Analizador Semántico

Aumentada

Z → P

```
CrearTS() = TSGl
Despl = 0
```

Inicio

P → BP

```
If(B.hasReturn){
        Error(Return Fuera de funcion)
}
```

P→ FP

```
P.atributos = P.atributos
```

P→ λ

Funciones

F → I J G

```
If(G.hasReturn && I.tipo==G.returnType || !G.hasReturn && I.tipo =
Empty){
        insertarTipoTS(I.idFuncion, J.tipo( I.tipo)
}else{
        Error(Tipo de retorno y return no equiparan)
}
DestruirTS()
TSActual = TSG
```

I → function H id

```
TSActual = CrearTs
Despl = 0
InsertarEti(id.pos,genEti())
I.tipo = H.tipo
I.idFuncion = id.pos
```

J → (A)

```
J.tipo = A.tipo
```

G → { C }

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

H → T

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

H → λ

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

A → T id K

```
insertarTipoTSActual(id.pos,T.tipo)
```

```
A.tipo = T.tipo x K.tipo
```

A → λ

```
A.tipo = Empty
```

K → , T id K

```
insertarTipoTSActual(id.pos,T.tipo)
A.tipo = T.tipo x K.tipo
```

K → λ

```
K.tipo = Empty
```

C → BC

```
C.atributos = merge(B.atributos,C.atributos)
// la funcion merge, combina atributos dando prioridad a quien tenga
el return y el tipoError
```

C → λ

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

Sentencias:

Simples

S → id = E ;

```
tipoId = BuscaTipoTs(id.pos)
        if(tipoId == Empty){
                insertarTipoDesplTSGlobal (id.pos, Entero ,1)
                tipoId = Entero
}
If(E.tipo == tipoId){
        S.tipo = tipoOk
}else{
        S.tipo = tipoError
        Error(Asignacion incorrecta)
}
```

S → id (L) ;

```
If(isFuncion(id.pos)){
If(comprobarArg(id.pos, L.tipos)){
        S.tipo = tipoOK
}else{
        S.tipo = tipoError
        Error(Argumentos no correctos)
}
}else{
        Error(Función No declarada)
}
```

S → alert (E) ;

```
If(E.tipo ∈{String,Entero}){
        S.tipo = tipoOk
}else{
        S.tipo = tipoError
        Error(Alert mal construido)
}
```

S → input (id) ;

```
tipoId = BuscaTipoTs(id.pos)
        if(tipoId == Empty){
                insertarTipoDesplTSGlobal (id.pos, Entero ,1)
                tipoId = Entero
}
        If(tipoId ∈{String,Entero}){
                S.tipo = tipoOk
}else{
        S.tipo = tipoError
        Error(input mal construido)
}
```

S → return X ;

```
S.returnType = X.tipo
S.hasReturn = True
```

S → id -= E ;

```
tipoId = BuscaTipoTs(id.pos)
        if(tipoId == Empty){
                insertarTipoDesplTSGlobal (id.pos, Entero ,1)
                tipoId = Entero
}
If(E.tipo == Entero && tipoId==Entero){
        S.tipo = tipoOk
}else{
        S.tipo = tipoError
        Error(Asignacion Con resta incorrecta)
}
```

Compuestas

B → if (E) S

```
If(E.tipo == Logico && S.tipo == tipoOk)
        B.tipo = tipoOk
Else{
        B.tipo = tipoError
        Error(Construccion del If incorrecta)
}
```

B → let T id ;

```
tipoID = BuscaTipoTS(id.pos)
If(tipoID == Empty){
        insertarTipoTS(id.pos,T.tipo)
        B.tipo = tipoOk
}else{
        B.tipo = tipoError
        Error("Tipo del <id> ya declarado")
}
```

B → S

```
B.atributos = S.Atributos
// ".atributos" denota todos los atributos que tenga
```

B → do { C } while (E);

```
If(C.tipo == tipoOK && E.tipo == Logico){
        B.atributos = C.atributos
}else{
        B.tipo = tipoError
}
```

```
// ".atributos" denota todos los atributos que tenga
```

T → number

```
T.tipo = Entero
T.ancho = 1
```

T → boolean

```
T.tipo = Logico
T.ancho = 1
```

T → String

```
T.tipo = String
T.ancho = 64
```

Expresiones

E → E || R

```
If(E1.tipo == R.tipo == logico){
        E.tipo = logico
}else{
        E.tipo = tipoError
}
```

E → R

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

R → R && U

```
If(R1.tipo==U.tipo == Logico){
        R.tipo = logico
}else{
        R.tipo = tipoError
        Error(AND incorrecto)
}
```

R → U

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

U → U == V

```
If(U1.tipo = V.Tipo = Entero){
        U.tipo = logico
}else{
        U.tipo = tipoError
        Error(Igualdad incorrecta)
}
```

U → U != V

```
If(U1.tipo = V.Tipo = Entero){
        U.tipo = logico
}else{
        U.tipo = tipoError
        Error(NegacionIgualdad Incorrectp)
}
```

U → V

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

V → V + W

```
If(V.tipo == W.tipo == Entero){
        V.tipo = Entero
}else{
        V.tipo = tipoError
        Error(Suma incorrecta)
}
```

V → V - W

```
If(V.tipo == W.tipo == Entero){
        V.tipo = Entero
}else{
        V.tipo = tipoError
        Error(Resta incorrecta)
}
```

V → W

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

W → id

```
tipoId = BuscaTipoTs(id.pos)
if(tipoId == Empty){
        W.tipo = Entero
        insertarTipoDesplTSGlobal(id.pos, entero,1)
}else{
        W.tipo = tipoId
}
```

W → (E)

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

W → id (L)

```
If(isFuncion(id.pos)){
If(buscaTipoTS(id.pos) = L.tipos(t){
        W.tipo = t
}else{
        W.tipo = tipoError
        Error(Argumentos no correctos)
}
}else{
        Error(Funcion No declarada)
}
```

W → entero

```
W.tipo = Entero
```

W → cadena

```
W.tipo = String
```

Argumentos de la función

L → E Q

```
L.tipo = E.tipo x Q.tipo
```

L → λ

```
L.tipo = Empty
```

Q → , E Q

```
Q.tipo = E.tipo x Q1.tipo
```

Q → λ

```
Q.tipo = Empty
```

Return

X → E

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

X → λ

```
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