# Sintaxis del lenguaje mediante una Gramática Libre de Contexto.

**grammar** Grammar

;

**import** Lexicon

;

@parser::header {

**import** ast.\*;

**import** ast.enumerable.AmbitoVariable;

}

/\* Inicio del programa \*/

start **returns**[Programa ast]

: definiciones *EOF* { $ast = **new** Programa($definiciones.list); }

;

/\* Contenido del programa \*/

definiciones **returns** [List<Definicion> list = new ArrayList<Definicion>()]

: (definicion{$list.add($definicion.ast);})\*

;

/\* Define el contenido del programa: variables, estructuras o funciones \*/

definicion **returns**[Definicion ast]

: defVariable {$ast = $defVariable.ast;}

| defStruct {$ast = $defStruct.ast;}

| defFuncion {$ast = $defFuncion.ast;}

;

/\* Definicion de una variable \*/

defVariable **returns**[DefVariable ast]

: 'var' *IDENT* ':' tipo ';' { $ast = **new** DefVariable($IDENT, $tipo.ast, AmbitoVariable.GLOBAL); }

;

/\* Definicion de una estructura \*/

defStruct **returns**[DefStruct ast]

: 'struct' *IDENT* '{' cuerpoStruct '}' ';' { $ast = **new** DefStruct($IDENT, $cuerpoStruct.list); }

;

/\* Definicion de una funcion \*/

defFuncion **returns**[DefFuncion ast]

: *IDENT* listaParametros (':')? tipoRetorno '{' variables sentencias '}' { $ast = **new** DefFuncion($IDENT, $listaParametros.list, $tipoRetorno.ast, $variables.list, $sentencias.list); }

;

/\* Diferentes tipos reconocidos en el programa \*/

tipo **returns**[Tipo ast]

: *INT* { $ast = **new** TipoEntero(); $ast.setPositions($INT);}

| *FLOAT* { $ast = **new** TipoReal(); $ast.setPositions($FLOAT);}

| *CHAR* { $ast = **new** TipoChar(); $ast.setPositions($CHAR);}

| *IDENT* { $ast = **new** TipoStruct($IDENT); }

| tipoArray { $ast = $tipoArray.ast; }

;

tipoRetorno **returns**[Tipo ast]

: tipo { $ast = $tipo.ast;}

| { $ast = **new** TipoVoid(); }

;

/\* Cuerpo de la estructura que guarda todos los posibles campos \*/

cuerpoStruct **returns**[List<CuerpoStruct> list = new ArrayList<CuerpoStruct>()]

: (*IDENT* ':' tipo ';' { $list.add(**new** CuerpoStruct($IDENT, $tipo.ast)); } )\*

;

/\* Define el tipo array del contenido de los tipos \*/

tipoArray **returns**[Tipo ast]

: '[' *INT\_CONSTANT* ']' tipo { $ast = **new** TipoArray($INT\_CONSTANT, $tipo.ast); }

;

/\* Lista de parametros usada en la funcion -puede estar vacio- \*/

listaParametros **returns**[List<DefVariable> list = new ArrayList<DefVariable>();]

: '(' (paramFuncion{$list.add($paramFuncion.ast);}(',' paramFuncion{$list.add($paramFuncion.ast);})\*) ')'

| '(' ')'

;

/\* Parametros que se pueden pasar a una funcion \*/

paramFuncion **returns**[DefVariable ast]

: *IDENT* ':' tipo { $ast = **new** DefVariable($IDENT, $tipo.ast, AmbitoVariable.PARAM); }

;

/\* Lista de definiciones de variables \*/

variables **returns**[List<DefVariable> list = new ArrayList<DefVariable>();]

: (defVariable { $list.add(**new** DefVariable($defVariable.ast.getIdent(), $defVariable.ast.getTipo(), AmbitoVariable.LOCAL)); })\*

;

/\* Lista de sentencias \*/

sentencias **returns**[List<Sentencia> list = new ArrayList<Sentencia>();]

: (sentencia { $list.add($sentencia.ast); })\*

;

/\* Diferentes tipos de sentencias reconocidas en el programa \*/

sentencia **returns**[Sentencia ast]

