**REGLAS ORIGINALES:**

1. **<Programa>** → PROGRAMA id ; <decl\_var> <instrucciones> .
2. **<decl\_var>** → VAR <lista\_id> : <tipo> ; <decl\_v>
3. **<decl\_var>** → λ
4. **<decl\_v>** → <lista\_id> : <tipo> ; <decl\_v>
5. **<decl\_v>** → λ
6. **<lista\_id>** → id <resto\_listaid>
7. **<resto\_listaid>** → , <lista\_id>
8. **<resto\_listaid>** → λ
9. **<Tipo>** → <tipo\_std>
10. **<Tipo>** → VECTOR [num] DE <Tipo\_std>
11. **<Tipo\_std>** → ENTERO
12. **<Tipo\_std>** → REAL
13. **<Tipo\_std>** → BOOLEANO
14. **<instrucciones>** → INICIO <lista\_inst> FIN
15. **<lista\_inst>** → <instrucción> ; <lista\_inst>
16. **<lista\_inst>** → λ
17. **<instrucción>** → INICIO <lista\_inst> FIN
18. **<instrucción>** → <inst\_simple>
19. **<instrucción>** → <inst\_es>
20. **<instrucción>** →  SI <expresion> ENTONCES <instrucción> SINO <instrucción>
21. **<instrucción>** →  MIENTRAS <expresión> HACER <instrucción>
22. **<inst\_simple>** → id <resto\_instsimple>
23. **<resto\_instsimple>** → opasigna <expresión>
24. **<resto\_instsimple>** → [ <expr\_simple> ] opasigna <expresión>
25. **<resto\_instsimple>** → λ
26. **<variable>** → id <resto\_var>
27. **<resto\_var>** → [ <expr\_simple> ]
28. **<resto\_var>** → λ
29. **<inst\_es>** → LEE ( id )
30. **<inst\_es>** → ESCRIBE ( <expr\_simple>)
31. **<expresión>** → <expr\_simple> <expresiónPrime>
32. **<expresiónPrime>** → oprel <expr\_simple>
33. **<expresiónPrime>** → λ
34. **<expr\_simple>** → <término> <resto\_exsimple>
35. **<expr\_simple>** → <signo> <término> <resto\_exsimple>
36. **<resto\_exsimple>** → opsuma <término> <resto\_exsimple>
37. **<resto\_exsimple>** → O <término> <resto\_exsimple>
38. **<resto\_exsimple>** → λ
39. **<término>** → <factor> <resto\_term>
40. **<resto\_term>** → opmult <factor> <resto\_term>
41. **<resto\_term>** → Y <factor> <resto\_term>
42. **<resto\_term>** → λ
43. **<factor>** → <variable>
44. **<factor>** → num
45. **<factor>** → ( <expresión> )
46. **<factor>** → NO <factor>
47. **<factor>** → CIERTO
48. **<factor>** → FALSO
49. **<signo>** → +
50. **<signo>** → -

**ATRIBUTOS PARA TIPO DE VARIABLE**

1. **<Programa>** → PROGRAMA id *{si equivaleAReservada(id.lexema) o tablasimbolos.contiene(id.lexema) entonces error  
   sino   
    tablaSimbolos.inserta(id.lexema, “programa”)  
    tablaSimbolos.insertainfo(id.lexema, “clase”, “programa”)  
    fin si}*; <decl\_var> <instrucciones> .

