```
grammar Grammar;

import Lexer;

program returns[Program ast]
    : definitions EOF
                                                    { $ast = new
Program($definitions.list); }
    ;

definitions returns[List<Definition> list = new ArrayList<Definition>()]
      : (definition { $list.add($definition.ast); })*
      ;

definition returns[Definition ast]
    : varDefinition
                                                        { $ast =
$varDefinition.ast; }
    | 'struct' IDENT '{' attrDefinitions '}'
                                        { $ast = new
StructDefinition($IDENT, $attrDefinitions.list); }
      | IDENT '(' params ')' (':' type)? '{' varDefinitions statements
'}'                          { $ast = new FunctionDefinition($IDENT,
$params.list, $type.ctx != null ? $type.ast : null, $varDefinitions.list,
$statements.list); }
    ;

attrDefinitions returns[List<AttrDefinition> list = new
ArrayList<AttrDefinition>()]
      : (attrDefinition { $list.add($attrDefinition.ast); })*
      ;

attrDefinition returns[AttrDefinition ast]
    : IDENT ':' type ';'
                                                    { $ast = new
AttrDefinition($IDENT, $type.ast); }
    ;

params returns[List<VarDefinition> list = new ArrayList<VarDefinition>()]
    : (param { $list.add($param.ast); } (',' param {
$list.add($param.ast); })*)?
      ;

param returns[VarDefinition ast]
    : IDENT ':' type
                                                    { $ast = new
VarDefinition($IDENT, $type.ast); }
      ;

varDefinitions returns[List<VarDefinition> list = new
ArrayList<VarDefinition>()]
      : (varDefinition { $list.add($varDefinition.ast); })*
      ;
```

```
varDefinition returns[VarDefinition ast]
    : 'var' IDENT ':' type ';'
                                                    { $ast = new
VarDefinition($IDENT, $type.ast); }
    ;

statements returns[List<Statement> list = new ArrayList<Statement>()]
      : (statement { $list.add($statement.ast); })*
      ;

statement returns[Statement ast]
    : 'read' expression ';'
                                                    { $ast = new
Read($expression.ast); }
      | 'print' expressions ';'
                                                    { $ast = new
Print($expressions.list); }
      | 'println' expressions ';'
                                                    { $ast = new
Println($expressions.list); }
      | 'printsp' expressions ';'
                                                    { $ast = new
Printsp($expressions.list); }
      | 'return' expression? ';'
                                                    { $ast = new
Return($expression.ctx != null ? $expression.ast : null);
$ast.updatePositions($ctx.start); }
      | IDENT '(' expressions ')' ';'?
                                                { $ast = new
FunctionCallStatement($IDENT, $expressions.list); }
    | left=expression '=' right=expression ';'
                                                { $ast = new
Assignment($left.ast, $right.ast); }
    | 'while' '(' expression ')' '{' statements '}'
                                            { $ast = new
While($expression.ast, $statements.list); }
      | 'if' '(' cond=expression ')' '{' tr=statements '}' ('else' '{'
fs=statements '}')?    { $ast = new Ifelse($cond.ast, $tr.list, $fs.ctx
!= null ? $fs.list : null); }
    ;

expressions returns[List<Expression> list = new ArrayList<Expression>()]
    : (expression { $list.add($expression.ast); } (',' expression {
$list.add($expression.ast); })*)?
      ;

expression returns[Expression ast]
    : INT_LITERAL
                                                    { $ast = new
IntLiteral($INT_LITERAL); }
    | REAL_LITERAL
                                                          { $ast = new
FloatLiteral($REAL_LITERAL); }
```

```
    | CHAR_LITERAL
                                                                { $ast = new
CharLiteral($CHAR_LITERAL); }
      | IDENT '(' expressions ')'
                                                          { $ast = new
FunctionCallExpression($IDENT, $expressions.list); }
      | IDENT
                                                                    { $ast =
new Variable($IDENT); }
      | expression1=expression '[' expression2=expression ']'
                                          { $ast = new
ArrayAccess($expression1.ast, $expression2.ast); }
      | expr=expression '.' IDENT
                                                          { $ast = new
FieldAccess($expr.ast, $IDENT); }
      | '!' expression
                                                          { $ast = new
Not($expression.ast); }
      | left=expression op=('*' | '/' | '%') right=expression
                                          { $ast = new
Arithmetic($left.ast, $op, $right.ast); }
      | left=expression op=('+' | '-') right=expression
                                          { $ast = new
Arithmetic($left.ast, $op, $right.ast); }
      | left=expression op=('>=' | '<=' | '>' | '<') right=expression
                                          { $ast = new Logic($left.ast,
$op, $right.ast); }
      | left=expression op=('==' | '!=' ) right=expression
                                          { $ast = new Logic($left.ast,
$op, $right.ast); }
      | left=expression op='&&' right=expression
                                                { $ast = new
Logic($left.ast, $op, $right.ast); }
      | left=expression op='||' right=expression
                                                { $ast = new
Logic($left.ast, $op, $right.ast); }
      | '(' expression ')'
                                                          { $ast =
$expression.ast; }
      | '<' type '>' '(' expression ')'
                                                { $ast = new
Cast($type.ast, $expression.ast); }
    ;

type returns[Type ast]
      : 'int'
                                                                    { $ast =
new IntType(); $ast.updatePositions($ctx.start); }
      | 'float'
                                                          { $ast = new
FloatType(); $ast.updatePositions($ctx.start); }
      | 'char'
                                                          { $ast = new
CharType(); $ast.updatePositions($ctx.start); }
```

```
      | 'void'
                                                                { $ast = new
VoidType(); $ast.updatePositions($ctx.start); }
      | '[' INT_LITERAL ']' type
                                                          { $ast = new
ArrayType($INT_LITERAL, $type.ast); $ast.updatePositions($ctx.start); }
      | IDENT
                                                                  { $ast =
new StructType($IDENT); $ast.updatePositions($ctx.start); }
      ;
```