: 'return' expresion ';' { $ast = **new** Return($expresion.ast); }

| et='return' ';' { $ast = **new** Return(**null**); $ast.setPositions($et); }

| expresion '=' expresion ';' { $ast = **new** Asignacion($ctx.expresion(0).ast, $ctx.expresion(1).ast); }

| 'printsp' expresion ';' { $ast = **new** Printsp($expresion.ast); }

| 'print' expresion ';' { $ast = **new** Print($expresion.ast); }

| 'println' expresion ';' { $ast = **new** Println($expresion.ast); }

| 'println' ';' { $ast = **new** Println(**null**); }

| 'read' expresion ';' { $ast = **new** Read($expresion.ast); }

| bucleWhile { $ast = $bucleWhile.ast; }

| sentenciaCondicional { $ast = $sentenciaCondicional.ast; }

| *IDENT* parametrosInvocacion ';' { $ast = **new** InvocacionFuncionSentencia($IDENT, $parametrosInvocacion.list); }

;

/\* Diferentes parametros que puede tener una invocacion a una funcion -puede estar vacio- \*/

parametrosInvocacion **returns**[List<Expresion> list = new ArrayList<Expresion>()]

: '(' expresion {$list.add($expresion.ast);} (',' expresion {$list.add($expresion.ast);})\* ')'

| '(' ')'

;

/\* Lista de las diferentes expresiones recogidas en el programa \*/

expresion **returns**[Expresion ast]

: *IDENT* { $ast = **new** Variable($IDENT);}

| *INT\_CONSTANT* { $ast = **new** LiteralInt($INT\_CONSTANT);}

| *REAL\_CONSTANT* { $ast = **new** LiteralReal($REAL\_CONSTANT);}

| *CHAR\_CONSTANT* { $ast = **new** LiteralChar($CHAR\_CONSTANT);}

| '(' ex=expresion ')' { $ast = $ex.ast;}

| ex1=expresion '[' ex2=expresion ']' { $ast = **new** AccesoArray($ex1.ast, $ex2.ast);}

| ex=expresion '.' *IDENT* { $ast = **new** AccesoStruct($ex.ast, $IDENT);}

| 'cast' '<' tipo '>' '(' ex=expresion ')' { $ast = **new** Cast($tipo.ast, $ex.ast);}

| '!' ex=expresion { $ast = **new** Negacion($ex.ast);}

| ex1=expresion op=('\*'|'/') ex2=expresion { $ast = **new** ExpresionAritmetica($ex1.ast, $op, $ex2.ast);}

| ex1=expresion op=('+'|'-') ex2=expresion { $ast = **new** ExpresionAritmetica($ex1.ast, $op, $ex2.ast);}

| ex1=expresion op=('>'|'>='|'<'|'<=') ex2=expresion { $ast = **new** ExpresionBinaria($ex1.ast, $op, $ex2.ast);}

| ex1=expresion op=('=='|'!=') ex2=expresion { $ast = **new** ExpresionBinaria($ex1.ast, $op, $ex2.ast);}

| ex1=expresion op='&&' ex2=expresion { $ast = **new** ExpresionLogica($ex1.ast, $op, $ex2.ast);}

| ex1=expresion op='||' ex2=expresion { $ast = **new** ExpresionLogica($ex1.ast, $op, $ex2.ast);}

| *IDENT* parametrosInvocacion { $ast = **new** InvocacionFuncionExpresion($IDENT, $parametrosInvocacion.list);}

;

/\* Definicion de la sentencia del bucle while \*/

bucleWhile **returns**[Sentencia ast]

: 'while' '(' expresion ')' '{' sentencias '}' { $ast = **new** BucleWhile($expresion.ast, $sentencias.list);}

;

/\* Definicion de la sentencia condicional \*/

sentenciaCondicional **returns**[Sentencia ast]

: 'if' '(' expresion ')' '{' sentencias '}' ('else' '{' sentencias '}') { $ast = **new** SentenciaCondicional($expresion.ast, $ctx.sentencias(0).list, $ctx.sentencias(1).list);}

| 'if' '(' expresion ')' '{' sentencias '}' { $ast = **new** SentenciaCondicional($expresion.ast, $ctx.sentencias(0).list, **null**);}

;