2022

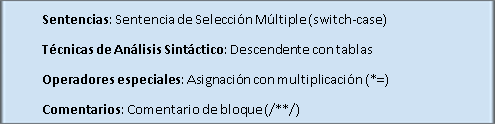
PRÁCTICA PROCESADORES DEL LENGUAJE

GRUPO 13

INTEGRANTES:

* Mario López Estaire
* Andrés Bravo Francos
* Grettell Umpierrez Sardiñas

2022



* Comentarios → /\*...\*/
* Constantes →
  + Enteras → d⁺ → 16b(1 palabra) → máx (32767)
  + Cadena …. → “c\*”🡪 máx. 64 caracteres
  + Lógicas → pal\_reservada(true, false)
* Operadores →
  + Relación → ==
  + Aritméticos → \*
  + Lógico → &&
  + Asignación → =
  + Asignación con multiplicación → \*=
* Identificadores → (l + \_) + (d + \_ + l) \* 🡪 l (a-z, A-Z) 🡪 d(0-9)
* Declaraciones → pal\_reservada(let)
* Datos→
  + Entero → pal\_reservada (int)
  + Lógico→ pal\_reservada(boolean)
  + Cadena → pal\_reservada (string)
* Entrada/Salida → pal\_reservada (print, input).
* Sentencias → pal\_reservada(return, function, if, switch, case, default, break, eof)
* Otros → ( ) { } ; : ,

# ANALIZADOR LÉXICO

**TOKENS**

***Generales***

<abrirParentesis, >

<cerrarParentesis, >

<abrirCorchete, >

<cerrarCorchete, >

<coma, >

<ptoComa, >

<dosPuntos, >

<eof, >

***Operadores***

<opAritmetico,1 >

<opAritmetico, 2 >

<opRelacional,1 >

<opLogico,1 >

<asigMultiplicacion, - >

<asignacion, - >

***Valores de Datos***

<cteEntera, valor>

<cadena, “lexema”>

< true, >

< false, >

**Declaraciones**

<id, posTs>

<let, >

**Tipos de Datos** *(Palabras reservadas)*

<int, >

<boolean, >

<string, >

***Funciones*** *(Palabras reservadas)*

<function, >

<return, >

***Sentencias*** *(Palabras reservadas)*

<if, >

<switch, >

<case, >

<default, >

<break, >

***E/S*** *(Palabras reservadas)*

<input, >

<print, >

Op\_aritmético 1: \*

2: +

Op\_relacional 1: ==

Op\_lógico 1: &&

# Cada Token generado será volcado en un fichero “tokens.txt”.

# GRAMÁTICA

| :

LEYENDA:

: {a-z,A\_Z}

: {0 - 9}

: cualquier carácter no contemplado en el nodo

: todos los caracteres – {\*}

: todos los caracteres – {\*,/}

: todos los caracteres – {“}

: tab, espacio, eol

# AUTÓMATA

# 

# ACCIONES SEMÁNTICAS

# S→S: leer();

# S→A: lexema = c; leer();

# A→A: lexema = lexema + c; leer();

# A→B:

# if(lexema == palabraReservada) then generarToken(lexema, - );

# else{

# pos=BuscarLugarTS(lexema);

# if(pos ¡= null) then generarToken(id, pos);

# else{

# pos=insertarIdTS(lexema);

# generarToken(id,pos);

# }

# S→C: valor = char\_int(d); leer();

# C→C: valor = valor \* 10 + char\_int(d); leer();

# C→D: if (valor > 32767) then error (60);

# else generarToken(cte\_entera, valor);

# S→E: leer();

# E→F: leer();

# F→F: leer();

# F→G: leer();

# G→F: leer();

# G→G: leer();

# G→S: leer();

# S→K: lexema = “ ”; contador=0; leer();

# K→K: lexema = lexema + c; leer();contador++;

# K→L: if(contador > 64 ) then error (61);

# else generarToken(cadena, lexema);

# S→H: leer();

# H→I: generarToken(opRelacional, 1);leer();

# H→J: generarToken(asignación, -);

# S→M: leer();

# M→N: generarToken(asigMultiplicacion, - );leer();

# M→O: generarToken(opAritmetico, 1 );

# S→P: leer();

# P→Q: generarToken(opLogico, 1 ); leer();

# S→R: generarToken(abrirParentesis, - );leer();

# S→T: generarToken(cerrarParentesis, - ); leer();

# S→U: generarToken(abrirCorchete, - ); leer();

# S→V: generarToken(cerrarCorchete, - ); leer();

# S→W: generarToken(ptoComa, - ); leer();

# S→X: generarToken(coma, - ); leer();

# S→Y: generarToken(dosPuntos, - ); leer();

# S→Z: generarToken(opAritmetico, 2 ); leer();

# MATRIZ AFD

LEYENDA:

