" to have a 'python virtualenv' environment, and then install the required library list as "pip install -r requirements.txt". This completes the preparation for running the program.

First, put the code you want to parse into the text file named input\_code.txt. The next time you run "python lexer.py " in the console window, a file is created with the entered code "tokens.txt" as a token. Next, the command "python slr\_parser.py cfg.txt tokenens.txt" in the console window creates a slr-parse table and outputs the parsing process and the parse tree in token. If you run the command as "python slr\_parser.py -g cfg.txt token.txt", you will get automaton.

I'll show you how to do it in more detail through pictures.

텍스트, 스크린샷, 소프트웨어, 컴퓨터이(가) 표시된 사진

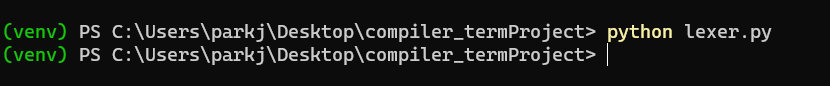
자동 생성된 설명

Open powershell in the project directory.

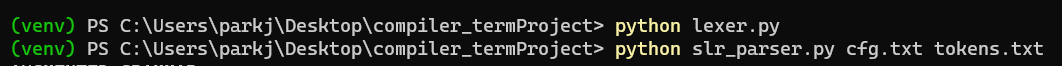
텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

Run Python's virtualenv.



Run lexer.py to tokenize the code. If you want to insert token, skip this task and modify token.txt only.



Run the command to parse.

3. Code Configuration.

First, the code is divided into three parts. It is divided into lexer.py , which receives the code and does the flexing, grammar.py , which receives and interprets the CFG needed to do the parsing, and slr\_parser.py, which builds the parser.

lexer.py has a definition of token to change it to token. When you receive input\_code.txt., it is listed as a single token, and Tokens.txt is output.

grammar.py contains code that interprets CFG. Because it is provided in text form, it helps to interpret CFG well, including the process of parsing text.

In slr\_parser, import grammar for a given CFG to build a parser. This code has a function that configures and represents the first\_follow set, and within the SLRParser class are the construct\_table function that creates a parsing table, the generate\_automaton function that creates automaton, and the LR\_Parser function that creates the most important LR\_Parser. To explain the LR\_Parser function, use stack to perform parsing. The code implementing the Parsing table was also created by reusing the stack used here. In particular, an algorithm was used to make reduce a parent node.

4. Some simple examples

Let me show you the simplest example.

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

Create content in Input\_code.txt.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

Running Python lexer.py creates the following tokens.txt.

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

The command creates the following parsing processes and a parse tree.