Benjamin Scholar
Email: scholar@case.edu
Course: CSDS 337 - Compiler Design
Instructor: Dr. Vipin Chaudhary

Programming Homework - 2
ID: ??
Term: Spring 2023
Due Date: TBD

Number of hours delay for this Problem Set: 0
Cumulative number of hours delay so far: 0

I discussed this homework with: Zachary Baldwin, Matthias Portzel

---

Problem 1: 100%
Using Flex and Bison, implement a lexer and parser for the C subset described by the following context-free grammar:

```
program -> decList | ε
decList -> decList dec | dec
dec -> funDef | funDec

type -> "int" | "float" | "string" | "bool" | "void"
varDec -> type ID
varDecs -> varDecs varDec ; | ε

funDec -> type ID ( params ) ;
funDef -> type ID (params) { varDecs stmts }
params -> paramsList | ε
paramList -> paramList , type ID | type ID

stmt -> exprStmt | { stmts } | selStmt
     | iterStmt | jumpStmt
exprStmt -> expr ; | ;
stmts -> stmts stmt | ε
selStmt -> IF ( expr ) stmt ELSE stmt
        | IF ( expr ) stmt
iterStmt -> while ( expr ) stmt
jumpStmt -> break ; | return ; | return expr ;
expr -> orExpr | ID = expr
orExpr -> andExpr | orExpr OR andExpr
andExpr -> unaryRelExpr | andExpr AND unaryRelExpr
unaryRelExpr -> NOT unaryRelExpr | relExpr
relExpr -> term relop term | term
relop -> > | < | >= | <= | == | !=
term -> factor | term + factor | term - factor
factor -> primary | factor * primary | factor / primary
     | factor % primary
primary -> ID | (expr) | call | constant
call -> ID ( args ) | ID ( )
args -> args, expr | expr
constant -> INT_LIT | FLOAT_LIT | STR_LIT
```

You will need a lexer capable of reading the tokens used in above grammar (including simple ones like semicolons, commas, etc, but also more complicated regular expressions such as those needed for string literals).

Starter code has been provided with a few examples filled in for both the lexer and parser. A makefile has also been provided — run `make` in the directory containing the code to generate the lexer and the parser.

Your completed code should generate a parse tree for any valid input to the program, or otherwise throw a syntax error. Code has been provided to draw the parse trees for you using `graphviz`. The starter code will generate a `parsetree.dot` file, from which you can run the command `dot -Tpng parsetree.dot > tree.png` to create an image of the parse tree. The specific appearance and structure of the parse tree may vary somewhat with your implementation (e.g. the names of nodes may be different), but the overall structure must be correct.

Deliverables: A zip file containing

- **All files from the starter code you were given, plus your changes**
- **A text file marked readme that contains:**
  - **Full name and Case ID**
  - **(not required) any special notes about your implementation the grader should be aware of**