UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROSINFORMÁTICOS

CURSO 2023 – 2024

“Compilador”

AUTORES:

Manuel Martínez Cano – c200077

Lucía Sánchez Navidad – c200034

Grupo 48

Contenido

1. Diseño Final del Proyecto

Para la implementación de la práctica se han tenido en cuenta las opciones correspondientes al grupo. Antes de describir estas opciones, también cabe a resaltar, que hemos visto oportuno realizar dicha práctica en el lenguaje de programación java.

- Sentencias: **Sentencia repetitiva (for)**
- Operadores especiales: **Pre-auto-decremento (-- como prefijo)**
- Técnicas de Análisis Sintáctico: **Descendente con tablas**
- Comentarios: **Comentario de bloque (/* */)**
- Cadenas: **Con comillas simples (' ')**

Opciones de nuestro grupo

Para la correcta implementación del código se han tenido que añadir una sería de librerías:

```
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.PrintWriter;
import java.util.Hashtable;
import java.util.Map;
import java.util.Scanner;
import java.util.Stack;
```

1.1 Funciones

Antes de poder utilizarla tiene que haber sido definida. Pueden recibir cualquiera de los tipos básicos del lenguaje (entero, lógico o cadena). Y a menos que la función omite el tipo en la declaración, está devolverá el mismo tipo que se ha usado para la declaración

function [Tipo] nombre ([argumento1, argumento 2, …]) { [sentencias] }

Lo que se encuentra entre [ ] significa que es opcional ponerlo

1.2 Sentencias

Las sentencias que se han implementado son:

- Asignación (**id = expresión**): no existe la conversión de tipos por tanto, tanto el lado izquierdo como el derecho tienen que tener el mismo tipo
- Condición simple (**if (condición) sentencia**): la condición tiene que ser true para que se pueda/n ejecutar la/s sentencia/s. En caso de que no sea, se acabaría la ejecución
- Retorno de una función (**return expresión**): si la función ha sido declarada sin tipo, el retorno de función no se pondría, por tanto, es opcional dependiendo del código
- Llamada a una función (nombreDeLaFuncion ([argumento1, …])): el número de argumentos nombrados tiene que coincidir con los de la función declarada con el mismo nombre
- Sentencia repetitiva (for (inicialización ; condición ; actualización) {sentencia/s}): la inicialización se realiza mediante una asignación, por tanto, no se puede poner int i = 0 ya que sobraría el int

1.3 Entrada/ Salida

La única instrucción de entrada/ salida que hay que implementar es: **put expresión**

Está instrucción evalúa la expresión e imprime el resultado por pantalla. La expresión puede ser tipo cadena o entera

1.4 Declaraciones

La declaración de variables es de la siguiente forma: **var id T**

var – variable
T – tipo de la variable

1.5 Tipos de Datos

Constante entera → tipo = int
Constante cadena → tipo = string
Constante lógica → tipo = bool

1.6 Identificadores

Los nombres de los identificadores están compuestos por cualquier cantidad de letras, dígitos y subrayados. Pero la primera letra tiene que ser siempre o una letra o un subrayado. Y se distinguen entre minúsculas y mayúsculas

1.7 Operadores

Los operadores implementados son los siguientes:

- Aritméticos: -
- Lógicos: &&
- Relacionales: >
- Asignación: =
- Especial: pre-auto-decremento (-- como prefijo)

1.8 Constantes

Hay 3 tipos de constantes, que son:

- Enteras: se utilizan dígitos decimales, se tiene que poder representar con una palabra (16 bits), por lo que el máximo entero válido es 32767
- Cadenas de Caracteres: puede aparecer cualquier carácter imprimible y van encerradas entre comillas simples (‘ ‘)
- Lógicas

1.9 Comentarios

Como se puede observar en la foto de arriba, el comentario obtenido es /* */. Esto quiere decir que todo lo que se escriba entre esos símbolos será parte del comentario. El comentario puede ir en cualquier parte del lenguaje y no genera token

2. Diseño Analizador Léxico

El formato que deben seguir los tokens es:

del* < del* código del* , del* [atributo] del* > del* RE

del* → es cualquier cantidad de espacios en blanco o tabuladores, o nada
código → el código del token correspondiente. Son caracteres alfanuméricos. Mínimo hay 1
atributo →
→ el atributo es opcional. Pueden ser caracteres alfanuméricos con mínimo 1, o un número entero con signo opcional o una cadena de caracteres
RE → salto de línea o fin de fichero

