Gramatica libre de contexto sin la creacion de nodos AST

```
start
       : bloques
       ;

bloques
       : (bloque)* EOF
       ;

bloque
       : definicion_variable_global
       | struct
       | funcion
       ;

definicion_variable_global
       :'var' IDENT ':' tipo ';'
       ;

struct
       : 'struct' IDENT '{' definiciones '}' ';'
       ;

definiciones
       : (definicion_campo_struct)*
       ;

definicion_campo_struct
       : IDENT ':' tipo ';'
       ;

funcion
       : IDENT '(' parametros ')' retorno '{' locales sentencias '}'
       ;

parametros
       : (parametro (',' parametro)*)?
       ;

parametro
       : IDENT ':' tipo
       ;

retorno
       : (':' tipo)
       |
       ;
locales
       : (definicion_variable_local)*
       ;
```

```
definicion_variable_local
	:'var' IDENT ':' tipo ';'
	;

sentencias
	: (sentencia)*
	;

sentencia
	: sentencia_asignacion
	| sentencia_print
	| sentencia_read
	| sentencia_if
	| sentencia_while
	| sentencia_llamada_funcion
	| sentencia_return
	;

sentencia_asignacion
	: expr '=' expr ';'
	;

sentencia_print
	: 'print' expr ';'
	| 'printsp' expr ';'
	| 'println' (expr) ';'
	| 'println' ';'
	;

sentencia_read
	: 'read' expr ';'
	;

sentencia_if
	: 'if' '(' expr ')' '{' sentencias '}'
	| 'if' '(' expr ')' '{' sentencias '}' 'else' '{' sentencias '}'
	;

sentencia_while
	: 'while' '(' expr ')' '{' sentencias '}'
	;

sentencia_llamada_funcion
	: IDENT '(' parametros_llamada ')' ';'
	;

sentencia_return
	: 'return'(expr) ';'
	| 'return' ';'
	;

tipo
	: 'int'
	| 'float'
	| 'char'
	| IDENT
	| array
	;
```

```
array
       : '[' INT_CONSTANT ']' tipo
       ;

expr returns[Expr ast]
       : INT_CONSTANT
       | REAL_CONSTANT
       | CHAR_CONSTANT
       | IDENT
       | '(' expr ')'
       | expr '.' expr
       | '!' expr
       | expr operador expr
       | expr ('['expr']')
       | 'cast' '<' tipo '>' '(' expr ')'
       | IDENT '(' parametros_llamada ')'
       ;


parametros_llamada
       : (expr  (',' expr)*)?
       ;

operador returns[Operador ast]
       : op=('*'|'/')
       | op=('+'|'-')
       | op=('=='|'!=')
       | op=('<'|'>'|'>='|'<=')
       | op='&&'
       | op='||'
       ;
```

Gramatica libre de contexto con la creacion de nodos AST

```
grammar Grammar;
import Lexicon;

@parser::header {
    import ast.*;
}

////////////// Program //////////////
start returns[Program ast]
       : bloques { $ast = new Program($bloques.ast); }
       ;

bloques returns[List<Bloque> ast = new ArrayList<Bloque>()]
       : (bloque { $ast.add($bloque.ast); })* EOF
       ;

bloque  returns[Bloque ast]
       : definicion_variable_global { $ast = $definicion_variable_global.ast;
}
       | struct { $ast = $struct.ast; }
       | funcion { $ast = $funcion.ast; }
       ;

////////////// Variables globales //////////////
definicion_variable_global returns[Definicion_variable_global ast]
       :'var' IDENT ':' tipo ';'{ $ast = new
Definicion_variable_global($IDENT, $tipo.ast); }
       ;

////////////// Struct //////////////
struct returns[Struct ast]
       : 'struct' IDENT '{' definiciones '}' ';'{ $ast = new Struct($IDENT,
$definiciones.ast); }
       ;

definiciones returns[List<Definicion_campo_struct> ast = new
ArrayList<Definicion_campo_struct>()]
       : (definicion_campo_struct { $ast.add($definicion_campo_struct.ast);
})*
       ;

definicion_campo_struct returns[Definicion_campo_struct ast]
       : IDENT ':' tipo ';'{ $ast = new Definicion_campo_struct($IDENT,
$tipo.ast); }
       ;
```

```
////////////// Funcion //////////////
funcion  returns[Funcion ast]
	: IDENT '(' parametros ')' retorno '{' locales sentencias '}' { $ast =
new Funcion($IDENT, $parametros.ast, $retorno.ast, $locales.ast,
$sentencias.ast); }
	;

parametros  returns[List<Parametro> ast = new ArrayList<Parametro>()]
	: (parametro { $ast.add($parametro.ast); } (',' parametro {
$ast.add($parametro.ast); })*)?
	//:(parametro (',' parametro)*)?
