```
Context Free Grammar Vital Statistics Checker (version 0.00)
---- Please report bugs and problems to robin@cpsc
============================================================

Your grammar is:


prog ->  stmt.

stmt ->  IF expr THEN stmt ELSE stmt
       | WHILE expr DO stmt
       | DO stmt UNTIL expr
       | READ ID
       | ID ASSIGN expr
       | PRINT expr
       | BEGIN stmtlist END.

stmtlist ->  stmtlist stmt SEMICOLON
           |.

expr ->  expr addop term
       | term.

addop ->  ADD
        | SUB.

term ->  term mulop factor
       | factor.

mulop ->  MUL
        | DIV.

factor ->  LPAR expr RPAR
         | ID
         | NUM
         | SUB NUM.


All nonterminals are reachable and realizable.


The nullable nonterminals are:

 stmtlist.

The endable nonterminals are:

 factor term expr prog stmt.

The following nonterminals are left recursive:
 stmtlist expr term

The grammar is not LL(1).



The first sets are:

stmt ====> { IF WHILE DO READ ID PRINT BEGIN }
addop ====> { ADD SUB }
mulop ====> { MUL DIV }
factor ====> { LPAR ID NUM SUB }
prog ====> { IF WHILE DO READ ID PRINT BEGIN }
stmtlist ====> { IF WHILE DO READ ID PRINT BEGIN }
term ====> { LPAR ID NUM SUB }
```

```
expr ====> { LPAR ID NUM SUB }


The follow sets are:

mulop ====> { LPAR ID NUM SUB }
factor ====> { MUL DIV ELSE UNTIL SEMICOLON THEN DO ADD SUB RPAR }
addop ====> { LPAR ID NUM SUB }
term ====> { MUL DIV ELSE UNTIL SEMICOLON THEN DO ADD SUB RPAR }
stmtlist ====> { END IF WHILE DO READ ID PRINT BEGIN }
expr ====> { ELSE UNTIL SEMICOLON THEN DO ADD SUB RPAR }
stmt ====> { ELSE UNTIL SEMICOLON }




=====================================================================

                TRANSFORMING THE GRAMMAR
                to remove left recursion.

=====================================================================

Your grammar is:


prog ->  stmt.

stmt ->  IF expr THEN stmt ELSE stmt
       | WHILE expr DO stmt
       | DO stmt UNTIL expr
       | READ ID
       | ID ASSIGN expr
       | PRINT expr
       | BEGIN stmtlist END.

stmtlist ->  stmtlist+.

stmtlist+ ->  stmt SEMICOLON stmtlist+
            |.

expr ->  term expr+.

expr+ ->  addop term expr+
        |.

addop ->  ADD
        | SUB.

term ->  factor term+.

term+ ->  mulop factor term+
        |.

mulop ->  MUL
        | DIV.

factor ->  LPAR expr RPAR
         | ID
         | NUM
         | SUB NUM.


All nonterminals are reachable and realizable.
```

The nullable nonterminals are:

stmtlist+ expr+ term+ stmtlist.

The endable nonterminals are:

term+ factor expr+ term expr prog stmt.

The grammar has no left recursive nonterminals.

The grammar is LL(1).

The first sets are:

```
stmt ====> { IF WHILE DO READ ID PRINT BEGIN }
addop ====> { ADD SUB }
mulop ====> { MUL DIV }
factor ====> { LPAR ID NUM SUB }
prog ====> { IF WHILE DO READ ID PRINT BEGIN }
stmtlist+ ====> { IF WHILE DO READ ID PRINT BEGIN }
expr+ ====> { ADD SUB }
term ====> { LPAR ID NUM SUB }
term+ ====> { MUL DIV }
stmtlist ====> { IF WHILE DO READ ID PRINT BEGIN }
expr ====> { LPAR ID NUM SUB }
```

The follow sets are:

```
mulop ====> { LPAR ID NUM SUB }
term+ ====> { ADD SUB ELSE UNTIL SEMICOLON THEN DO RPAR }
factor ====> { MUL DIV ADD SUB ELSE UNTIL SEMICOLON THEN DO RPAR }
addop ====> { LPAR ID NUM SUB }
expr+ ====> { ELSE UNTIL SEMICOLON THEN DO RPAR }
term ====> { ADD SUB ELSE UNTIL SEMICOLON THEN DO RPAR }
stmtlist ====> { END }
stmtlist+ ====> { END }
expr ====> { ELSE UNTIL SEMICOLON THEN DO RPAR }
stmt ====> { ELSE UNTIL SEMICOLON }
```