2.1 Tokens

Estos son los tokens reconocidos por el analizador léxico:

<bool, - >	<for, - >
<func, - >	<if, - >
<get, - >	<int, - >
<let, - >	<put, - >
<return, - >	<string, - >
<cad, 'lex'>	<ig, - >
<coma, - >	<pc, - >
<pa, - >	<p, - >
<la, - >	<lc, - >
<por, - >	<res, - >
<div, - >	<and, - >
<may, - >	<ad, - >
<d, ent>	<id, posTS>
<void, >	

Leyenda:
pa -> paréntesis de abrir
la -> llave de abrir ig -> igual pc -> punto y coma p -> paréntesis de cerrar lc -> llave de cerrar may -> mayor que d -> dígito
ad -> autodecremento

Para poder identificar de forma más rápida las palabras reservadas se ha creado un array de nombre tokens donde se han guardado todas las palabras reservadas

El token d, es decir, el token de un entero contiene directamente el valor del entero correspondiente, al igual que el token cadena (cad). Mientras que los tokens de identificadores (id) tienen guardado la posición de la tabla de símbolos

2.2 Gramática Regular

La gramática regular es:

0 $A \rightarrow dA \mid lB \mid dC \mid \_B \mid , \mid ; \mid (\mid) \mid \{ \mid \} \mid > \mid -M \mid \&P \mid /S \mid * \mid =$ (8 9 10 11 12 13 14 15 17)

1 $B \rightarrow lB \mid dB \mid \_B \mid \lambda$

2 $C \rightarrow dC \mid \lambda$

3 $M \rightarrow - \mid \lambda$

4 $P \rightarrow \& 18$

5 $S \rightarrow *D$

6 $D \rightarrow c_1 D \mid *F$ $\quad c_1 := car - \{`*'\}$

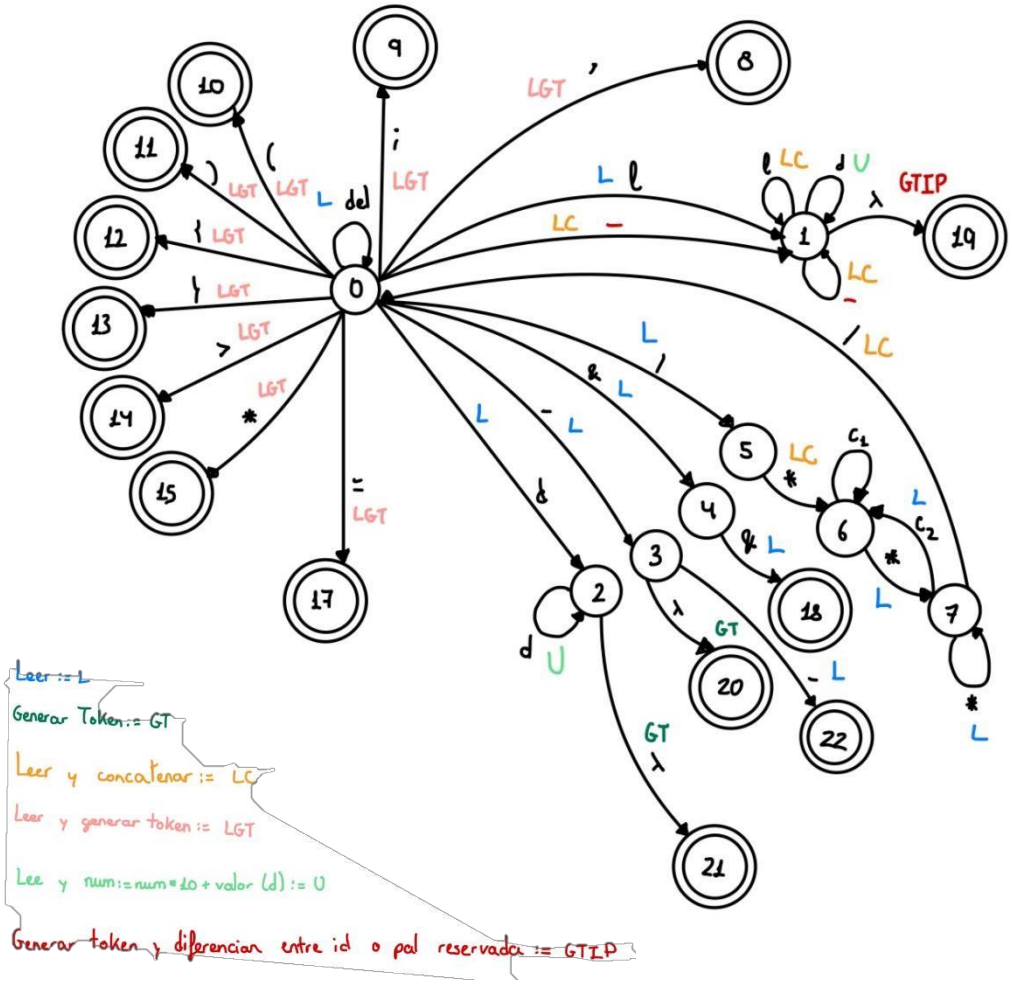
7 $F \rightarrow /A \mid c_2 D \mid *F$ $\quad c_2 := car - \{`*', `/'\}$

