GLC – Gramática Libre de Contexto

Paula Suárez Prieto – UO269745

Diseño de Lenguajes de Programación, curso 2023-2024

Grupo PL-02

```
grammar Grammar;

import Tokenizer;

@header {
        import ast.sentence.*;
        import ast.expression.*;
        import ast.type.*;
        import ast.*;
}

// ##INICIO program: Programa principal
program returns[Program ast]
    : 'class' name=IDENT ';' ('global' ('types' dt+=defTuple*)? ('vars' vars)?)? 'create' (b+=featureBuilder ';')+
fd+=featureDef* 'end' runCall EOF
        { $ast = new Program($name.text, $ctx.dt != null ? $dt : null, $ctx.vars != null ? $vars.list :
            new ArrayList<VarDefinition>(), $b, $fd,
            ($ctx.runCall.ast != null ? $runCall.ast : new FunctionCallSent("error", new ArrayList<>()))); }
    ;
// ##FIN program

// ##INICIO defTtuple: Definición de estructuras

defTuple returns [StructDefinition ast]
    : 'deftuple' IDENT 'as' f+=field* 'end'          { $ast = new StructDefinition(new StructType($IDENT), $f); }
    ;

field returns [FieldDefinition ast]
    : IDENT ':' type ';'   { $ast = $ctx.type.ast != null ? new FieldDefinition($IDENT, $type.ast) :
                                          new FieldDefinition($IDENT, new VoidType()); }
    ;

// ##FIN defTtuple


// ##INICIO vars: Lista de declaraciones de variables
vars returns [List<VarDefinition> list = new ArrayList<VarDefinition>()]
    : (varListDefinition { $list.addAll($varListDefinition.list); })*
    ;
// ##FIN vars

// ##INICIO varListDefinition: Declaración de variables
varListDefinition returns [List<VarDefinition> list = new ArrayList<VarDefinition>()]
    : varListIdents ':' type ';'
        { for (int i = 0; i < $varListIdents.list.size(); i++)
            $list.add(
                 $ctx.type.ast != null ? new VarDefinition($varListIdents.list.get(i), $type.ast) :
                              new VarDefinition($varListIdents.list.get(i), new VoidType()));
        }
    ;

// ##FIN vars


// ##INICIO varListIdents: Lista de identificadores de variables
varListIdents returns [List<String> list = new ArrayList<String>()]
    :( IDENT { $list.add($IDENT.text); }) (',' IDENT { $list.add($IDENT.text); })*
    ;

// ##FIN varListIdents
// ##INICIO featureBuilder: Declaración de funciones (constructores)
featureBuilder returns [FunctionBuilder ast]
    : name=IDENT                          { $ast = new FunctionBuilder($name); $ast.updatePositions($name);}
    ;
// ##FIN featureBuilder

// ##INICIO featureDef: Definición de funciones
featureDef returns [FunctionDefinition ast]
    : 'feature' IDENT ('(' (p+=param (',' p+=param)*)? ')')? (':' type)? 'is' ('local' vars)? 'do' s+=sentence* 'end'
        { $ast = new FunctionDefinition($IDENT, $ctx.p != null ? $p : new ArrayList<>(), $ctx.type != null ?
        $type.ast : new VoidType(), $ctx.vars != null ? $vars.list : new ArrayList<VarDefinition>(), $s); }
    ;

// ##FIN featureDef

// ##INICIO param: Parámetros de funciones
param returns [VarDefinition ast]
    : IDENT ':' type            { $ast = $ctx.type.ast != null ? new VarDefinition($IDENT, $type.ast) :
                                    new VarDefinition($IDENT, new VoidType()) ; }
    ;
// ##FIN params


// ##INICIO runCall: Llamada a función principal
runCall returns [RunCall ast]
    : 'run' IDENT '(' (args+=expr (',' args+=expr)*)? ')' ';'    { $ast = new RunCall($IDENT, $ctx.args != null ?
                                                                          $args : new ArrayList<>()); }
    ;
// ##FIN runCall
// ##INICIO sentence: Sentencias
sentence returns [Sentence ast]
    : 'if' expr 'then' tb+=sentence* ('else' fb+=sentence*)? 'end'      { $ast = new IfElse($expr.ast, $tb,
                                                                          $ctx.fb != null ? $fb : null); }

    | ('from' initFromLoop)? 'until' expr 'loop' c+=sentence* 'end'      { $ast = new Loop($ctx.initFromLoop !=
                                                  null ? $initFromLoop.initializations : null, $expr.ast, $c); }

    | 'read' args+=expr (',' args+=expr)* ';' { $ast = new Read($ctx.args != null ? $args : new ArrayList<>()); }

    | op=('print'|'println') (args+=expr (',' args+=expr)*)? ';'     { $ast = new Print($op, $ctx.args != null ?
                                                                             $args : new ArrayList<>()); }

    | left=expr ':=' right=expr ';'                          { $ast = new Assignment($left.ast, $right.ast); }

    | token='return' expr? ';'            { $ast = new Return($ctx.expr != null ? $expr.ast : null);
                                                                            $ast.updatePositions($token);}

    | IDENT '(' (args+=expr (',' args+=expr)*)? ')' ';'    { $ast = new FunctionCallSent($IDENT,
                                                             $ctx.args != null ? $args : new ArrayList<>()); }
    ;
// ##FIN sentence


// ##INICIO initFromLoop: Inicialización de variables del bucle
initFromLoop returns [List<Assignment> initializations = new ArrayList<Assignment>()]
    : (left=expr ':=' right=expr ';' { $initializations.add(new Assignment($left.ast, $right.ast)); })+
    ;
// ##FIN initFromLoop
```

