<programa>::= <declaraciones> BEGIN.PROGRAM <lista\_sentencias> END.PROGRAM | BEGIN.PROGRAM <print> END.PROGRAM //print es la unica sentencia que puede ejecutarse sin uso del bloque de declaración de variables

<declaraciones>::= DECLARE <lista\_linea\_declaraciones> ENDDECLARE

<lista\_linea\_declaraciones>::= <lista\_linea\_declaraciones><linea\_declaracion> | <linea\_declaracion>

<linea\_declaracion>::= [ <id><linea\_declaracion><tipo> ] | , <id><linea\_declaracion><tipo> , | ] = [ // esta union de reglas me permite escribir cosas como:

DECLARE

[var1 [ var2 [ var3 ]=[ float ] integer ] string]

ENDDECLARE

**Modificar para que reconozca correctamente.**

<lista\_sentencias>::= <sentencia><lista\_sentencias> | <sentencia>

<sentencia>::= <asignacion> | <print> | <if> | <while>

<asignacion>::= **ID ASIGN** <expresion>

<expresion>::= <expresion> + <termino> | <expresion> - <termino> | <termino>

<termino> ::= <termino> \* <factor> | <termino> / <factor> | <factor>

<factor> ::= **ID** | <cte\_numerica> | ( <expresion> ) | <take>

<cte\_numerica>::= **CTE\_INT| CTE\_FLOAT** //tokens en lexico

<take>::= **TAKE** **PARENTESIS\_A** <op\_num> **PTO\_COMA** **CTE\_INT PTO\_COMA** **CORCHETE\_A CORCHETE\_C** **PARENTESIS\_C** | **TAKE** **PARENTESIS\_A** <op\_num> **PTO\_COMA** **CTE\_INT PTO\_COMA** **CORCHETE\_A** <lista\_take> **CORCHETE\_C** **PARENTESIS\_C**

<op\_num>::= **MAS | MENOS | DIV | POR** //tokens en lexico recordarlo para toda la gramática

<lista\_take>::= <lista\_take> ; **CTE\_INT** | **CTE\_INT** //utilizar siempre recursividad a izquierda.

<print>::= **PRINT CTE\_STRING** //solo puedo hacer print de constante string

<if>::= if <lista\_condiciones> { <bloque> } | if <lista\_condiciones> { <bloque> } else { <bloque> }

<while>::= while <lista\_condiciones> { <bloque> }

<bloque>::= <lista\_sentencias>

<lista\_condiciones>::= ( <condicion> && **<condicion>** ) | ( <condicion> || **<condicion>**) | ( <condicion> )

<condicion>::= **<expresion><op\_logico><expresion>**

<op\_logico>::= **IQUAL | DISTINTO | MENOR | MENOR\_IQUAL | MAYOR | MAYOR\_EQUAL** //tokens en lexico

~~<id>::= <letra> | <letra><lista\_chars>~~

<tipo>::= **INT | FLOAT | STRING** //terminales se expresan en mayúsculas

~~<cte\_int>::= <numero>~~

~~<cte\_float>::= <numero> . | . <numero> | <numero>.<numero>~~

~~<cte\_string>::= " <lista\_chars> " | ""~~

~~<lista\_chars>::= <char><cte\_string> | <char>~~

~~<char>::= <letra> | <digito> | \_~~

~~<numero>::= <digito><numero> | <digito>~~

~~<letra>::= a | b | c | ... | z | A | B | C | ... | Z~~

~~<digito>::= 0 | 1 | 2 | ... | 9~~

// no se hacen gramáticas para tokens que ya se encuentras definidos por expresiones regulares en el lexico, se usan los nombres de los tokens como terminales en la gramática.