```
grammar Control;

// tokens
NIL : 'nil';
INT     : [0-9]+ ;
ID      : [a-zA-Z_][a-zA-Z0-9_]* ;
FLOAT     : [0-9]+'.'[0-9]+ ;
STRING  : "" (~["\r\n] | """")* "" ;
CHAR       : '' ~['\r\n] '' ;
//skip

WHITESPACE: [ \\\r\n\t]+ -> skip;
COMMENT : '/*' .*? '*/' -> skip;
LINE_COMMENT : '//' ~[\r\n]* -> skip;
// rules

prog: block EOF;

block: (stmt)*
     ;

stmt: assignstmt
    | printlnstmt
    | printstmt
    | ifstmt
    | whilestmt
    | switchstmt
    | forstmt
    | guardstmt
    | vectorPpts
    | matrixstmt
```

```
    | funcstmt
    | callFunction
    | returnstmt
    ;

assignstmt
    : ID  '=' expr                      # reasignacion
    | var_type ID ':' primitivo '=' expr # asignacion
    | var_type ID ':' primitivo '?'      # asignacionNoExpr
    | var_type ID '=' expr               # asignacionNoPrimitive
    | ID '+=' expr                       # incremento
    | ID '-=' expr                      # decremento
    | var_type ID ':' '[' primitivo ']' '=' '[]'         # asignacionVectorVacio
    | var_type ID ':' '[' primitivo ']' '=' '['listaExp']'          # asignacionVector
    | ID '['expr']' '=' expr # reasignacionVector
    | ID '['expr']''['expr']' '=' expr # reasignacionMatrixTwoD
    ;

vectorPpts
    : ID '.' 'append' '('expr')' #vectorAppend
    | ID '.' 'removeLast' '(')' #vectorRemoveLast
    | ID '.' 'remove' '(' 'at' ':' expr ')' #vectorRemoveAt
    ;
matrixstmt
    : var_type ID ':' '''['primitivo']''' '=' defMatrix #matrixTwoD
    | var_type ID ':' '''''['primitivo']'''' '=' defMatrix #matrixThreeD
    ;
defMatrix
    : listaValores_Mat
    ;
listaValores_Mat
```

```
    : '[' listaValores_Mat2 ']'
    ;
lista_Expresiones
    : expr (','expr)*
    ;
listaValores_Mat2
    : listaValores_Mat2 ',' listaValores_Mat
    | listaValores_Mat
    | lista_Expresiones
    ;



funcstmt

    : 'func' ID '('listaParams?')' '{' block '}' #func_sinRetorno
    | 'func' ID '('listaParams?')' '->' primitivo '{'block ret='return'? exp=expr?
'}' #func_conRetorno_conTipo
    ;
 callFunction
    :  ID '('listaParamsCall?')'
    ;


listaParamsCall
    : ( ID ':')? ref='&'? expr (',' ( ID ':')? ref='&'? expr)*
    ;


listaParams
    : ext=(ID | '_' )? ID ':' ino='inout'? '['? primitivo isVector=']'? # oneParam
    |  ext=(ID | '_' )? ID ':' ino='inout'? '['? primitivo isVector=']'? ','
listaParams #numParams
    ;
```

```
listaExp
    : expr #oneExpr
    | expr ',' listaExp # numExpr
    ;
returnstmt
    : 'return' expr?
    ;
printlnstmt
    : 'println' '(' expr ')'
    ;

printstmt
    : 'print' '(' expr (',' expr)*')'
    ;

ifstmt
    :'if' expr '{' block (transfer_sentence)? '}' 'else' ifstmt       #else_if
    | 'if' expr '{' block (transfer_sentence)? '}' 'else' '{' block
(transfer_sentence)? '}'  #else
    | 'if'  expr  '{' block (transfer_sentence)? '}'                  #ifNormal



    ;

switchstmt
    : 'switch'  expr  '{' cases '}' ;

cases
    : (caseblock)* ;

caseblock
    : 'case' expr ':' block ('break')? # unCase
```

```
    | 'default' ':' block  # unDefault
    ;


whilestmt
    : 'while'  expr  '{' block '}'
    ;
forstmt
    : 'for' ID 'in' expr '...' expr '{' block '}' #forNormal
    | 'for' ID 'in' expr '{'block '}'             #forEach
    ;
guardstmt
    : 'guard' expr 'else"{' block (transfer_sentence)? '}'
    ;

expr
    : '!' right=expr                    # NotExpr
    | '-'   right=expr                  #negExpr
    | left=expr op='%' right=expr         # OpExpr
    | left=expr op=('*'|'/') right=expr    # OpExpr
    | left=expr op=('+'|'-') right=expr    # OpExpr
    | left=expr op=('>='|'>') right=expr   # OpExpr
    | left=expr op=('<='|'<') right=expr   # OpExpr
    | left=expr op=('=='|'!=') right=expr  # OpExpr
    | left=expr op=('&&'|'||') right=expr  # OpExpr
    | '(' expr ')'                      # ParExpr
    | NIL                               # nilExpr
    | INT                               # IntExpr
    | STRING                              # StrExpr
    | ('true' | 'false')                 # BoolExpr
    | FLOAT                               # FloatExpr
```

```
    | CHAR                                # CharExpr
    | ID '('listaParamsCall?')' #callFuncAsExpr
    | ID                                  # IdExpr
    | ID '.' 'isEmpty' #vectorIsEmpty
    | ID '.' 'count' #vectorCount
    | ID '['expr']'  #vectorGetElement
    | ID '['expr']''['expr']' #accesoMatrixTwoD
    | 'String' '('expr')' #toString
    | 'Int' '('expr')'  #toInt
    | 'Float' '('expr')' #toFloat


    ;

primitivo
    : 'Int'
    | 'String'
    | 'Float'
    | 'Bool'
    | 'Character'
    ;
transfer_sentence
    : 'continue'
    | 'break'
    ;

var_type
    : 'let'
    | 'var'
    ;
```