: {a-z,A\_Z}

: {0 - 9}

: cualquier carácter no contemplado en el nodo

: todos los caracteres – {\*}

: todos los caracteres – {\*,/}

: todos los caracteres – {“}

: tab, espacio, eol

**\*Los estados encerrados EN VERDE representan estados finales (y por tanto no reciben nada).**

**\*Los números EN ROJO representan ERRORES**

# 

# TABLA DE SÍMBOLOS

COMPLETAR

# ERRORES

# Léxicos

# Código 50: Fue introducido un carácter no esperado.

# Código 51: Fue introducido un carácter no válido. Se esperaba \*.

# Código 52: Fue introducido un carácter no esperado.

# Código 53: Fue introducido un carácter no esperado.

# Código 54: Fue introducido un carácter no esperado en la cadena.

# Código 55: Fue introducido un carácter no válido. Se esperaba &.

# Código 60: El número entero introducido está fuera de rango (es mayor que 32767 o menor que -32768).

# Código 61: La cadena supera los 64 caracteres.

# Los errores son volcados en un fichero “errores.txt”

# Cada error en nuestro procesador tendrá la siguiente estructura en el fichero “errores.txt” :

# “Línea ” + *num\_línea* + “ – Código de error ” + *cod\_error* + “ :” + \n\t + *mensaje\_error*

# \n : Salto de línea

# \t : Tabulador

# *num\_línea* : el número de línea en el fichero fuente donde se encuentra el error.

# *cod\_error* : código de error según lo especificado anteriormente ( 50 || 51 || 52 || 53 || 54 || 55 || 60 || 61 )

# *mensaje\_error* : mensaje específico del error en el formato siguente

# *“ERROR “ + tipo\_error* + “ – “ + *mensaje\_descriptivo*

# *tipo\_error : LÉXICO || SINTÁCTICO || SEMÁNTICO*

# *mensaje\_descriptivo:* mensaje descriptivo del error que incluye el carácter, cadena o valor incorrecto, incluyendo el carácter esperado en determinados casos.

# Un ejemplo del contenido de un error generado por nuestro Analizador Léxico:

# Línea 1 - Código de error 51:

# ERROR LÉXICO - carácter CAR: 'l' no válido. Se esperaba \*.

# ANALIZADOR SINTÁCTICO

# GRAMÁTICA

# // Axioma general

# // Sentencias compuestas



// Sentencias simples



// Expresiones

1. // Operaciones lógicas
3. // Operaciones relacionales
4. // Operaciones aritméticas (suma)
5. // Operaciones aritméticas (producto)

// Operandos



// Argumentos de función



// Valor de retorno



// Tipos de variables



// Declaración de funciones



// Inicialización de identificadores

// Cuerpo de switch

# FIRST Y FOLLOW

|  |  |  |
| --- | --- | --- |
|  | FIRST | FOLLOW |
| A | if | let | switch | id | print | input | return | function | eof | eof |
| B | if | let | switch | id | print | input | return | break | if | let | switch | id | print | input | return | break | function | eof | } | case | default |
| S | id | print | input | return | break | if | let | switch | id | print | input | return | break | function | eof | } | case | default |
| S’ | = | \*= | ( | ; |
| E | id | ( | cteEntera | cadena | true | false | ) | ; | , |
| E’ | && | λ | ) | ; | , |
| R | id | ( | cteEntera | cadena | true | false | && | ) | ; | , |
| R’ | == | λ | && | ) | ; | , |
| U | id | ( | cteEntera | cadena | true | false | == | && | ) | ; | , |
| U’ | + | λ | == | && | ) | ; | , |
| V | id | ( | cteEntera | cadena | true | false | + | == | && | ) | ; | , |
| V’ | \* | λ | + | == | && | ) | ; | , |
| P | id | ( | cteEntera | cadena | true | false | \* | + | == | && | ) | ; | , |
| P’ | ( | λ | \* | + | == | && | ) | ; | , |
| L | id | ( | cteEntera | cadena | true | false | λ | ) |
| Q | , | λ | ) |
| X | id | ( | cteEntera | cadena | true | false | λ | ; |
| T | int | boolean | string | id | ( | ; | = | \*= |
| F | function | if | let | switch | id | print | input | return | break | function | eof |
| H | int | boolean | string | λ | ( |
| D | int | boolean | string | λ | ) |
| K | , | λ | ) |
| C | if | let | switch | id | print | input | return | break | } |
| C’ | if | let | switch | id | print | input | return | break | λ | } |
| N | ; | = | \*= | if | let | switch | id | print | input | return | break | function | eof | } | case | default |
| Z | case | default | } |
| O | if | let | switch | id | print | input | return | λ | break | case | default | } |
| O’ | break | λ | case | default | } |

