SINTACTICO

```
grammar Grammar;

import Tokenizer;

@header {
 import ast.declaraciones.*;
 import ast.tipo.*;
 import ast.sentencia.*;
 import ast.expression.*;
 import ast.*;
}

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


declaraciones returns[List<Declaraciones> list = new
ArrayList<Declaraciones>()]

: (declaracionstructs { $list.add(
$declaracionstructs.ast);}
|declaracionglobales { $list.add(
$declaracionglobales.ast);}
|declaracionfuncion { $list.add( $declaracionfuncion.ast);
}
)+
;

declaracionstructs returns[Declaraciones ast]
: 'struct' IDENT '{' listdeclaraciones '}' { $ast = new
Declaracionstructs($IDENT.text, $listdeclaraciones.list);
}
;
```

```
declaracionglobales returns[Declaraciones ast]
: 'var' definicion ';' { $ast = new
Declaracionglobales($definicion.ast); }
;

declaracionfuncion returns [Declaraciones ast]
: IDENT '(' argumentos ')'(':' tipo)? '{'
listaVariablesLocales sentencias'}'
{ $ast = new Declaracionfuncion($IDENT.text,
$argumentos.list, ($tipo.ctx == null) ? null : $tipo.ast,
$listaVariablesLocales.list, $sentencias.list); }

;
definicion returns[Definicion ast]
: IDENT ':' tipo { $ast = new Definicion($IDENT.text,
$tipo.ast); }
;

listaVariablesLocales returns[List<Definicion> list = new
ArrayList<Definicion>()]
: ( 'var' definicion ';'{$list.add($definicion.ast);})*
;
argumentos returns[List<Definicion> list = new
ArrayList<Definicion>()]
: (definicion {$list.add($definicion.ast);}(',' definicion
{$list.add($definicion.ast);})*)*
;

listdeclaraciones returns[List<Definicion> list = new
ArrayList<Definicion>()]
: (definicion ';' {$list.add($definicion.ast);})*
;
sentencia returns[Sentencia ast]
: 'print' expresiones? ';' { $ast = new
PrintSentencia($expresiones.ctx == null ? new
ArrayList<Expression>() : $expresiones.list); }
```

```
| 'read' expresiones? ';' { $ast = new
ReadSentencia($expresiones.ctx == null ? new
ArrayList<Expression>() : $expresiones.list); }
| 'printsp' expresiones? ';' { $ast = new
PrintspSentencia($expresiones.ctx == null ? new
ArrayList<Expression>() : $expresiones.list); }
| 'println' expresiones? ';' { $ast = new
PrintlnSentencia($expresiones.ctx == null ? new
ArrayList<Expression>() : $expresiones.list); }
| IDENT '(' expresiones? ')' ';'      { $ast = new
FuncionSentencia($IDENT.text, $expresiones.ctx == null ?
new ArrayList<Expression>() : $expresiones.list); }
| left=expr '=' right=expr ';' { $ast = new
AsignacionSentencia($left.ast, $right.ast); }
| 'return' expr? ';' { $ast = new
ReturnSentencia(($expr.ctx == null) ? null : $expr.ast); }
| 'if' '(' expr ')' '{' entonces=sentencias '}' ('else'
'{' otro=sentencias '}')?
{ $ast = new IfSentencia($expr.ast, $entonces.ctx == null
? new ArrayList<Sentencia>() : $entonces.list, $otro.ctx
== null ? new ArrayList<Sentencia>() : $otro.list); }
| 'while' '(' expr ')' '{' entonces=sentencias '}'
{ $ast = new WhileSentencia($expr.ast, $entonces.ctx ==
null ? new ArrayList<Sentencia>() : $entonces.list); }
;
sentencias returns [List<Sentencia> list = new
ArrayList<Sentencia>()]
: (sentencia{$list.add($sentencia.ast);})*
;
expresiones returns [List<Expression> list = new
ArrayList<Expression>()]
: expr{$list.add($expr.ast);} (','
expr{$list.add($expr.ast);})*
;
expr returns[Expression ast]
: INT_LITERAL { $ast = new IntExpresion($INT_LITERAL); }
```

```
| REAL_LITERAL { $ast = new RealExpresion($REAL_LITERAL);
}
| IDENT { $ast = new IdentificadorExpresion($IDENT); }
| CHAR_LITERAL { $ast = new CharExpresion($CHAR_LITERAL);
}
| left=expr '.' IDENT { $ast = new AcederCap($left.ast,
$IDENT.text);}
| acceso=expr '[' indice=expr ']' { $ast = new
AccessoArrayExpresion($acceso.ast, $indice.ast); }
| '(' expr ')' { $ast = new
ParentesisExpresion($expr.ast); }
| '<'tipo'>' '('expr')' { $ast = new
CastExpresion($tipo.ast, $expr.ast); }
| '!' expr { $ast = new NegacionExpresion($expr.ast); }
| left=expr operator=( '*' | '/' | '%') right=expr { $ast
= new ArithmeticExpresion($left.ast, $operator.text,
$right.ast); }
| left=expr operator=( '+' | '-' ) right=expr { $ast = new
ArithmeticExpresion($left.ast, $operator.text,
$right.ast); }
| left=expr operator=( '<' | '>' | '<=' | '>=' )
right=expr { $ast = new LogicExpression($left.ast,
$operator.text, $right.ast); }
| left=expr operator='==' right=expr { $ast = new
LogicExpression($left.ast, $operator.text, $right.ast); }
| left=expr operator='!=' right=expr {$ast = new
LogicExpression($left.ast, $operator.text, $right.ast);}
| left=expr operator='&&' right=expr { $ast = new
BoolExpression($left.ast, $operator.text, $right.ast); }
| left=expr operator='||' right=expr { $ast = new
BoolExpression($left.ast, $operator.text, $right.ast); }
| IDENT '(' expresiones? ')' { $ast = new
FuncionExpresion($IDENT.text, $expresiones.ctx == null ?
null : $expresiones.list); }
;
```

```
tipo returns[Tipo ast]
: var='int' { $ast = new IntTipo($var); }
| var='float' { $ast = new FloatTipo($var); }
| var='char' { $ast = new CharTipo($var); }
| '['INT_LITERAL']' tipo { $ast = new
ArrayTipo($INT_LITERAL, $tipo.ast); }
| var=IDENT { $ast = new StringTipo($var); }
;
```