$l = \{a-z \mid U \mid A-Z\}$

$d = \{0-9\}$

2.3 Autómata y Acciones Semánticas

Por cada GenerarToken, se usa una acción semántica distinta



2.4 Errores

Todas las transiciones no especificadas son caso de error. Los casos más significativos son:

- Si se encuentra algún elemento no presente en el lenguaje
- Si se pasa el número del rango (número > 32767)
- Si se pasa el string del rango (string > 64)

3. Diseño Analizador Sintáctico

3.1 Gramática

0. PZ -> P
1. P -> D P
2. P -> F P
3. P -> lambda
4. E -> R E'
5. E' -> && R E'
6. E' -> lambda
7. R -> U R'
8. R' -> > U R'
9. R' -> lambda
10. U -> C U'
11. U' -> - C U'
12. U' -> lambda
13. C -> id C'
14. C -> (E)
15. C -> ent
16. C -> cad
17. C' -> = ent
18. C' -> (L)
19. C' -> lambda
20. S -> id S'
21. S -> Y
22. S' -> = E ;
23. S' -> (L) ;
24. S' -> ;
25. L -> E Q
26. L -> lambda
27. Q -> , E Q
28. Q -> lambda
29. S -> put E ;
30. S -> get id ;
31. S -> return X ;
32. X -> E
33. X -> lambda
34. D -> if (E) S
35. D -> S
36. D -> let id T ;
37. T -> int
38. T -> boolean
39. T -> string
40. M -> int
41. M -> boolean
42. M -> string
43. M -> void

44. D -> for (Z ; E ; W) { I }
45. F -> function id H (G) { I }
46. H -> M
47. G -> void
48. G -> T id K
49. G -> lambda
50. K -> , T id K
51. K -> lambda
52. I -> D I
53. I -> lambda
54. Z -> id = E
55. Z -> lambda
56. W -> -- id
57. W -> id = E
58. W -> lambda
59. Y -> -- id