# Justificación

# Verificación condición LL

# Las reglas de A cumplen la condición LL porque:

# porque

# porque

# porque

# No hay regla por tanto no se necesitan más comprobaciones para determinar que se cumple la condición LL

# Las reglas de B cumplen la condición LL porque:

# porque

# porque

# porque

# porque

# porque

# porque

# No hay regla por tanto no se necesitan más comprobaciones para determinar que se cumple la condición LL

# Las reglas de S cumplen la condición LL porque:

# porque

# porque

# porque

# porque

# porque

# porque

# porque

# porque

# porque

# porque

# No hay regla por tanto no se necesitan más comprobaciones para determinar que se cumple la condición LL

# Las reglas de S’ cumplen la condición LL porque:

# porque

# porque

# porque

# No hay regla por tanto no se necesitan más comprobaciones para determinar que se cumple la condición LL

# Las reglas de E cumplen la condición LL porque:

* Solamente tiene una regla por tanto no se necesitan más comprobaciones para determinar que se cumple la condición LL

# Las reglas de E’ cumplen la condición LL porque:

# Solamente hay dos reglas y una de ellas es por tanto no hay que comprobar si los first tienen símbolos terminales en común

# Como una de las reglas es

# porquepor tanto se cumple la condición LL.

# Las reglas de R cumplen la condición LL porque:

* Solamente tiene una regla por tanto no se necesitan más comprobaciones para determinar que se cumple la condición LL

# Las reglas de R’ cumplen la condición LL porque:

# Solamente hay dos reglas y una de ellas es por tanto no hay que comprobar si los first tienen símbolos terminales en común

# Como una de las reglas es

# porquepor tanto se cumple la condición LL.

# Las reglas de U cumplen la condición LL porque:

* Solamente tiene una regla por tanto no se necesitan más comprobaciones para determinar que se cumple la condición LL

# Las reglas de U’ cumplen la condición LL porque:

# Solamente hay dos reglas y una de ellas es por tanto no hay que comprobar si los first tienen símbolos terminales en común

# Como una de las reglas es

# porquepor tanto se cumple la condición LL.

# Las reglas de V cumplen la condición LL porque:

* Solamente tiene una regla por tanto no se necesitan más comprobaciones para determinar que se cumple la condición LL

# Las reglas de V’ cumplen la condición LL porque:

# Solamente hay dos reglas y una de ellas es por tanto no hay que comprobar si los first tienen símbolos terminales en común

# Como una de las reglas es

# porquepor tanto se cumple la condición LL.

# Las reglas de P cumplen la condición LL porque:

# porque

# porque

# porque

# porque

# No hay regla por tanto no se necesitan más comprobaciones para determinar que se cumple la condición LL

# Las reglas de P’ cumplen la condición LL porque:

# Solamente hay dos reglas y una de ellas es por tanto no hay que comprobar si los first tienen símbolos terminales en común

# Como una de las reglas es

# porquepor tanto se cumple la condición LL.

# Las reglas de L cumplen la condición LL porque:

# Solamente hay dos reglas y una de ellas es por tanto no hay que comprobar si los first tienen símbolos terminales en común

# Como una de las reglas es

# porquepor tanto se cumple la condición LL.

# Las reglas de Q cumplen la condición LL porque:

# Solamente hay dos reglas y una de ellas es por tanto no hay que comprobar si los first tienen símbolos terminales en común