*{si ERROR entonces <Programa>.arb = NodoVacio(token.linea)*

*Sino*

*<Programa>.arb = <instrucciones>.arb }*

1. **<decl\_var>** → VAR <lista\_id> : <tipo> ; {  
   *para v en <lista\_id>.lista hacer  
    si equivaleAReservada(v) o tablaSimbolos.contiene(v) entonces error  
    sino   
    tablaSimbolos.inserta(v, <tipo>.t)  
    tablaSimbolos.insertainfo(v, “clase”, <tipo>.clase)  
    si <tipo>.clase == “vector” entonces tablaSimbolos.insertainfo(v, “longitud”, <tipo>.longitud)  
    fin si   
   fin para*} <decl\_v>
2. **<decl\_var>** → λ
3. **<decl\_v>** → <lista\_id> : <tipo> ; {  
   *para v en <lista\_id>.lista hacer  
    si equivaleAReservada(v) o tablaSimbolos.contiene(v) entonces error  
    sino   
    tablaSimbolos.inserta(v, <tipo>.t)  
    tablaSimbolos.insertainfo(v, “clase”, <tipo>.clase)  
    si <tipo>.clase == “vector” entonces tablaSimbolos.insertainfo(v, “longitud”, <tipo>.longitud)  
    fin si   
   fin para*} <decl\_v>
4. **<decl\_v>** → λ
5. **<lista\_id>** → id *{<resto\_listaid>.lh = [id.lexema]}* <resto\_listaid> *{<lista\_id>.lista = <resto\_listaid>.lista}*
6. **<resto\_listaid>** → , <lista\_id> *{<resto\_listaid>.lista = <resto\_listaid>.lh ++ <lista\_id>.lista}*
7. **<resto\_listaid>** → λ *{<resto\_listaid>.lista = <resto\_listaid>.lh}*
8. **<Tipo>** → <tipo\_std> *{<Tipo>.t = <tipo\_std>.t; <Tipo>.clase = “variable”, <Tipo>.longitud = 0}*
9. **<Tipo>** → VECTOR [num] DE <Tipo\_std> *{<Tipo>.t = <tipo\_std>.t; <Tipo>.clase = “vector”; <Tipo>.longitud = num.valor}*
10. **<Tipo\_std>** → ENTERO *{<tipo\_std>.t = “entero”}*
11. **<Tipo\_std>** → REAL *{<tipo\_std>.t = “real”}*
12. **<Tipo\_std>** → BOOLEANO *{<tipo\_std>.t = “booleano”}*

//AST (+ Programa ^)

1. **<instrucciones>** → INICIO <lista\_inst> FIN

*{si ERROR entonces <instrucciones>.arb = NodoVacio(token.linea)*

*Sino*

*<instrucciones>.arb = NodoCompuesta(<lista\_inst>.lista, token.linea) }*

1. **<lista\_inst>** → <instrucción> ; <lista\_inst>1

*{si ERROR entonces <lista\_inst>.lista = [NodoVacio(token.linea)] + <lista\_inst>1.lista*

*Sino*

*<lista\_inst>.lista = [<instrucción>.arb] ++ <lista\_insr>1.lista }*

1. **<lista\_inst>** → λ

*{si ERROR entonces <lista\_inst>.lista = [NodoVacio(token.linea)]*

*Sino*

*<lista\_inst>.lista = []}*

1. **<instrucción>** → INICIO <lista\_inst> FIN

*{si ERROR entonces <instrucción>.arb = NodoVacio(token.linea)*

*Sino*

*<instrucción>.arb = NodoCompuesta(<lista\_inst>.lista, token.linea)}*

1. **<instrucción>** → <inst\_simple>

*{si ERROR entonces <instrucción>.arb = NodoVacio(token.linea)*

*Sino*

*<instrucción>.arb = <inst\_simple>.arb}*

1. **<instrucción>** → <inst\_es>

*{si ERROR entonces <instrucción>.arb = NodoVacio(token.linea)*

*Sino*

*<instrucción>.arb = <inst\_es>.arb}*

1. **<instrucción>** →  SI <expresion> ENTONCES <instrucción>1 SINO <instrucción>2

*{si ERROR entonces <instrucción>.arb = NodoVacio(token.linea)*

*Sino*

*<instrucción>.arb = NodoSi(<expresion>.arb, <instrucción>1.arb, <instrucción>2.arb, token.linea) }*

1. **<instrucción>** →  MIENTRAS <expresión> HACER <instrucción>1

*{si ERROR entonces <instrucción>.arb = NodoVacio(token.linea)*

*Sino*

*<instrucción>.arb = NodoMientras(<expresion>.arb, <instrucción>1.arb, token.linea)}*