3.2 Tabla LL(1)

	$	&&	>	-	id	(	)	ent	cad	=	:	,	put	get	return	if	let	int	boolean	string	void	for	{	}	function	--
S	S ::= PZ $				S ::= PZ $								S ::= PZ $	S ::= PZ $	S ::= FZ $	S ::= PZ $	S ::= PZ $					S ::= PZ $			S ::= PZ $	S ::= PZ $
PZ	PZ ::= P				PZ ::= P								PZ ::= P	PZ ::= P	PZ ::= P	PZ ::= P	PZ ::= P					PZ ::= P			PZ ::= P	PZ ::= P
P	P ::= ε				P ::= O P								P ::= O P	P ::= O P	P ::= O P	P ::= O P	P ::= O P					P ::= O P			P ::= F P	P ::= O P
E					E ::= R EZ	E ::= R EZ		E ::= R EZ	E ::= R EZ																	
EZ			EZ ::= && R EZ								EZ ::= ε	EZ ::= ε														
R					R ::= U RZ	R ::= U RZ		R ::= U RZ																		
RZ			RZ ::= > U RZ				RZ ::= ε				RZ ::= ε	RZ ::= ε														
U					U ::= C UZ	U ::= C UZ		U ::= C UZ	U ::= C UZ																	
UZ			UZ ::= ε	UZ ::= - C UZ			UZ ::= ε				UZ ::= ε	UZ ::= ε														
C					C ::= id OZ	C ::= (E)		C ::= ent	C ::= cad																	
OZ			OZ ::= ε			OZ ::= (L)	OZ ::= ε			OZ ::= = E	OZ ::= ε	OZ ::= ε														
N					N ::= id NZ								N ::= put E ;	N ::= get id ;	N ::= return X ;											
NZ						NZ ::= (L) ;				NZ ::= = E ;	NZ ::= ;															Ñ ::= Y
L					L ::= E Q	L ::= E Q		L ::= E Q	L ::= E Q																	
Q							O ::= ε					O ::= , E Q														
X					X ::= E	X ::= E		X ::= E	X ::= E		X ::= ε															
D					D ::= Ñ								D ::= Ñ	D ::= Ñ	D ::= Ñ	D ::= if (E) Ñ	D ::= let id T ;					D ::= for (Z ; E ; W) { I }			D ::= Ñ	
T																		T ::= int	T ::= boolean	T ::= string							
M																		M ::= int	M ::= boolean	M ::= string	M ::= void						
F																									F ::= function id H (G) { I }		
H																		H ::= M	H ::= M	H ::= M	H ::= M						
G							G ::= ε											G ::= T id K	G ::= T id K	G ::= T id K	G ::= void						
K							K ::= ε					K ::= T id K															
I					I ::= O I								I ::= O I	I ::= O I	I ::= O I	I ::= O I	I ::= D I					I ::= O I		I ::= ε		I ::= O I
Z					Z ::= id OZ						Z ::= ε															
W					W ::= id NZ		W ::= ε																			W ::= M
Y																										Y ::= Z = E

3.3 Comprobación LL(1)

First(PZ) = {if, id, put, get, return, let, for,
function, --}
Follow(PZ) = {$}

First(P) = {if, id, put, get, return, let, for,
function, --}
Follow(P) = {$}

First(E) = {id, (, ent, cad}
Follow(E) = {), ;, ,}

First(EZ) = {&&}
Follow(EZ) = {), ;, ,}

First(R) = {id, (, ent, cad}
Follow(R) = {&&,), ;, ,}

First(RZ) = {>}
Follow(RZ) = {&&,), ;, ,}

First(U) = {id, (, ent, cad}
Follow(U) = {&&, >,), ;, ,}

First(UZ) = {-}
Follow(UZ) = {&&, >,), ;, ,}

First(C) = {id, (, ent, cad}
Follow(C) = {&&, >, -,), ;, ,}

First(CZ) = {=, (}
Follow(CZ) = {&&, >, -,), ;, ,}

First(S) = {id, put, get, return, --}
Follow(S) = {if, id, put, get, return, let, for,
function, --, }, $}

First(SZ) = {=, (, ;}
Follow(SZ) = {if, id, put, get, return, let, for,
function, --, }, $}

First(L) = {id, (, ent, cad}
Follow(L) = {)}

First(Q) = {,}
Follow(Q) = {)}

First(X) = {id, (, ent, cad}
Follow(X) = {;}

First(D) = {if, id, put, get, return, let, for, --}
Follow(D) = {if, id, put, get, return, let, for,
function, --, }, $}

First(T) = {int, boolean, string}
Follow(T) = {;, id}

First(M) = {int, boolean, string, void}
Follow(M) = {(}

First(F) = {function}
Follow(F) = {if, id, put, get, return, let, for,
function, --, $}

First(H) = {int, boolean, string, void}
Follow(H) = {(}

First(G) = {void, int, boolean, string}
Follow(G) = {)}

First(G) = {void, int, boolean, string}
Follow(G) = {)}

First(K) = {,}
Follow(K) = {)}

First(I) = {if, id, put, get, return, let, for, --}
Follow(I) = {}}

First(Z) = {id}
Follow(Z) = {;}

First(W) = {--, id}
Follow(W) = {)}

First(Y) = {--}
Follow(Y) = {if, id, put, get, return, let, for, function, --, }, $}