# Como una de las reglas es

# porquepor tanto se cumple la condición LL.

# Las reglas de X cumplen la condición LL porque:

# Solamente hay dos reglas y una de ellas es por tanto no hay que comprobar si los first tienen símbolos terminales en común

# Como una de las reglas es

# porquepor tanto se cumple la condición LL.

# Las reglas de T cumplen la condición LL porque:

# porque

# porque

# porque

# No hay regla por tanto no se necesitan más comprobaciones para determinar que se cumple la condición LL

# Las reglas de F cumplen la condición LL porque:

* Solamente tiene una regla por tanto no se necesitan más comprobaciones para determinar que se cumple la condición LL

# Las reglas de H cumplen la condición LL porque:

# Solamente hay dos reglas y una de ellas es por tanto no hay que comprobar si los first tienen símbolos terminales en común

# Como una de las reglas es

# porquepor tanto se cumple la condición LL.

# Las reglas de D cumplen la condición LL porque:

# Solamente hay dos reglas y una de ellas es por tanto no hay que comprobar si los first tienen símbolos terminales en común

# Como una de las reglas es

# porquepor tanto se cumple la condición LL.

# Las reglas de K cumplen la condición LL porque:

# Solamente hay dos reglas y una de ellas es por tanto no hay que comprobar si los first tienen símbolos terminales en común

# Como una de las reglas es

# porquepor tanto se cumple la condición LL.

# Las reglas de C cumplen la condición LL porque:

* Solamente tiene una regla por tanto no se necesitan más comprobaciones para determinar que se cumple la condición LL

# Las reglas de C’ cumplen la condición LL porque:

# Solamente hay dos reglas y una de ellas es por tanto no hay que comprobar si los first tienen símbolos terminales en común

# Como una de las reglas es

# porque por tanto se cumple la condición LL.

# Las reglas de N cumplen la condición LL porque:

# porque

# porque

# porque

# No hay regla por tanto no se necesitan más comprobaciones para determinar que se cumple la condición LL

# Las reglas de Z cumplen la condición LL porque:

# porque

# No hay regla por tanto no se necesitan más comprobaciones para determinar que se cumple la condición LL

# Las reglas de O cumplen la condición LL porque:

# Solamente hay dos reglas y una de ellas es por tanto no hay que comprobar si los first tienen símbolos terminales en común

# Como una de las reglas es

# porque

# Las reglas de Z’ cumplen la condición LL porque:

# Solamente hay dos reglas y una de ellas es por tanto no hay que comprobar si los first tienen símbolos terminales en común

