```
program:            functions
                    ;

functions:          /* empty */
                |   function functions
                ;

function:           FUNCTION IDENT SEMICOLON BEGIN_PARAMS declarations
END_PARAMS BEGIN_LOCALS declarations END_LOCALS BEGIN_BODY statement
SEMICOLON statements END_BODY

                ;

The rule below identifies a single declaration followed by a semicolon
followed by any number of declarations including 0 declarations. Example
code that this would identify:
  n : integer;
  i, j, k: integer;
  t : array [20] of integer;

declarations:       /* empty */
                |   declaration SEMICOLON declarations
                ;

declaration:        IDENT idents COLON ARRAY L_SQUARE_BRACKET NUMBER
R_SQUARE_BRACKET OF INTEGER
                |   IDENT idents COLON INTEGER
                ;

The rule below matches any number of identifiers that follow a single
identifier.  Example code that this rule matches:
, j, k
idents:             /* empty */
                |   COMMA IDENT idents
                ;
The rule below matches a statement followed by a semicolon followed by any
number of statements including 0.  Example code that this rule matches:
k := k + 1;11111111111111111111111111
statements:         /* empty */
                |   statement SEMICOLON statements
                ;
```

```
The rule below matches any statement. Example code that this rule matches:
k := 0; matches var ASSIGN expression
statement:          var ASSIGN expression
                |   IF bool_expr THEN statement SEMICOLON statements ENDIF
                |   IF bool_expr THEN statement SEMICOLON statements ELSE
statement SEMICOLON statements ENDIF
                |   WHILE bool_expr BEGINLOOP statement SEMICOLON
statements ENDLOOP
                |   DO BEGINLOOP statement SEMICOLON statements ENDLOOP
WHILE bool_expr
                |   READ var vars
                |   WRITE var vars
                |   CONTINUE
                |   RETURN expression
                ;

bool_expr:          relation_and_expr bool_expressions
                ;

bool_expressions:   /* empty */
                |   OR relation_and_expr bool_expressions
                ;

relation_and_expr:      relation_expr relation_and_expressions
                    ;

relation_and_expressions:   /* empty */
                        |   AND relation_expr relation_and_expressions
                        ;
relation_expr:              expression comp expression
                        |   TRUE
                        |   FALSE
                        |   L_PAREN bool_expr R_PAREN
                        |   NOT expression comp expression
                        |   NOT TRUE
                        |   NOT FALSE
                        |   NOT L_PAREN bool_expr R_PAREN
                        ;
The rule below matches any comparison operator.  Example code that this
rule matches:
==
comp:               EQ
```

```
                |   NEQ
                |   LT
                |   GT
                |   LTE
                |   GTE
                ;

expression:         multiplicative_expr expression_loop
                ;

expression_loop:    /* empty */
                |   ADD multiplicative_expr expression_loop
                |   SUB  multiplicative_expr expression_loop
                ;

The rule below matches any number of occurrences of expressions that is
greater than or equal to 1.  Example code:
i * 2
expressions:        expression COMMA expressions
                |   expression
                |   /* empty */
                ;

terms:              /* empty */
                |   MOD term terms
                |   DIV term terms
                |   MULT term terms
                ;

multiplicative_expr:        term terms
                        ;

term:               IDENT L_PAREN expressions R_PAREN
                |   NUMBER
                |   var
                |   L_PAREN expression R_PAREN
                |   UMINUS NUMBER
                |   UMINUS var
                |   UMINUS L_PAREN expression R_PAREN
                |   UMINUS IDENT L_PAREN expressions R_PAREN
                ;
```

```
var:                IDENT
                |   IDENT L_SQUARE_BRACKET expression R_SQUARE_BRACKET
                ;
The rule below matches any number of variables including 0 that follow a
var.  Example code that the rule below matches:
, j, k
vars:               /* empty */
                |   COMMA var vars
                ;
```