```
/*************************************************************************
 Grammar for base programs
 *************************************************************************/
program     ::= declList
            ;

declList    ::= declList decl
            | /* epsilon */
            ;

decl        ::= varDecl
            | fctnDecl
            | tupleDecl       // tuple definitions only at top level
            ;

varDeclList ::= varDeclList varDecl
            | /* epsilon */
            ;

varDecl     ::= type id DOT
            | TUPLE id id DOT
            ;

fctnDecl    ::= type id formals fctnBody
            ;

tupleDecl   ::= TUPLE id LCURLY tupleBody RCURLY DOT
            ;

tupleBody   ::= tupleBody varDecl
            | varDecl
            ;

formals     ::= LCURLY RCURLY
            | LCURLY formalsList RCURLY
            ;

formalsList ::= formalDecl
            | formalDecl COMMA formalsList
            ;

formalDecl  ::= type id                 // note: no tuple parameters
            ;

fctnBody    ::= LSQBRACKET varDeclList stmtList RSQBRACKET
            ;

stmtList    ::= stmtList stmt
            | /* epsilon */
            ;

stmt ::= assignExp DOT
     | fctnCall DOT
     | loc PLUSPLUS DOT
     | loc MINUSMINUS DOT
     | IF exp LSQBRACKET varDeclList stmtList RSQBRACKET
     | IF exp LSQBRACKET varDeclList stmtList RSQBRACKET ELSE LSQBRACKET varDeclList stmtList RSQBRACKET
     | WHILE exp LSQBRACKET varDeclList stmtList RSQBRACKET
     | READ INPUTOP loc DOT
     | WRITE OUTPUTOP exp DOT
     | RETURN exp DOT
     | RETURN DOT
     ;
```

```
assignExp   ::= loc ASSIGN exp
            ;

exp         ::= assignExp
            | exp PLUS exp
            | exp MINUS exp
            | exp TIMES exp
            | exp DIVIDE exp
            | exp EQUALS exp
            | exp NOTEQUALS exp
            | exp GREATER exp
            | exp GREATEREQ exp
            | exp LESS exp
            | exp LESSEQ exp
            | exp AND exp
            | exp OR exp
            | NOT exp
            | MINUS term
            | term
            ;

term        ::= loc
            | TRUE
            | FALSE
            | INTLITERAL
            | STRLITERAL
            | LPAREN exp RPAREN
            | fctnCall
            ;

fctnCall    ::=  id LPAREN RPAREN              // fctn call with no args
            | id LPAREN actualList RPAREN      // fctn call with args
            ;

actualList  ::= exp
            | actualList COMMA exp
            ;

type        ::= INTEGER
            | LOGICAL
            | VOID
            ;

loc         ::= id
            | loc COLON id
            ;

id          ::= ID
            ;
```