# Como una de las reglas es

# porque

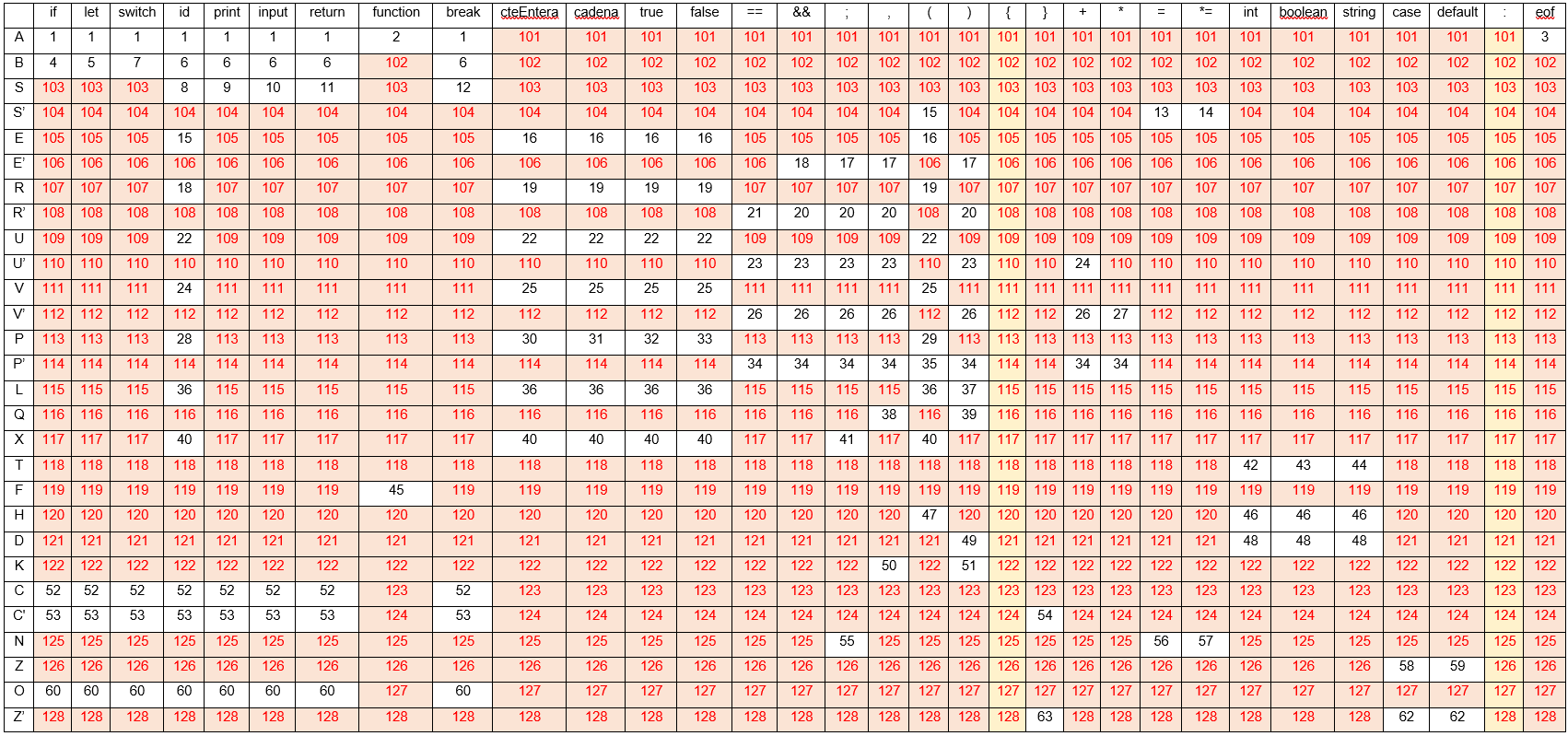
# La gramática obtenida para el Analizador Sintáctico Descendente LL(1) con Tablas es LL(1) porque:

# Para cada no terminal para el que haya más de una regla de producción, dichas reglas no derivan un mismo terminal (la gramática está factorizada).

# Cumple la condición LL

# No existe recursividad por la izquierda

**TABLA A.S DESCENDENTE (LL)**



|  |  |  |
| --- | --- | --- |
|  |  | Error Sintáctico |

# ERRORES

# Sintácticos

# Código 100: Token *siguiente\_token* no esperado. Se esperaba *cima\_pila*.

# Código 101: El token *siguiente\_token* no pertenece al primer elemento de una sentencia válida, una función o es fin de fichero.

# Código 102: El token *siguiente\_token* no pertenece al primer elemento de una sentencia válida.

# Código 103: El token *siguiente\_token* no pertenece al primer elemento de una sentencia simple válida.

# Código 104: El token *siguiente\_token* no es una asignación válida o la llamada a una función.

# Código 105: El token *siguiente\_token* no pertenece a una expresión válida.

# Código 106: El token *siguiente\_token* no pertenece a una expresión lógica u otra expresión válida.

# Código 107: El token *siguiente\_token* no pertenece a una expresión válida.

# Código 108: El token *siguiente\_token* no pertenece a una expresión relacional u otra expresión válida.

# Código 109: El token *siguiente\_token* no pertenece a una expresión válida.

# Código 110: El token *siguiente\_token* no pertenece a una expresión aritmética de suma u otra expresión válida.

# Código 111: El token *siguiente\_token* no pertenece a una expresión válida.

# Código 112: El token *siguiente\_token* no pertenece a una expresión aritmética de multiplicación u otra expresión válida.

