```
// source: http://media.pragprog.com/titles/tpantlr2/code/examples/Cymbol.g4

/**
 * Simple statically-typed programming language with functions and variables
 * taken from "Language Implementation Patterns" book.
 */

grammar Cymbol;

@header {
package cymbol;
}

prog : (varDecl | functionDecl)* EOF ;

varDecl : type ID ('=' expr)? ';' ;

type : 'int' | 'double' | 'void' ;

functionDecl : type ID '(' formalParameters? ')' block ;

formalParameters : formalParameter (',' formalParameter)* ;

formalParameter : type ID ;

block : '{' stat* '}' ;

stat : block
     | varDecl
     | 'if' expr 'then' stat ('else' stat)?
     | 'return' expr? ';'
     | expr '=' expr ';'
     | expr ';'
     ;

expr: ID '(' exprList? ')'    # Call // function call
    | expr '[' expr ']'       # Index // array subscripts
    | op = '-' expr                # Negate // right association
    | op = '!' expr                # Not // right association
    | <assoc = right> expr '^' expr # Power
    | lhs = expr (op = '*'| op = '/') rhs = expr     # MultDiv
    | lhs = expr (op = '+'| op = '-') rhs = expr     # AddSub
    | lhs = expr (op = '==' | op = '!=') rhs = expr  # EQNE
    | '(' expr ')'            # Parens
    | ID                      # Id
    | INT                     # Int
    ;

exprList : expr (',' expr)* ;
/////////////////////////////////////////////
// You can use "Alt + Insert" to automatically generate
// the following lexer rules for literals in the grammar above.
// Remember to check and modify them if necessary.

SEMI : ';' ;
COMMA : ',' ;
LPAREN : '(' ;
RPAREN : ')' ;
LBRACK : '[' ;
RBRACK : ']' ;
LBRACE : '{' ;
RBRACE : '}' ;

IF : 'if' ;
THEN : 'then' ;
ELSE : 'else' ;
RETURN : 'return' ;

INTTYPE : 'int' ;
DOUBLETYPE : 'double' ;
VOIDTYPE : 'void' ;

ADD : '+' ;
SUB : '-' ;
MUL : '*' ;
DIV : '/' ;

EQ : '=' ;
NE : '!=' ;
EE : '==' ;
////////////////////////////////////////////
WS  : [ \t\n\r]+ -> skip ;
SL_COMMENT : '//' .*? '\n' -> skip ;

ID : LETTER (LETTER | DIGIT)* ;
INT : DIGIT+ ;

fragment LETTER : [a-zA-Z] ;
fragment DIGIT : [0-9] ;
```

```
grammar CymbolCFG;

@header {
package cymbol.cfg;
}

prog : prog decl
     | decl
     ;

decl : varDecl
     | funcDecl
     ;

varDecl : type ID
        | type ID '=' expr
        ;

funcDecl : ' ' ;

type : 'int' | 'double' | 'void' ;

expr : ID ;

ID : [a-z]+ ;
```