1. **<inst\_simple>** → id *{<resto\_instsimple>.h = id}* <resto\_instsimple>

*{si ERROR entonces <instr\_simple>.arb = NodoVacio(token.linea)*

*Sino*

*<instr\_simple>.arb = <resto\_instsimple>.arb }*

1. **<resto\_instsimple>** → opasigna <expresión>

*{si ERROR entonces <resto\_instsimple>.arb = NodoVacio(token.linea)*

*Sino*

*variable = NodoAccesoVariable(<resto\_instsimple>.h.lexema, token.linea)*

*<resto\_instsimple>.arb = NodoAsignación(variable, <expresión>.arb, token.linea) }*

1. **<resto\_instsimple>** → [ <expr\_simple> ] opasigna <expresión>

*{si ERROR entonces <resto\_instsimple>.arb = NodoVacio(token.linea)*

*Sino*

*vector = NodoAccesoVector(<resto\_instsimple>.h.lexema, <expresión>.arb, token.linea, <resto\_instsimple>.h.tipo)*

*<resto\_instsimple>.arb = NodoAsignación(vector, token.linea) }*

1. **<resto\_instsimple>** → λ

*{si ERROR entonces <resto\_instsimple>.arb = NodoVacio(token.linea)*

*Sino*

*<resto\_instsimple>.arb = NodoAccesoVariable(<resto\_instsimple>.h.lexema, token.linea, <resto\_instsimple>.h.tipo) }*

1. **<variable>** → id *{<resto\_var>.h = id}* <resto\_var>

*{si ERROR entonces <variable>.arb = NodoVacio(token.linea)*

*Sino*

*<variable>.arb = <resto\_var>.arb }*

1. **<resto\_var>** → [ <expr\_simple> ]

*{si ERROR entonces <resto\_var>.arb = NodoVacio(token.linea)*

*Sino*

*<resto\_var>.arb = NodoAccesoVector(<resto\_var>.h.lexema, <expresión>.arb, token.linea, <resto\_var>.h.tipo)}*

1. **<resto\_var>** → λ

*{si ERROR entonces <resto\_var>.arb = NodoVacio(token.linea)*

*Sino*

*<resto\_var>.arb = NodoAccesoVariable(<resto\_var>.h.lexema, token.linea, <resto\_var>.h.tipo)}*

1. **<inst\_es>** → LEE ( id )

*{si ERROR entonces <inst\_es>.arb = NodoVacio(token.linea)*

*Sino*

*var = NodoAccesoVariable(id.lexema, token.linea, id.tipo)*

*<inst\_es>.arb = NodoLee(var, token.linea)}*

1. **<inst\_es>** → ESCRIBE ( <expr\_simple>)

*{si ERROR entonces <inst\_es>.arb = NodoVacio(token.linea)*

*Sino*

*var = NodoAccesoVariable(id.lexema, token.linea, id.tipo)*

*<inst\_es>.arb = NodoEscribe(var, token.linea)}*

1. **<expresión>** → <expr\_simple> {<expresiónPrime>.h = <expr\_simple>.arb} <expresiónPrime>

*{si ERROR entonces <expresion>.arb = NodoVacio(token.linea)*

*Sino*

*<expresion>.arb = <expresionPrime>.arb}*

1. **<expresiónPrime>** → oprel <expr\_simple>

*{si ERROR entonces <expresionPrime>.arb = NodoVacio(token.linea)*

*Sino*

*<expresionPrime>.arb = NodoComparacion(<expresionPrime>.h, <expr\_simple>.arb, token.linea, oprel.operacion) }*

1. **<expresiónPrime>** → λ

*{si ERROR entonces <expresionPrime>.arb = NodoVacio(token.linea)*

*Sino*

*<expresionPrime>.arb = <expresionPrime>.h}*

1. **<expr\_simple>** → <término> *{<resto\_exsimple>.h = <término>.arb}* <resto\_exsimple>

*{si ERROR entonces <expr\_simple>.arb = NodoVacio(token.linea)*

*Sino*

*<expr\_simple>.arb = <resto\_exsimple>.arb}*

1. **<expr\_simple>** → <signo> <término> *{<resto\_exsimple>.h = <término>.arb}* <resto\_exsimple>