Paula Suárez Prieto, UO269745
Grupo de prácticas PL-02
Curso 2023-2024

```
// ##INICIO expr: Expresiones
expr returns [Expression ast]
: INT_CONSTANT                                               { $ast = new IntConstant($INT_CONSTANT); }

| REAL_CONSTANT                                              { $ast = new RealConstant($REAL_CONSTANT); }

| CHAR_CONSTANT                                              { $ast = new CharConstant($CHAR_CONSTANT); }

| IDENT                                                                 { $ast = new Variable($IDENT); }

| '(' expr ')'                                                          { $ast = $expr.ast; }

| IDENT '(' (args+=expr (',' args+=expr)*)? ')'          { $ast = new FunctionCallExpr($IDENT, $args); }

| root=expr '.' IDENT                                    { $ast = new FieldAccess($root.ast, $IDENT); }

| array=expr'[' index=expr ']'                           { $ast = new ArrayAccess($array.ast, $index.ast); }

| 'to<' castType=type '>(' value=expr ')'      { $ast = $ctx.castType.ast != null ? new CastExpr($castType.ast,
                                                    $value.ast) : new CastExpr(new VoidType(), $value.ast);}

| '-' expr                                                              { $ast = new MinusExpr($expr.ast); }

| 'not' expr                                                            { $ast = new NotExpr($expr.ast); }

| op1=expr operator=('*' | '/' | 'mod') op2=expr  { $ast = new ArithmeticExpr($op1.ast, $operator, $op2.ast); }

| op1=expr operator=('+' | '-') op2=expr      { $ast = new ArithmeticExpr($op1.ast, $operator, $op2.ast); }
| op1=expr operator=( '>' | '<' | '>=' | '<=') op2=expr  { $ast = new ComparationExpr($op1.ast, $operator, $op2.ast); }
| op1=expr operator=('=' | '<>' ) op2=expr      { $ast = new ComparationExpr($op1.ast, $operator, $op2.ast); }

| op1=expr operator='and' op2=expr              { $ast = new LogicalExpr($op1.ast, $operator, $op2.ast); }

| op1=expr operator='or' op2=expr               { $ast = new LogicalExpr($op1.ast, $operator, $op2.ast); }
;
// ##FIN expr


// ##INICIO type: Tipos de datos
type returns [Type ast]
    : token='INTEGER'                         { $ast = new IntType(); $ast.updatePositions($token);}
    | token='DOUBLE'                          { $ast = new DoubleType(); $ast.updatePositions($token);}
    | token='CHARACTER'                       { $ast = new CharType(); $ast.updatePositions($token);}
    | '[' INT_CONSTANT ']' type               {$ast = new ArrayType(new IntConstant($INT_CONSTANT),$type.ast);
                                                   $ast.updatePositions($ctx.start);}
    | IDENT                                   { $ast = new StructType($IDENT); $ast.updatePositions($ctx.start); }
    ;
// ##FIN type
```