Para cada par de producciones $A \to \alpha \mid \beta$:

1. $First(\alpha) \cap First(\beta) = \emptyset$
2. Si $\beta \to \lambda$, $First(\alpha) \cap Follow(A) = \emptyset$

Por tanto, la gramática es LL(1)

4 Diseño Analizador Semántico

PZ -> {TS := CrearTabla(); despl := 0; zona_decl := true, retorno = false} P [illegible]uirTS(TS)}

P -> D {zonaDecl=false} P1
P -> F P
P -> lambda {}

E -> R {EZ.tipo::=R.tipo} EZ {E.tipo::=EZ.tipo}

EZ -> && R EZ {if(EZ.tipo!=logico || R.tipo!=logico) then EZ.tipo :=tipo_error (La operacion logica AND solo puede usarse con logicos).}

EZ -> lambda {}

R -> U {RZ.tipo::=U.tipo} RZ {R.tipo::=RZ.tipo}

RZ -> > U RZ {if(RZ.tipo!=entero || U.tipo!=entero) then RZ.tipo :=tipo_error (Los operadores relacionales solo pueden usarse con enteros) else RZ.tipo::=logico}

RZ -> lambda {}

U -> C {UZ.tipo := C.tipo} UZ {U.tipo := UZ.tipo}

UZ -> - C UZ {if (UZ.tipo!=entero || C.tipo!=entero) then UZ.tipo:=tipo_error(No se puede restar algo que no sean enteros)}

UZ -> lambda {}

C -> id CZ {
    if(buscaTipo(TS, id)==null then (insertaTipo(TS, id.pos, entero)
    if(TS==TSG) then insertaDesplazamiento(TS, id.pos, despG) despG=despG+1 else insertaDesplazamiento (TS, id.pos, despL)) despL=despL+1
    if(CZ.tipo!=vacio) then (
        if(buscaTipo(TS, id.pos)==CZ.tipo) then C.tipo::=CZ.tipo
        else C.tipo::=tipo_error(el tipo de id es incorrecto))
    else C.tipo::=buscaTipo(TS, id.pos) }