# Código 113: El token *siguiente\_token* no pertenece a una expresión válida.

# Código 114: El token *siguiente\_token* no es un identificador de variable o la llamada a una función

# Código 115: El token *siguiente\_token* no es un argumento de función válido porque no pertenece a una expresión válida

# Código 116: El token *siguiente\_token* no es un argumento de función válido porque no pertenece a una expresión válida

# Código 117: El token *siguiente\_token* no es un valor de retorno de función válido porque no pertenece a una expresión válida

# Código 118: El token *siguiente\_token* no es un tipo de variable válido

# Código 119: El token *siguiente\_token* no pertenece a una declaración válida de una función

# Código 120: El token *siguiente\_token* no es un tipo de valor de retorno de función válido

# Código 121: El token *siguiente\_token* no es una declaración válida de un argumento de una función

# Código 122: El token *siguiente\_token* no es una declaración válida de un argumento de una función

# Código 123: El token *siguiente\_token* no pertenece a una sentencia válida

# Código 124: El token *siguiente\_token* no pertenece a una sentencia válida

# Código 125: El token *siguiente\_token* no es una asignación válida

# Código 126: El token *siguiente\_token* no es un cuerpo válido para la condicional múltiple switch

# Código 127: El token *siguiente\_token* no pertenece a una sentencia válida

# Código 128: El token *siguiente\_token* no pertenece a una sentencia válida

# *siguiente\_token*: último token devuelto por el Analizador Léxico

# *cima\_pila:* último token apilado dado el algoritmo del Analizador Sintáctico Descendente por Tablas.

# ANALIZADOR SINTÁCTICO

# GRAMÁTICA DE ATRIBUTOS

|  |  |
| --- | --- |
|  | 1.1)TSG := crearTabla();desplG = 0; |
|  | 1.2)LiberarTabla(TSG);desplG = null; |
|  |  |
|  |  |
|  |  |
|  | 5.1)S.funcion := B.funcionS.switch := B.switch |
|  | 5.2)B.tipo := if (E.tipo == 'tipo\_logico')then S.tipoelse tipo\_error (201);B.tipoRet := S.tipoRet |
|  | 6.1)zona\_decl = true |
|  | 6.2)añadeTipo(id.pos,T.tipo);if(TSL == null)then añadeDespl(id.pos, desplG);desplG += T.tamaño;else añadeDespl(id.pos, desplL);desplL += T.tamaño);zona\_decl = false; |
|  | 6.3)B.tipo := if ( N.tipo == 'tipo\_vacio' orbuscarTipo(id.pos) == N.tipo)then 'tipo\_ok'else 'tipo\_error' (202) |
|  | 7.1)S.funcion = B.funcionS.switch := B.switch |
|  | 7.2)B.tipo := S.tipo;B.tipoRet := S.tipoRet; |
|  | 8.1)Z.funcion := B.funcion;Z.switch := true; |
|  | 8.2)B.tipo := if (E.tipo == 'tipo\_entero')then if (Z.tipo = 'tipo\_ok')then 'tipo\_ok';else 'tipo\_error'; (203)else 'tipo\_error'; (204)B.tipoRet := Z.tipoRet; |
|  | 9.1)decl\_impl = true; |
|  | 9.2)if ( buscarTipo(id.pos) == False)then añadeTipo(id.pos, 'tipo\_entero');if(TSL == null)then añadeDespl(id.pos, desplG);desplG += 2;else añadeDespl(id.pos, desplL);desplL += 2;decl\_impl = false; |
|  | 9.3)S.tipo := if (buscarTipo(id.pos) == 'tipo\_funcion')then if(buscarTipoParametros(id.pos) == S'.tipo)then 'tipo\_ok'else 'tipo\_error' (205)else if (buscarTipo(id.pos) == S'.tipo) then 'tipo\_ok'else 'tipo\_error' (206) |