*{si ERROR entonces <expr\_simple>.arb = NodoVacio(token.linea)*

*Sino*

*<expr\_simple>.arb = NodoSigno(<signo>.signo, <resto\_exsimple>.arb) }*

1. **<resto\_exsimple>** → opsuma <término> *{ <resto\_exprsimple>1.h = NodoAritmetico(<resto\_exsimple>.h, <término>.arb, token.linea, opsuma.operacion) }*  <resto\_exsimple>1

*{si ERROR entonces <resto\_exsimple>.arb = NodoVacio(token.linea)*

*Sino*

*<resto\_exprsimple>.arb = <resto\_exprsimple>1.arb}*

1. **<resto\_exsimple>** → O <término> *{ <resto\_exprsimple>1.h = NodoLogico(<resto\_exsimple>.h, <término>.arb, token.linea, “O”) }*  <resto\_exsimple>1

*{si ERROR entonces <resto\_exsimple>.arb = NodoVacio(token.linea)*

*Sino*

*<resto\_exprsimple>.arb = <resto\_exprsimple>1.arb}*

1. **<resto\_exsimple>** → λ

*{si ERROR entonces <resto\_exsimple>.arb = NodoVacio(token.linea)*

*Sino*

*<resto\_exprsimple>.arb = <resto\_exprsimple>.h}*

1. **<término>** → <factor> *{<resto\_term>.h = <factor>.arb}* <resto\_term>

*{si ERROR entonces <término>.arb = NodoVacio(token.linea)*

*Sino*

*<término>.arb = <resto\_term>.arb}*

1. **<resto\_term>** → opmult <factor> *{<resto\_term>1.h = NodoAritmetico(<resto\_term>.h, <factor>.arb, token.linea, opmult.operacion) }* <resto\_term>1

*{si ERROR entonces <resto\_exsimple>.arb = NodoVacio(token.linea)*

*Sino*

*<resto\_term>.arb = <resto\_term>1.arb}*

1. **<resto\_term>** → Y <factor> *{<resto\_term>1.h = NodoLogico(<resto\_term>.h, <factor>.arb, token.linea, “Y”) }* <resto\_term>1

*{si ERROR entonces <resto\_term>.arb = NodoVacio(token.linea)*

*Sino*

*<resto\_term>.arb = <resto\_term>1.arb}*

1. **<resto\_term>** → λ

*{si ERROR entonces <resto\_term>.arb = NodoVacio(token.linea)*

*Sino*

*<resto\_term>.arb = <resto\_term>.h}*

1. **<factor>** → <variable>

*{si ERROR entonces <factor>.arb = NodoVacio(token.linea)*

*Sino*

*<factor>.arb = <variable>.arb }*

1. **<factor>** → num

*{si ERROR entonces <factor>.arb = NodoVacio(token.linea)*

*Sino*

*si num.tipo = “entero” entonces*

*<factor>.arb = NodoEntero(num.valor, token.linea)*

*sino*

*<factor>.arb = NodoReal(num.valor, token.linea)}*

1. **<factor>** → ( <expresión> )

*{si ERROR entonces <factor>.arb = NodoVacio(token.linea)*

*Sino*

*<factor>.arb = <expresión>.arb }*

1. **<factor>** → NO <factor>1

*{si ERROR entonces <factor>.arb = NodoVacio(token.linea)*

*Sino*

*<factor>.arb = NodoNo(<factor>1.arb, token.linea) }*

1. **<factor>** → CIERTO

*{si ERROR entonces <factor>.arb = NodoVacio(token.linea)*

*Sino*

*<factor>.arb = NodoBooleano(1, token.linea) }*

1. **<factor>** → FALSO

*{si ERROR entonces <factor>.arb = NodoVacio(token.linea)*

*Sino*

*<factor>.arb = NodoBooleano(0, token.linea) }*

1. **<signo>** → +

*{si ERROR entonces <signo>.arb = NodoVacio(token.linea)*

*Sino*

*<signo>.signo = “+”}*

1. **<signo>** → -

*{si ERROR entonces <signo>.arb = NodoVacio(token.linea)*

*Sino*

*<signo>.signo = “-”}*