C -> (E) {C.tipo = E.tipo}
C -> ent {C.tipo = ent}
C -> cad {C.tipo = cad}

CZ -> = ent {CZ.tipo = ent}
CZ -> (L) {CZ.tipo = L.tipo}
CZ -> lambda {}

S -> id SZ {S.tipo :=
    if(buscaTipo(TS, id)==null then (insertaTipo(TS, id.pos, entero)

```
                if(TS==TSG) then insertaDesplazamiento(TS, id.pos, despG) despG=despG+1
                else insertaDesplazamiento (TS, id.pos, despL)) despL=despL+1
        else BuscarTipoTS(TS, id.pos)//Hay que diferenciar entre functionciones y variables
                idTipo = buscaTipoTS(pos)
          if(idTipo == function){
            numParam = NumParamTS(pos)
            tipoParam =  TipoParamTS(pos)
            if(SZ.numParam == numParam && SZ.tipoParam == tipoParam)
                    S.tipo = tipo_ok
            else
                 S.tipo = tipo_error
          }
          else if(idTipo == SZ.tipo)  S.tipo = tipo_ok }


S -> Y {S.tipo := if (Y.tipo==entero) then tipo_ok else tipo_error }

SZ -> = E ; {SZ.tipo = E.tipo}
SZ -> ( L ) ;        SZ.tipo = function
        SZ.numParam = L.numParam
        SZ.tipoParam = L.tipoParam
SZ -> ; {}

L -> E Q if(Q.tipo == vacío) {
          L.tipoParam = E.tipo
          L.numParam = 1
               }
         else
          L.tipoParam = E.tipo  y Q.tipoParam
          L.numParam = Q.numParam + 1
        }
L -> lambda {L.tipo::=vacio}

Q -> , E Q1 if(Q.tipo == vacío) {
          Q.tipoParam = E.tipo
          Q.numParam = 1
                    }
        else{
          Q.tipoParam = E.tipo y  Q''.tipoParam
          Q.numParam = Q''.numParam + 1
        }
Q -> lambda {Q.tipo := vacio}

S -> put E ; {S.tipo := if(E.tipo==entero || E.tipo==cadena) then tipo_ok else tipo_error(put solo se
puede usar con enteros o cadenas) }
S -> get id ; {S.tipo :=
        if(buscaTipo(TS, id))==null then (insertaTipo(TS, id.pos, entero)
        if(TS==TSG) then insertaDesplazamiento(TS, id.pos, despG) despG=despG+1
```

```
        else insertaDesplazamiento (TS, id.pos, despL)) despL=despL+1;
        if(buscaTipo(TS, id)==entero || buscaTipo(TS, id)==cadena) then tipo_ok
        else tipo_error(get solo se puede usar con enteros o cadenas)
S -> return X ; if(TSA == TSL){
            tipoRetorno = buscaTipoRetornoTS(funActual)
            if(X.tipo == tipoRet)  S.tipo = tipo_ok,   S.tipoRet = tipoRetorno
            else S.tipo = tipo_error, S.tipoRet = tipo_error
          }
          else S.tipo = tipo_error}

X -> E {X.tipo = E.tipo}
X -> lambda {X.tipo := vacio}

D -> if ( E ) S {D.tipo = if (E.tipo == logico) then S.tipo else tipo_error(La expresion del if debe ser
logica)}
D -> S {D.tipo = S.tipo, D.tipoRet = S.tipoRet}
D -> let id T ; {AñadeTipoTS(id.pos, T.tipo)}

T -> int {T.tipo = ent, T.ancho = 1}
T -> boolean {T.tipo = logico, T.ancho = 1}
T -> string {T.tipo = cad, T.ancho = 64}

M -> int {M.tipo = ent, T.ancho = 1}
M -> boolean {M.tipo = logico, T.ancho = 1}
M -> string {M.tipo = cad, T.ancho = 64}
M -> void {M.tipo = vacio}

D -> for ( Z ; E ; W ) { I } {D.tipo := if(Z.tipo==tipo_ok && E.tipo==logico && W.tipo!=tipo_error &&
I.tipo != tipo_error) then tipo_ok else tipo_error}


 F -> function id H
                        zonaDecl = true
                                TSL = crearTSL()
                    TSactual = TSL
                    functionActual = id
                    desplLocal = 0
                    insertarTipoTS(id.pos, function,tablaSimGlobal)
                    tipoRetorno = H.tipo
                    insertarTipoRetornoTS(id.pos, tipoRetorno, tablaSimGlobal)

( G )                insertarTipoParamTS(functionActual.pos,G.tipoParam)
                    insertarNParamTSG(functionActual.Pos, G.numParam)
                    zonaDecl = false
                    if(G.tipo == tipo_error){
                       F.tipo = tipo_error
                                        }
```

```
{ I }          if(!H.tipo == vacio && !tieneRetorno)
                 F.tipo = tipo_error
               else if ( I.tipo == vacio && !tipoRetorno == vacío){
                                   F.tipo = tipo_error
               }else     F.tipo = tipo_ok
               destruirTSL()

H -> M {H.tipo := M.tipo}

G -> void {G.tipo := vacio}
G -> T id K {AñadeTipoTS(TS, id.pos, T.tipo),  insertaDesplazamiento(TS, id.pos, despL),
despL=despL+T.ancho}
G -> lambda {G.tipo := vacio}

K -> , T id K {AñadeTipoTS(TS, id.pos, T.tipo), insertaDesplazamiento(TS, id.pos, despL),
despL=despL+T.ancho}
K -> lambda {K.tipo := vacio}

I -> D I1 if(D.tipo == tipo_error) I.tipo  = tipo_error
           if(I1.tipo == vacio)  I.tipo = tipo_ok
           else  I.tipo = I1.tipo
I -> lambda {I.tipo := vacio}

Z -> id = E {Z.tipo :=
        if(buscaTipo(TS, id))==null then (insertaTipo(TS, id.pos, entero)
        if(TS==TSG) then insertaDesplazamiento(TS, id.pos, despG) despG=despG+1
        else insertaDesplazamiento(TS, id.pos, despL)) despL=despL+1 ;
        if (BuscaTipoTS(TS, id.pos) = E.tipo) then tipo_ok
        else tipo_error(Ambos lados de la asignacion deben representar la misma clase de datos)}
Z -> lambda {Z.tipo = vacio}

W -> -- id {W.tipo :=
        if(buscaTipo(TS, id))==null then (insertaTipo(TS, id.pos, entero)
        if(TS==TSG) then insertaDesplazamiento(TS, id.pos, despG) despG=despG+1
        else insertaDesplazamiento(TS, id.pos, despL)) despL=despL+1 ;
        if (BuscaTipoTS(TS, id.pos) == ent) then tipo_ok
        else tipo_error}
W -> id = E {W.tipo =
        if(buscaTipo(TS, id))==null then (insertaTipo(TS, id.pos, entero)
        if(TS==TSG) then insertaDesplazamiento(TS, id.pos, despG) despG=despG+1
        else insertaDesplazamiento(TS, id.pos, despL)) despL=despL+1 ;
        if (BuscaTipoTS(TS, id.pos) = E.tipo) then tipo_ok else tipo_error}
W -> lambda {}

Y -> -- id {Y.tipo :=
        if(buscaTipo(TS, id))==null then (insertaTipo(TS, id.pos, entero)
        if(TS==TSG) then insertaDesplazamiento(TS, id.pos, despG) despG=despG+1
        else insertaDesplazamiento(TS, id.pos, despL)) despL=despL+1 ;
```

```
if (BuscaTipoTS(TS, id.pos) == ent) then tipo_ok
else tipo_error}
```

