Syntax-directed scheme

```
moudle                  --> "MODULE" identifier:i {module = new
Module(i.getValue()); }";"
                                declarations
                             begin
                             "END" "IDENTIFIER" "."
begin                   --> "BEGIN" statement_sequence
declarations            --> const_declaration type_declaration
var_declaration
                            procedure_declarations
const_declaration       --> "CONST" const_expression ";"
const_expression_prefix --> identifier "=" expression
const_expression_suffix --> ";" const_expression
const_expression        --> const_expression_prefix const_expression_suffix
type_declaration        --> type type_expression
type_expression_prefix --> identifier "=" type ";"
type_expression_suffix --> type_expression
type_expression        --> type_expression_prefix type_expression_suffix
var_declaration        --> "VAR" var_expression
var_expression_suffix  --> ":" type ";"
var_expression_prefix  --> var_expression
var_expression         --> identifier_list var_expression_prefix
var_expression_suffix
procedure_declarations --> procedure_declaration ";" procedure_declarations
procedure_declaration  --> procedure_heading:h
                           { scopeStack.push(new
Scope(module.add(h.getValue()))); }
                           ";"
                           procedure_body
procedure_body         --> declarations
                           begin
                           "END" identifier
procedure_heading      --> {
                             if (lookahead.getType() == Type.IDENTIFIER) {
                                _procedure = new
MyProcedure(lookahead.getValue());
                                res += match(Type.IDENTIFIER);
                             }
                            }
                            identifier formal_parameters
                            {_procedures.add(_procedure); _procedure =
null;}
formal_parameters      --> "(" fp_sections ")"
                       |   {throw new MissingLeftParenthesisException(); }
fp_sections            --> fp_section fp_sections_suffix
```

```
fp_sections_suffix      --> ";" fp_sections
fp_section              --> ["VAR"] identifier_list ":" type
                            {
                               ArrayList<String> l = _procedure.getTypes();
                               for (int i = 0; i < l.size(); ++i) {
                               if (l.get(i) == "") {
                                   l.set(i, tmp);
                               }
                             }
type                     --> array_type
                         |   identifier
                         |   record_type
field_list               --> identifier_list ":" type
field_list_with_semi     --> ";" field_list field_list_with_semi
array_type               --> "ARRAY" expression "OF" type
identifier_list          -->
                      { if (lookahead.getType() == Type.IDENTIFIER &&
_procedure != null) {
                            _procedure.getTypes().add("");
                        }
                       }
                       identifier identifier_list_with
identifier_list_with     -->
                      {if (lookahead.getType() == Type.IDENTIFIER &&
_procedure != null) {
                             _procedure.getTypes().add("");
                        }
                      }
                      identifier identifier_list_with
statement_sequence      --> statement statement_with_semi
statement_with_semi     --> ";" statement statement_with_semi
statement               --> if_statement
                         |  while_statement
                         |  identifier statement_suffix
                            {scopeStack.peek().add(new
PrimitiveStatement(res));}
statement_suffix        --> procedure_call
                         |  assignment
while_statement         --> "WHILE" expression:e
                            {While Statement whileSmt = new
WhileStatement(e.getValue())}
                            "DO" {scopeStatcik.push(new
Scope(whileSmt.getLoopBody()))}
                            statement_sequence
                            "END" {scopeStack.pop()}
if_statement            --> "IF" expression:e
                            {IfStatement ifSmt = new
IfStatement(e.getValue())}
                            {scopeStack.peek().add(ifSmt)}
```

```
                           {scopeStack.push(new
Scope(ifSmt.getFalseBody()))}
                           {scopeStack.push(new
Scope(ifSmt.getTrueBody()))}
                           "THEN" statement_sequence
                           {scopeStack.pop()}
                           elsif_statement else_statement "END"
elsif_statement        --> "ELSIF" expression
                           {IFStatement if Smt = new
IfStatement(e.getValue())}
                           {scopeStack.peek().add(ifSmt)}
                           {scopeStack.push(new
Scope(ifSmt.getTrueBody()))}
                           "THEN" statement_sequence
                           {scopeStack.pop()}
                           {scopeStack.pop()}
                           {scopeStack.push(new
Scope(ifSmt.getFalseBody()))}
                           elsif_statement
else_statement         --> "ELSE" statement_sequence {scopeStack.pop()}
procedure_call         --> actual_parameters:a
                           {
     if
(getFirst("actual_parameters").contains(Type.terminalNames[lookahead.getTyp
e()]))
        {
            numArgument = 0;
        }
        _procedureCalls.add(new ProcedureCall(name, numArgument));
                           }
actual_parameters      --> "(" expression_list ")"
assignment             --> simple_expression comp_expression
expression_list        --> expression:e
                           {
       if
(getFirst("expression").contains(Type.terminalNames[lookahead.getType()]))
{
            if (! e.string.equals("")) {
                numArgument++;
            }
        }
                           }
                           expression_list_with
expression_list_with   --> "," expression:e {if (! e.string.equasl("")) ++
numArgument}
                           expression_list_with
comp                   --> "<"
                        |  "<="
                        |  ">"
                        |  ">="
                        |  "#"
                        |  "="
comp_expression        --> comp simple_expression
unary                  --> "+"
                        |  "-"
binary_low             --> "OR"
                        |  "+"
                        |  "-"
binary_mid             --> "*"
                        |  "DIV"
                        |  "MOD"
                        |  "&"
simple_expression      --> unary term term_list_with
term_list_with         --> binary_low term term_list_with
term                   --> factor term_suffix
term_suffix            --> binary_mid factor
factor                 --> "~" factor
                        |  number
                        |  identifier selector
                        |  "(" expression ")"
number                 --> integer
selector               --> "[" expression "]" selector
                        |  "." identifier selector
```