MILITARY INSTITUTE OF SCIENCE AND TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
COURSE CODE: CSE-304, COURSE TITLE: COMPILER SESSIONAL SPRING 2024
ASSIGNMENT 3 (Group A1 & B1)

This Assignment is regarded for assessment of CO1 and CO2.

CO1: Remember, understand and apply the basic techniques of compiler construction and tools to perform syntax-directed translation of a high-level programming language into an executable code.

CO2: Understand the working mechanisms of lex and yacc compiler for debugging of programs.

1 Introduction

In the previous assignment, we have constructed a lexical analyzer to generate token streams. In this assignment, we will construct a syntax analyzer of a compiler considering subset of C language using a set of grammar rules. To do so, we will build a parser with the help of Lex (Flex) and Yacc (Bison).

Language:

You should consider the following subset of C language which has the following characteristics:

- Variables can be declared at any places of the source program. Multiple variables of the same type can be declared separated by comma and end with a semicolon.
- Initialization of variable in the declaration statement is also allowed (consider in only single variable declaration)
- Declaration of array is also permissible.
- All the operators used for any expression are included. Precedence and associativity rules should be maintained as per standard.

Note:

- There will be no pre-processing directives like #include or #define.
- No break statement, switch-case statement, loop and conditional statement will be used.
- Ignore consecutive logical operators or consecutive relational operators like, a && b && c, a < b < c.

2 Tasks

You have to complete the following tasks in this assignment building the Syntax Analyzer.

1. Incorporate the token from the last assignment (Assignment-3) with the sample grammar given in **Grammar-1** in your Yacc file. [attached at last]
2. Modify your Lex file from the previous assignment to use it with your Yacc file.
3. Insert all the identifiers and function in the symbol table when they are declared in the input file. For example, if you find int a, b, c; then insert a, b, and c in the symbol table.

Grammar-1: Sub Set of C_Grammar

```
mul_stmt: mul_stmt func_decl
        | func_decl

func_decl→ type_spec term LPAREN RPAREN LCURL stmt RCURL

stmt →stmt unit | unit
unit→var_decl NEWLINE
        | expr_decl NEWLINE


var_decl→type_spec decl_list SEMICOLON

type_spec→  INT
           |FLOAT
           |DOUBLE

decl_list→decl_list COMMA term
          | decl_list COMMA term LTHIRD CONST_INT RTHIRD
          | term
          | term LTHIRD CONST_INT RTHIRD
          |ass_list

ass_list→  term ASSIGNOP expr

expr→  CONST_INT
| CONST_FLOAT
| expr ADDOP expr
| expr MULOP expr
|<add other grammar for other arithmetic and logical operator>
| <add grammar rule for expression with parentheses>
|term

term→  ID
expr_decl-> term ASSIGNOP expr SEMICOLON
```

Bonus: If anyone can incorporate argument list within function declaration, bonus 10 mark will be given to that particular individual.

3 Input

The input will be a text file containing few statements of a C source program.

4 Output

In this assignment, there will be two output files. One is a file containing the symbol table. This file should be named as log.txt. You will output the symbol table containing all the identifiers in this file.

The other file is a log and error file named as log_error.txt. In this file you will output all the actions performed in your program. When a grammar matches the input from the C code, it should print the matching rule in the correct order in log_error.txt file. For each grammar rule matched with some portion of the code, print the rule along with the relevant portion of the code. Print well-formed syntax error messages with line numbers in the same log_error.txt file. Incorporate error recovery in your parser. You need to use Yacc's predefined token error for this purpose.
For more clarification about input output please refer to the sample input output file. You are highly encouraged to produce output exactly like the sample one.

Notes:

Any type of plagiarism is strongly forbidden. -100% marks will be rewarded to the students who will be found to be involved in plagiarism. It does not matter who is the server and who is the client.