Analizador Semántico

Aumentada

Z → P

```
1. Z.tipo = P.tipo
```

Inicio

P → BP

```
1. P.tipo = (B.tipo == tipoOk && P.tipo==tipoOk)? tipoOk :
   tipoError
```

P→ FP

```
1. P.tipo = (F.tipo == tipoOk && P.tipo==tipoOk)? tipoOk :
   tipoError
```

P→ λ

```
1. P.tipo = tipoOk
```

Funciones

F → I J G

```
1. insertarFuncion(I.id,J.numeros,J.tipos,G.returnType) &&
   F.tipo=G.tipo
```

I → function H id

```
1. if(!comprobarTS(id.valor)){
2.   int codigoTS = insertarFuncion(id,h.tipo)
3.   I.tipo = H.tipo
4.   I.id = codigoTS
5.   tsEnFuncion
6. }else{
7.   Error("Funcion o nombre de funcion ya declarado")
8. }
```

J → (A)

```
1. J.numeros=A.numeros && J.tipos=A.tipos
```

G → { C }

```
1. G.atributos = C.atributos
```

H → T

```
1. H.tipo = T.tipo
```

H → λ

```
1. H.tipo= EMPTY
```

A → T id K

```
1. A.numeros=1+k.numeros
2. A.tipos=[T] U K.tipos
3. insertarTSActual(id,T.tipo)
```

A → λ

```
1. A.numeros = 0
2. A.tipos=[]
```

K → , T id K

```
1. K.numeros=1+K1.numeros
2. K.tipos = [T.tipo] U K.tipos
3. insertarTSActual(id,T.tipo)
```

K → λ

```
1. K.numeros = 0
2. K.tipo = []
```

C → BC

```
1. C.tipo = (B.tipo = C.tipo = tipoOk)? tipoOk : tipoError
```

C → λ

```
1. C.tipo = tipoOk
```

Sentencias:

Simples

S → id = E ;

```
1. If(comprobarTS(id))
2.   S.tipo = (id.tipo== E.tipo)? tipoOk : tipoError
3. Else{
4.   insertarTSGlobal(id,Entero)
5.   S.tipo = (E.tipo== E.tipo)? tipoOk : tipoError
6. }
```

S → id (L) ;

```
1. If(comprobarTS(id))
```

```
2.   S.tipo = comprobarArgumentosYNum(L.tipos, L.numeros)? tipoOk :
   tipoError
3. Else
4.   Error("funcion no declarada")
```

S → alert (E) ;

```
1. comprobacionesAlert
```

S → input (id) ;

```
1. If(comprobarTS(id))
2.          Comprobaciones input
3. Else
4.   insertarTSGlobar(id,ENTERO)
5.   comprobacionesInput
```

S → return X ;

```
1. S.returnType = X.tipo
2. S.hasReturn=true
```

S → id -= E ;

```
1. if(comprobarTS)
2.   S.tipo = (id.tipo = E.tipo = INTEGER)? tipoOk: tipoError
3. Else{
4.   insertarTSGlobal(id,ENTERO)
5.   S.tipo = (E.tipo = INTEGER)? tipoOk: tipoError
6. }
```

Compuestas

B → if (E) S

```
1. B.atrib = (E.tipo == BOOLEAN && S.tipo=tipoOk)? S.atrib :
   error("if(<logico>) <sentencia Simple>")
```

B → let T id ;

```
1. insertarTSActual(id.tipo,T.tipo)
2. B.tipo = tipoOk
```

B → S

```
1. B.atributos = S.atributos
```

B → do { C } while (E);

```
1. B.atrib = (E.tipo == BOOLEAN && C.tipo=tipoOk)? C.atrib :
   error("do{<sentencias>}while(<logico>)")
```

T → number

```
1. T.tipo = NUMBER
```

T → boolean

```
1. T.tipo = BOOLEAN
```

T → String

```
1. T.tipo = STRING
```

Expresiones

E → E || R

```
1. E.tipo = (E1.tipo = R.tipo = BOOLEAN)? BOOLEAN: error("Se
   compara entre dos logicos")
```

E → R

```
1. E.tipo = R.tipo
```

R → R && U

```
1. R.tipo = (R1.tipo = U.tipo = BOOLEAN)? BOOLEAN: error("Se
   compara entre dos logicos")
```

R → U

```
1. R .tipo = U.tipo
```

U → U == V

```
1. U.tipo = (U1.tipo == V.tipo)? BOOLEAN: error("No juntes churras
   con merinas")
```

U → U != V

```
1. U.tipo = (U1.tipo == V.tipo)? BOOLEAN: error("No juntes churras
   con merinas")
```

U → V

```
1. U.tipo = V.tipo
```

V → V + W

```
1. V.tipo = (V1.tipo == NUMBER && W.tipo == NUMBER)? NUMBER :
   error("La suma es con numeros")
```

V → V - W

```
1. V.tipo = (V1.tipo == NUMBER && W.tipo == NUMBER)? NUMBER :
   error("La resta es con números")
```

V → W

```
1. V.tipo = W.tipo
```

W → id

```
1. if(!estaEnTS(id.valor)) {
2.          añadir a TSGlobal como entero
3. }
4. W.tipo = E.tipo
```

W → (E)

```
1. W.tipo = E.tipo
```

W → id (L)

```
1. if(tipoReturn(id) != EMPTY){
2.   if(comprobarArg(id,L.tipos,L.numeros)){
3.          W.tipo = tipoReturn(id)
4.   }else{
5.          Error("El numero o tipo de parámetros son incorrectos")
6.   }
7. }else{
8. Error("La function tiene que devolver algo")
9. }
```

W → entero

```
1. W.tipo = NUMBER
```

W → cadena

```
1. W.tipo = STRING
```

Argumentos de la función

L → E Q

```
1. L.numeros = Q.numeros+1
2. L.tipos= [E.tipo] U Q.tipos
```

L → λ

```
1. L.numeros = 0
2. L.tipos=[]
```

Q → , E Q

```
1. Q.numeros = Q1.numeros+1
2. Q.tipos = [E.tipo] U Q.tipos
```

Q → λ

```
1. Q.numeros = 0
2. Q.tipos=[]
```

Return

X → E

```
1. X.tipo = E.tipo
```

X → λ

```
1. X.tipo = EMPTY
```

0. Z → P
1. P → BP
2. P→ FP
3. P→ λ
4. F → I J G
5. I → function H id
6. J → (A)
7. G → { C }
8. H → T
9. H → λ
10. A → T id K
11. A → λ
12. K → , T id K
13. K → λ
14. C → BC
15. C → λ
16. S → id = E ;
17. S → id (L) ;
18. S → alert (E) ;
19. S → input (id) ;
20. S → return X ;
21. S → id -= E ;
22. B → if (E) S
23. B → let T id ;
24. B → S
25. B → do { C } while (E);
26. T → number
27. T → boolean
28. T → String
29. E → E || R
30. E → R
31. R → R && U
32. R → U
33. U → U == V
34. U → U != V
35. U → V
36. V → V + W
37. V → V – W
38. V → W
39. W → id
40. W → (E)
41. W → id (L)
42. W → entero
43. W → cadena
44. L → E Q
45. L → λ
46. Q → , E Q
47. Q → λ
48. X → E
49. X → λ