```
grammar Grammar;

import Tokenizer;

@header {
    import ast.*;
    import ast.datatype.*;
    import ast.expression.*;
    import ast.stmt.*;
}

program returns[Program ast]:
    classDef global? create features 'end' runInvoc EOF
    {$ast = new Program(
        $classDef.ast,
        ($ctx.global != null ? $global.ast : null),
        $create.ast,
        $features.list,
        $runInvoc.ast
    );}
;

classDef returns[ClassDef ast]:
    'class' IDENT ';'
    {$ast = new ClassDef($IDENT);}
;

runInvoc returns[RunInvocation ast]:
    'run' procedure ';'
    { $ast = new RunInvocation($procedure.ast); }
;

stmts returns[List<Stmt> list = new ArrayList<Stmt>()]:
    (stmt {$list.add($stmt.ast);})+
;
stmt returns[Stmt ast]:
    'read' args ';'
    {$ast = new ReadStmt(($args.list != null ? $args.list : new ArrayList<>()));}
    | ('print' args? ';'
    {$ast = new PrintStmt(($ctx.args != null ? $args.list : new ArrayList<>()), "");}
    )
    | ('println' args? ';'
    {$ast = new PrintStmt(($ctx.args != null ? $args.list : new ArrayList<>()),
    "ln");} )
    | assign
    {$ast = new AssignStmt($assign.ast);}
    | 'if' expression 'then' ifstmt = stmts ('else' elsestmt = stmts)? 'end'
    { $ast = new IfStmt(
        $expression.ast,
        $ifstmt.list,
        ($ctx.elsestmt != null ? $elsestmt.list : new ArrayList<>())
    );}
    | ('from' assigns? )? 'until' expression 'loop' stmts 'end'
    { $ast = new FromStmt(
        ($ctx.assigns != null ? $assigns.list : new ArrayList<>()),
        $expression.ast,
        $stmts.list
    );}
    | procedure ';'
    {$ast = new ProcedureStmt($procedure.ast);}
    | returnInvoc
    {$ast = new ReturnStmt($returnInvoc.ast);}
;

assigns returns[List<Assignment> list = new ArrayList<Assignment>()]:
    (assign {$list.add($assign.ast);})+
;
assign returns[Assignment ast]:
    left = expression ':=' right = expression ';'
    { $ast = new Assignment(
        $left.ast,
        $right.ast
```

```
71      );}
72  ;
73
74  procedure returns[Procedure ast]:
75      IDENT op='(' args? cp=')'
76      { $ast = new Procedure(
77          $IDENT.text,
78          ($ctx.args != null ? $args.list : new ArrayList<>())
79      );
80      $ast.updatePositions($IDENT,$op,($ctx.args != null ? $args.list : null),$cp);}
81  ;
82
83  expression returns[Expression ast]:
84      value = INT_LITERAL
85      { $ast = new IntLiteral($value); }
86      | value = REAL_LITERAL
87      { $ast = new RealLiteral($value); }
88      | value = CHAR_LITERAL
89      { $ast = new CharLiteral($value); }
90      | name = IDENT
91      { $ast = new Variable($name.text); $ast.updatePositions($name);}
92      | procedure
93      { $ast = new ProcedureExpression($procedure.ast); }
94      | exprArray = expression '[' index = expression ']'
95      { $ast = new ArrayExpression($exprArray.ast, $index.ast); }
96      | expStruct = expression '.' IDENT
97      { $ast = new StructExpression($expStruct.ast, $IDENT); }
98      | '-' expression
99      { $ast = new MinusExpression($expression.ast); }
100     | '(' expression ')'
101     { $ast = $expression.ast; }
102     | 'not' expression
103     { $ast = new NotExpression($expression.ast); }
104     | 'to' '<' dataType '>' '(' expression ')'
105     { $ast = new Cast($dataType.ast, $expression.ast);}
106     | left = expression operator = ('*' | '/' | 'mod') right = expression
107     { $ast = new ArithmeticExpression($left.ast, $operator.text, $right.ast); }
108     | left = expression operator = ('+' | '-') right = expression
109     { $ast = new ArithmeticExpression($left.ast, $operator.text, $right.ast); }
110     | left = expression operator = ('<' | '>' | '<=' | '>=') right = expression
111     { $ast = new ComparisonExpression($left.ast, $operator.text, $right.ast); }
112     | left = expression operator = ('=' | '<>') right = expression
113     { $ast = new ComparisonExpression($left.ast, $operator.text, $right.ast); }
114     | left = expression operator = 'and' right = expression
115     { $ast = new LogicExpression($left.ast, $operator.text, $right.ast); }
116     | left = expression operator = 'or' right = expression
117     { $ast = new LogicExpression($left.ast, $operator.text, $right.ast); }
118 ;
119
120 args returns[List<Expression> list = new ArrayList<Expression>()]:
121     expression {$list.add($expression.ast);} (',' expression