6. Anexo

El anexo esta compuesto por 5 casos corrector y 5 casos error para mostrar el comportamiento del Procesador.

El manejo de errores de nuestro Procesador es el siguiente. Cada vez que encuentra 1 error el programa finaliza y mantiene la información generada de tabla de símbolos, analizador léxico y parse. A excepción de los errores semánticos que borra lo guardado con anterioridad en estos archivos

Y por último, la siguiente aclaración respecto a los árboles. Debido al gran tamaño de alguno de los árboles, este se guardará en formato html en la carpeta donde se encuentran el resto de ficheros a entregar y en su correspondiente código se indicará el nombre del árbol

6.1 Caso Correcto 1

Código

```
let posicion int;

function algo int() {
for (i = 10; i >0; i--) {
posicion = posicion + i;
}
print posicion;
print 'Resultado correcto';
return posicion;
}
```

Tokens

<let, - >
<id, 0>
<int, - >
<pc, - >
<func, - >
<id, 1>
<int, - >
<pa, - >
<p, - >
<la, - >
<for, - >
<pa, - >
<id, 1>
<ig, - >
<d, 10>
<pc, - >
<id, 1>
<may, - >
<d, 0>
<pc, - >
<ad, - >
<id, 1>
<p, - >
<la, - >
<id, 0>
<ig, - >
<id, 0>
<res, - >
<id, 1>
<pc, - >
<lc, - >
<print, - >
<id, 0>
<pc, - >
<print, - >
<cad, 'Resultado correcto'>
<pc, - >
<return, - >
<id, 0>
<pc, - >
<lc, - >

Árbol

En la carpeta Caso Correcto 1, el fichero arbol caso correcto 1 que es .html

Tabla de Símbolos

6.2 Caso Correcto 2

Código

```
let palabra int;
let s int;
s = palabra - s ;
function algo string (int a) {
for (i = 10; i > 0; --i) {
if (i > 0) print atributo;
palabra = 'algo';
x = x/2;
}
print palabra;
print 'Resultado correcto';
input x;
return palabra;
}
```

Tokens

<let, - >
<id, 0>
<int, - >
<pc, - >
<let, - >
<id, 1>
<int, - >
<pc, - >
<id, 1>
<ig, - >
<id, 0>
<res, - >
<id, 1>
<pc, - >
<func, - >
<id, 2>
<string, - >
<pa, - >
<int, - >
<id, 2>
<p, - >
<la, - >
<for, - >
<pa, - >
<id, 2>
<ig, - >
<d, 10>
<pc, - >
<id, 2>
<may, - >
<d, 0>
<pc, - >
<ad, - >
<id, 2>
<p, - >
<la, - >
<if, - >
<pa, - >
<id, 2>
<may, - >
<d, 0>
<p, - >
<print, - >
<id, 2>
<pc, - >
<id, 0>
<ig, - >
<cad, 'algo'>
<pc, - >
<id, 2>
<ig, - >
<id, 2>
<div, - >
<d, 2>
<pc, - >
<lc, - >
<print, - >
<id, 0>
<pc, - >
<print, - >
<cad, 'Resultado correcto'>
<pc, - >
<input, - >
<id, 2>
<pc, - >
<return, - >
<id, 0>
<pc, - >
<lc, - >

