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); }

;