        {$list.add($expression.ast);} )*
122 ;
123
124 create returns[Create ast]:
125     c='create' idents {$ast = new Create($idents.list );
        $ast.updatePositions($c,$idents.list);}
126 ;
127
128 idents returns[List<String> list = new ArrayList<String>()]:
129     (IDENT ';' {$list.add($IDENT.text);})+
130 ;
131
132 varsDefinitions returns[List<VarDefinition> list = new ArrayList<VarDefinition>()]:
133   (varListDefinition { $list.addAll($varListDefinition.list);})*
134 ;
135
136 varListDefinition returns[List<VarDefinition> list = new ArrayList<VarDefinition>()]:
137     varListIdents ':' dataType ';'
138      { for (int i = 0; i < $varListIdents.list.size(); i++) $list.add(
139       new VarDefinition($varListIdents.list.get(i),$dataType.ast)
140       );}
```

```
141   ;
142
143   varListIdents returns[List<String> list = new ArrayList<>()]:
144       (IDENT {$list.add($IDENT.text);}) (',' IDENT {$list.add($IDENT.text);})*
145   ;
146
147   fields returns[List<Field> list = new ArrayList<>()]:
148       (field {$list.add($field.ast);})*
149   ;
150
151   field returns[Field ast]:
152       IDENT ':' dataType ';'
153       {$ast = new Field($IDENT.text, $dataType.ast);}
154   ;
155
156   dataType returns[DataType ast]:
157       intType = 'INTEGER'
158       {$ast = new IntegerType(); $ast.updatePositions($intType);}
159       |  doubleType = 'DOUBLE'
160       {$ast = new DoubleType(); $ast.updatePositions($doubleType);}
161       |  characterType = 'CHARACTER'
162       {$ast = new CharacterType(); $ast.updatePositions($characterType);}
163       |  name = IDENT
164       {$ast = new StructType($name);}
165       | '[' INT_LITERAL ']' dataType
166       {$ast = new ArrayType($INT_LITERAL.text, $dataType.ast);}
167   ;
168
169   features returns[List<Feature> list = new ArrayList<Feature>()]:
170       (feature {$list.add($feature.ast);})+
171   ;
172   feature returns[Feature ast]:
173       'feature' IDENT params? (':' dataType)? 'is' localBlock? doBlock 'end'
174       {$ast = new Feature(
175           $IDENT.text,
176           ($ctx.params != null ? $params.list : new ArrayList<>()),
177           ($ctx.dataType != null ? $dataType.ast : null),
178           ($ctx.localBlock != null ? $localBlock.ast : null),
179           $doBlock.ast
180       ) ;}
181   ;
182
183   params returns[List<VarDefinition> list = new ArrayList<VarDefinition>()]:
184       '(' IDENT ':' dataType {$list.add(new VarDefinition($IDENT,$dataType.ast));}
185       (',' IDENT ':' dataType {$list.add(new VarDefinition($IDENT,$dataType.ast));} )*
          ')'
186   ;
187
188   localBlock returns[LocalBlock ast]:
189       'local' varsDefinitions
190       {$ast = new LocalBlock(
191           ($ctx.varsDefinitions != null ? $varsDefinitions.list : new ArrayList<>())
192       );}
193   ;
194   doBlock returns[DoBlock ast]:
195       'do' stmts?
196       {$ast = new DoBlock(
197           ($ctx.stmts != null ? $stmts.list : new ArrayList<>())
198       );}
199   ;
200   returnInvoc returns[ReturnInvoc ast]:
201       'return' expression? ';'
202       {$ast = new ReturnInvoc(
203           ($ctx.expression != null ? $expression.ast : null)
204       );}
205   ;
206
207   global returns[Global ast]:
208       'global' globalTypes? varsTypes?
209       {$ast = new Global(
210           ($ctx.globalTypes != null ? $globalTypes.ast : null),
211           ($ctx.varsTypes != null ? $varsTypes.ast : null)
```

```
212         );}
213     ;
214
215     globalTypes returns[GlobalTypes ast]:
216         'types' deftuples?
217         {$ast = new GlobalTypes(
218             ($ctx.deftuples != null ? $deftuples.list : new ArrayList<>())
219         );}
220     ;
221
222     varsTypes returns[VarsTypes ast]:
223         'vars' varsDefinitions
224         {$ast = new VarsTypes(
225             ($ctx.varsDefinitions != null ? $varsDefinitions.list : new ArrayList<>())
226         );}
227     ;
228
229     deftuples returns[List<Deftuple> list = new ArrayList<Deftuple>()]:
230         (deftuple {$list.add($deftuple.ast);})+
231     ;
232     deftuple returns[Deftuple ast]:
233         'deftuple' IDENT 'as' fields 'end'
234         {$ast = new Deftuple(
235             $IDENT.text,
236             ($ctx.fields != null ? $fields.list : new ArrayList<>())
237         );}
238     ;
```