Árbol

En la carpeta Caso Correcto 2, el fichero arbol caso correcto 2 que es .html

Tabla de Símbolos

6.3 Caso Correcto 3

Código

```
let x int;
let y int;
let z int;

z = x-y;
print --z;
```

Tokens

<let, - >
<id, 0>
<int, - >
<pc, - >
<let, - >
<id, 1>
<int, - >
<pc, - >
<let, - >
<id, 1>
<int, - >
<pc, - >
<id, 1>
<ig, - >
<id, 0>
<res, - >
<id, 1>
<pc, - >
<print, - >
<ad, - >
<id, 1>
<pc, - >

Árbol

En la carpeta Caso Correcto 3, el fichero arbol caso correcto 3 que es .html

Tabla de Símbolos

6.4 Caso Correcto 4

Código

```
function mayor string (int a) {
let numero int;
let cadena string;
cadena = 'b es mayor';
if (a>b) cadena = 'a es mayor';

return cadena;
}
```

Tokens

<func, - >
<id, 0>
<string, - >
<pa, - >
<int, - >
<id, 1>
<p, - >
<la, - >
<let, - >
<id, 1>
<int, - >
<pc, - >
<let, - >
<id, 1>
<string, - >
<pc, - >
<id, 1>
<ig, - >
<cad, 'b es mayor'>
<pc, - >
<if, - >
<pa, - >
<id, 1>
<may, - >
<id, 1>
<p, - >
<id, 1>
<ig, - >
<cad, 'a es mayor'>
<pc, - >
<return, - >
<id, 1>
<pc, - >
<lc, - >

Árbol

En la carpeta Caso Correcto 4, el fichero arbol caso correcto 4 que es .html

Tabla de Símbolos

6.5 Caso Correcto 5

Código

```
let comprueba boolean;
let encontrado boolean;
let x int;

if (comprueba) print comprueba;

for (i = 10; i > 0; --i) {
input x;
}
```

Tokens

<let, - >
<id, 0>
<bool, - >
<pc, - >
<let, - >
<id, 1>
<bool, - >
<pc, - >
<let, - >
<id, 1>
<int, - >
<pc, - >
<if, - >
<pa, - >
<id, 0>
<p, - >
<print, - >
<id, 0>
<pc, - >
<for, - >
<pa, - >
<id, 1>
<ig, - >
<d, 10>
<pc, - >
<id, 1>
<may, - >
<d, 0>
<pc, - >
<ad, - >
<id, 1>
<p, - >
<la, - >
<input, - >
<id, 1>
<pc, - >
<lc, - >

Árbol

En la carpeta Caso Correcto 5, el fichero arbol caso correcto 5 que es .html

Tabla de Símbolos

6.6 Caso Erróneo 1

Código

```
let x int;
x = 32768;
let palabra string;

x = x - palabra;
```

Error

```
Error en linea 2. Numero fuera de rango
```

6.7 Caso Erróneo 2

Código

```
function prueba string (string palabra) {
let palabra string;
palabra = 'asdfasdfasdfasdfasdfasdfasdfasdfasdfasdfasdfasdfasdasdfasdfasd';
return palabra;
}
```

Error

```
Error en linea 3. Cadena fuera de rango
```

6.8 Caso Erróneo 3

Código

```
function prueba int (int x) {
x = x + x;
}
```

Error

```
Error en linea 2. + no esta reconocido en el lenguaje
```

6.9 Caso Erróneo 4

Código

```
function prueba int (int x) {
if (x < x)
return x;
}
```

Error

```
Error en linea 2. < no esta reconocido en el lenguaje
```

6.10 Caso Erróneo 5

Código

```
function prueba int (int x) {
let comprueba int;
if (x != comprueba)
return x;
}
```

Error

```
Error en linea 3. ! no esta reconocido en el lenguaje
```