```
program    ::= declList

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

decl ::= varDecl
       | fnDecl

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

varDecl ::= type id SEMICOLON

fnDecl ::= type id formals fnBody

formals ::= LPAREN RPAREN
          | LPAREN formalsList RPAREN

formalsList ::= formalDecl
              | formalDecl COMMA formalsList

formalDecl  ::= type id

type ::= INT

fnBody ::= LCURLY varDeclList stmtList RCURLY

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

stmt ::= IF LPAREN exp RPAREN LCURLY varDeclList stmtList RCURLY
      | IF LPAREN exp RPAREN LCURLY varDeclList stmtList RCURLY
                    ELSE LCURLY varDeclList stmtList RCURLY
      | WHILE LPAREN exp RPAREN
                    LCURLY varDeclList stmtList RCURLY
      | RETURN exp SEMICOLON
      | RETURN SEMICOLON
      | fncall SEMICOLON
      | FOR LPAREN forStmt SEMICOLON exp SEMICOLON forStmt
                    RPAREN LCURLY varDeclList stmtList RCURLY
      | assign SEMICOLON
      | readStmt
      | writeStmt

readStmt ::= CIN INPUTSIGN id SEMICOLON

writeStmt ::= COUT OUTPUTSIGN exp SEMICOLON


assign ::= loc ASSIGN exp

forStmt ::= assign
```

```
          | /* epsilon */

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

term ::= loc
       | INTLITERAL
       | STRINGLITERAL
       | fncall

fncall ::= id LPAREN RPAREN
         | id LPAREN actualList RPAREN

actualList ::= exp
             | actualList COMMA exp

loc ::= id

id ::= ID
```