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

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

decl          ::= varDecl
              | structDecl       // struct definitions only at top level
              | funcDecl
              ;

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

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

structDecl    ::= STRUCT id LSQUARE structBody RSQUARE
              ;

structBody    ::= structBody varDecl
              | varDecl
              ;

funcDecl      ::= type id formals funcBody
              ;

formals       ::= LSQUARE RSQUARE
              | LSQUARE formalsList RSQUARE
              ;

formalsList   ::= formalDecl
              | formalDecl COMMA formalsList
              ;

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

funcBody      ::= LSQUARE varDeclList stmtList RSQUARE
              ;

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

stmt          ::= assignExp DOT
              | loc PLUSPLUS DOT
              | loc MINUSMINUS DOT
              | funcCall DOT
              | RETURN exp DOT
              | RETURN DOT
              | INPUT READOP loc DOT
              | DISPLAY WRITEOP exp DOT
              | WHILE LPAREN exp RPAREN LCURLY varDeclList stmtList RCURLY
              | IF LPAREN exp RPAREN LCURLY varDeclList stmtList RCURLY
              | IF LPAREN exp RPAREN LCURLY varDeclList stmtList RCURLY ELSE LCURLY varDeclList
stmtList RCURLY
              ;
```

```
assignExp     ::= loc ASSIGN exp
              ;

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

term          ::= loc
              | INTLIT
              | STRINGLIT
              | TRUE
              | FALSE
              | LPAREN exp RPAREN
              | funcCall
              ;

funcCall      ::=  id LPAREN RPAREN            // function call with no args
              | id LPAREN actualList RPAREN    // function call with args
              ;

actualList    ::= exp
              | actualList COMMA exp
              ;

type          ::= BOOLEAN
              | INTEGER
              | VOID
              ;

loc           ::= id
              | loc COLON id
              ;

id            ::= ID
              ;
```