	;

parametro  returns[Parametro ast]
	: IDENT ':' tipo { $ast = new Parametro($IDENT, $tipo.ast); }
	;

retorno returns[Tipo ast]
	: (':' tipo { $ast = $tipo.ast; })
	| { $ast = new TipoVoid(); }
	;

////////////// Variables locales //////////////
locales returns[List<Definicion_variable_local> ast = new
ArrayList<Definicion_variable_local>()]
	: (definicion_variable_local{
$ast.add($definicion_variable_local.ast); }  )*
	;

definicion_variable_local returns[Definicion_variable_local ast]
	:'var' IDENT ':' tipo ';'{ $ast = new
Definicion_variable_local($IDENT, $tipo.ast); }
	;

////////////// Sentencias //////////////
sentencias returns[List<Sentencia> ast = new ArrayList<Sentencia>()]
	: (sentencia{ $ast.add($sentencia.ast); }    )*
	;

sentencia returns[Sentencia ast]
	: sentencia_asignacion{ $ast = $sentencia_asignacion.ast; }
	| sentencia_print{ $ast = $sentencia_print.ast; }
	| sentencia_read{ $ast = $sentencia_read.ast; }
	| sentencia_if{ $ast = $sentencia_if.ast; }
	| sentencia_while{ $ast = $sentencia_while.ast; }
	| sentencia_llamada_funcion { $ast = $sentencia_llamada_funcion.ast; }
	| sentencia_return{ $ast = $sentencia_return.ast; }
	;

sentencia_asignacion returns[Sentencia_asignacion ast]
	: expr '=' expr ';'{ $ast = new Sentencia_asignacion($ctx.expr(0).ast,
$ctx.expr(1).ast); }
	;
```

```
sentencia_print returns[Sentencia_print ast]
	: 'print' expr ';'{ $ast = new Sentencia_print($expr.ast, ""); }
	| 'printsp' expr ';'{ $ast = new Sentencia_print($expr.ast, "sp"); }
	| 'println' (expr) ';'{ $ast = new Sentencia_print($expr.ast, "ln"); }
	| 'println' ';' { $ast = new Sentencia_print(null, "\\n"); }
	;

sentencia_read returns[Sentencia_read ast]
	: 'read' expr ';'{ $ast = new Sentencia_read($expr.ast); }
	;

sentencia_if returns[Sentencia_if ast]
	: 'if' '(' expr ')' '{' sentencias '}'{ $ast = new
Sentencia_if($expr.ast, $sentencias.ast, new ArrayList<Sentencia>()); }
	| 'if' '(' expr ')' '{' sentencias '}' 'else' '{' sentencias '}' {
$ast = new Sentencia_if($expr.ast, $ctx.sentencias(0).ast,
$ctx.sentencias(1).ast); }
	;

sentencia_while returns[Sentencia_while ast]
	: 'while' '(' expr ')' '{' sentencias '}'{ $ast = new
Sentencia_while($expr.ast, $sentencias.ast); }
	;

sentencia_llamada_funcion returns[Sentencia_llamada_funcion ast]
	: IDENT '(' parametros_llamada ')' ';' { $ast = new
Sentencia_llamada_funcion($IDENT, $parametros_llamada.ast); }
	;

sentencia_return returns[Sentencia_return ast]
	: 'return'(expr) ';'{ $ast = new Sentencia_return($expr.ast); }
	| 'return' ';'{ $ast = new Sentencia_return(null); }
	;

////////////// Tipos //////////////
tipo  returns[Tipo ast]
	: 'int'{ $ast = new TipoInt(); }
	| 'float' { $ast = new TipoFloat(); }
	| 'char' { $ast = new TipoChar(); }
	| IDENT { $ast = new TipoStruct($IDENT); }
	| array      { $ast = $array.ast; }
	;

array returns[TipoArray ast]
	: '[' INT_CONSTANT ']' tipo{ $ast = new TipoArray($INT_CONSTANT,
$tipo.ast); }
	;
```

```
////////////// Expresiones //////////////
expr returns[Expr ast]
	: INT_CONSTANT{ $ast = new Expr_int($INT_CONSTANT); }
	| REAL_CONSTANT{ $ast = new Expr_real($REAL_CONSTANT); }
	| CHAR_CONSTANT{ $ast = new Expr_char($CHAR_CONSTANT); }
	| IDENT{ $ast = new Expr_ident($IDENT); }
	| '(' expr ')'{ $ast = new Expr_parentesis($expr.ast); }
	| expr '.' expr{ $ast = new Expr_punto($ctx.expr(0).ast,
$ctx.expr(1).ast); }
	| '!' expr {$ast = new Expr_negada(new Operador_logico("!"),
$expr.ast);}
	| expr operador expr{ $ast = new Expr_binaria($ctx.expr(0).ast,
$operador.ast, $ctx.expr(1).ast); }
	| expr ('['expr']'){ $ast = new Expr_vector($ctx.expr(0).ast,
$ctx.expr(1).ast); }
	| 'cast' '<' tipo '>' '(' expr ')'{ $ast = new Expr_cast($tipo.ast,
$expr.ast); }
	| IDENT '(' parametros_llamada ')'{ $ast = new
Expr_llamada_funcion($IDENT, $parametros_llamada.ast); }
	;


parametros_llamada returns[List<Expr> ast = new ArrayList<Expr>()]
	: (expr { int iterador = 0; $ast.add($ctx.expr(iterador).ast); } (','
expr{ iterador++; $ast.add($ctx.expr(iterador).ast); })*)?
	;

////////////// Operadores //////////////
operador returns[Operador ast]
	: op=('*'|'/') { $ast = new Operador_aritmetico($op.text); }
	| op=('+'|'-') { $ast = new Operador_aritmetico($op.text); }
	| op=('=='|'!=') { $ast = new Operador_comparacion($op.text); }
	| op=('<'|'>'|'>='|'<=') { $ast = new Operador_comparacion($op.text);
}
	| op='&&' { $ast = new Operador_logico($op.text); }
	| op='||' { $ast = new Operador_logico($op.text); }
	;
```