MINI-GO LANGUAGE GRAMMAR

The Mini-GO language is a subset of the GOLang language that retains much of its original syntax, although it is limited not only in syntactic content but also in semantic content. In this sense, code examples from the original language that are applicable to the syntax presented here should be compatible, making it easier to write syntactically correct source code for this language.

PRINCIPAL RULE: “root”

```
root                : 'package'  IDENTIFIER ';' topDeclarationList
topDeclarationList  : ( variableDecl | typeDecl | funcDecl)*
variableDecl        : 'var' singleVarDecl ';'
                    | 'var'  '(' innerVarDecls ')' ';'
                    | 'var'  '(' ')' ';'
innerVarDecls       : singleVarDecl ';' (singleVarDecl ';')*
singleVarDecl       : identifierList declType '=' expressionList
                    | identifierList '=' expressionList
                    | singleVarDeclNoExps
singleVarDeclNoExps : identifierList declType
typeDecl            : 'type' singleTypeDecl ';'
                    | 'type' '(' innerTypeDecls ')' ';'
                    | 'type' '(' ')' ';'
innerTypeDecls      : singleTypeDecl ';' (singleTypeDecl ';')*
singleTypeDecl      : IDENTIFIER declType
funcDecl            : funcFrontDecl block ';'
funcFrontDecl       : 'func' IDENTIFIER '(' (funcArgDecls|epsilon) ')' (declType|epsilon)
funcArgDecls        : singleVarDeclNoExps (',' singleVarDeclNoExps)*
declType            : '(' declType ')'
                    | IDENTIFIER
                    | sliceDeclType
                    | arrayDeclType
```

```
                    | structDeclType
sliceDeclType       : '[' ']' declType
arrayDeclType       : '[' INTLITERAL ']' declType
structDeclType      : 'struct' '{' (structMemDecls|epsilon) '}'
structMemDecls      : singleVarDeclNoExps ';' (singleVarDeclNoExps ';')*
identifierList      : IDENTIFIER (',' IDENTIFIER)*
expression          : primaryExpression
                    | expression  '*' expression
                    | expression '/' expression
                    | expression '%' expression
                    | expression '<<' expression
                    | expression '>>' expression
                    | expression '&' expression
                    | expression '&^ ' expression
                    | expression '+' expression
                    | expression '-' expression
                    | expression '|' expression
                    | expression '^' expression
                    | expression '==' expression
                    | expression '!=' expression
                    | expression '<' expression
                    | expression '<=' expression
                    | expression '>' expression
                    | expression '>=' expression
                    | expression '&&' expression
                    | expression '||' expression
                    | '+' expression
                    | '-' expression
                    | '!' expression
```

```
                    | '^' expression
expressionList      : expression (',' expression)*
primaryExpression   : operand
                    | primaryExpression selector
                    | primaryExpression index
                    | primaryExpression arguments
                    | appendExpression
                    | lengthExpression
                    | capExpression
operand             : literal
                    | IDENTIFIER
                    | '(' expression ')'
literal             : INTLITERAL
                    | FLOATLITERAL
                    | RUNELITERAL
                    | RAWSTRINGLITERAL
                    | INTERPRETEDSTRINGLITERAL
index               : '[' expression ']'
arguments           : '(' expressionList | epsilon ')'
selector            : '.' IDENTIFIER
appendExpression    : 'append' '(' expression ',' expression ')'
lengthExpression    : 'len' '(' expression ')'
capExpression       : 'cap' '(' expression ')'
statementList       : statement*
block               : '{' statementList '}'
statement           : 'print' '(' expressionList | epsilon ')' ';'
                    | 'println' '(' expressionList | epsilon ')' ';'
                    | 'return' (expresión | epsilon) ';'
                    | 'break' ';'
```

```
                        | 'continue' ';'
                        | simpleStatement ';'
                        | block ';'
                        | switch ';'
                        | ifStatement ';'
                        | loop ';'
                        | typeDecl
                        | variableDecl
simpleStatement         : epsilon
                        | expression ('++' | '--' | epsilon)
                        | assignmentStatement
                        | expressionList ':=' expressionList
assignmentStatement     : expressionList '=' expressionList
                        |expression '+=' expression
                        |expression '&=' expression
                        |expression '-=' expression
                        |expression '|=' expression
                        |expression '*=' expression
                        |expression '^=' expression
                        |expression '<<=' expression
                        |expression '>>=' expression
                        |expression '&^=' expression
                        |expression '%=' expression
                        |expression '/=' expression
ifStatement             : 'if' expression block
                        | 'if' expression block 'else' ifStatement
                        | 'if' expression block 'else' block
                        | 'if' simpleStatement  ';' expression block
                        | 'if' simpleStatement ';'  expression block 'else' ifStatement
```

```
                        | 'if' simpleStatement  ';' expression block 'else' block
loop                    : 'for' block
                        | 'for' expression block
                        | 'for' simpleStatement ';' expression ';' simpleStatement block
                        | 'for' simpleStatement ';' ';' simpleStatement block
switch                  : 'switch' simpleStatement ';' expression '{' expressionCaseClauseList '}'
                        | 'switch' expression '{' expressionCaseClauseList '}'
                        | 'switch' simpleStatement ';' '{' expressionCaseClauseList '}'
                        | 'switch' '{' expressionCaseClauseList '}'
expressionCaseClauseList : epsilon
                        | expressionCaseClause expressionCaseClauseList
expressionCaseClause   : expressionSwitchCase ':' statementList
expressionSwitchCase : 'case' expressionList
                        | 'default'
```

Aclarations

1. The grammar is mostly represented in EBNF, although there may be BNF-style recursions and possibilities of factorization.
2. Tokens are represented in single quotes, like 'token'.
3. Epsilon is a metasymbol that denotes that the element can be empty. This symbol in each parser generator has its particular way of being expressed (check for ANTLR4).
4. Identifiers, integer literals, and floating-point literals (IDENTIFIER, INTLITERAL, and FLOATLITERAL) follow the GoLang standard.
5. Character literals also follow the GoLang standard and are denoted by RUNELITERAL.
6. String literals also follow the standard but in two specific formats known in GoLang as "raw string" (RAWSTRINGLITERAL) and "interpreted string" (INTERPRETEDSTRINGLITERAL).