|  | 10.1)S.tipo := if(E.tipo == 'tipo\_entero' or E.tipo == 'tipo\_cadena') then 'tipo\_ok'else if (E.tipo == 'tipo\_funcion')if(buscarTipoDevuelto(E.pos) == 'tipo\_entero' or buscarTipoDevuelto(E.pos) == 'tipo\_cadena')then 'tipo\_ok'else 'tipo\_error' (207)else 'tipo\_error' (207) |
|  | 11.1)decl\_impl = true; |
|  | 11.2)if (buscarTipo(id.pos) == False)then añadeTipo(id.pos, 'tipo\_entero');if(TSL == null)then añadeDespl(id.pos, desplG);desplG += 2;else añadeDespl(id.pos, desplL);desplL += 2;decl\_impl = false; |
|  | 11.3)S.tipo := if (buscarTipo(id.pos) == 'tipo\_entero'or buscarTipo(id.pos) == 'tipo\_cadena')then 'tipo\_ok'else 'tipo\_error' (208) |
|  | 12.1) S.tipo := if (X.tipo != 'tipo\_error')then if (S.funcion == true)then'tipo\_ok'else 'tipo\_error' (209)else 'tipo\_error' (210)S.tipoRet := X.tipo |
|  | 13.1) S.tipo := if (S.switch == true)then 'tipo\_ok'else 'tipo\_error' (211) |
|  | 14.1)S'.tipo := E.tipo |
|  | 15.1)S'.tipo := if (E.tipo == 'tipo\_entero')then U.tipoelse 'tipo\_error' (212) |
|  | 16.1)S'.tipo := L.tipo |
|  | 17.1)E.tipo := if ( (R.tipo == E'.tipo or E'.tipo == 'tipo\_vacio') and E'.tipo != 'tipo\_error')then R.tipoelse 'tipo\_error' (213) |
|  | 18.1)E'.tipo := 'tipo\_vacio' |
|  | 19.1)E'.tipo := if ( R.tipo == 'tipo\_logico' and ( E'1.tipo == 'tipo\_vacio' or E'1.tipo == 'tipo\_logico'))then R.tipoelse 'tipo\_error' (214) |
|  | 20.1)R.tipo := if (U.tipo == R'.tipo and R'.tipo != 'tipo\_error')then 'tipo\_logico'else if (R'.tipo == 'tipo\_vacio’ and and U.tipo != 'tipo\_error')then U.tipoelse 'tipo\_error' (215) |
|  | 21.1)R'.tipo := 'tipo\_vacio' |
|  | 22.1)R'.tipo := if ( U.tipo == R'1.tipo or R'1.tipo == 'tipo\_vacio') and ( R'1.tipo != 'tipo\_error') and U.tipo != 'tipo\_error')then U.tipoelse 'tipo\_error' (216) |
|  | 23.1)U.tipo := if ( (V.tipo == U'.tipo or U'.tipo == 'tipo\_vacio' ) and U'.tipo != 'tipo\_error' and V.tipo != 'tipo\_error')then V.tipoelse 'tipo\_error' (217) |
|  | 24.1)U'.tipo := 'tipo\_vacio' |
|  | 25.1)  U'.tipo := if ( (V.tipo == 'tipo\_entero'  and (U'1.tipo == 'tipo\_vacio'  or U'1.tipo == 'tipo\_entero'))  and U'1.tipo != 'tipo\_error'  and V.tipo != 'tipo\_error')  then V.tipo  else 'tipo\_error' (218) |
|  | 26.1) V.tipo := if ( (P.tipo == V'.tipo or V'.tipo ==  'tipo\_vacio')  and V'.tipo != 'tipo\_error'  and P.tipo != 'tipo\_error')    then P.tipo  else 'tipo\_error' (219) |
|  | 27.1) V'.tipo := 'tipo\_vacio' |
|  | 27.1) V'.tipo := if ((P.tipo == 'tipo\_entero' and ( V'1.tipo == 'tipo\_vacio' or V'1.tipo == 'tipo\_entero')) and V'1.tipo != 'tipo\_error')  then P.tipo    else 'tipo\_error' (220) |
|  | 29.1) decl\_impl = true |
|  | 29.2) if (buscarTipo(id.pos) == False)  then añadeTipo(id.pos, 'tipo\_entero')  if(TSL == null)  then añadeDespl(id.pos, desplG);  desplG += 2;  else añadeDespl(id.pos, desplL);  desplL += 2;  decl\_impl = false; |
|  | 29.3) P.tipo := if(buscarTipo(id.pos) == 'tipo\_funcion')  then if(buscarTipoParametros(id.pos) == P'.tipo )  then buscarValorDevuelto(id.pos)  else 'tipo\_error' (221)  else buscarTipo(id.pos) |
|  | 30.1) P.tipo := E.tipo |
|  | 31.1) P.tipo := 'tipo\_entero' |
|  | 32.1)P.tipo := 'tipo\_cadena' |
|  | 33.1)P.tipo := 'tipo\_logico' |
|  | 34.1)P.tipo := 'tipo\_logico' |
|  | 35.1)P'.tipo := 'tipo\_vacio' |
|  | 36.1)P'.tipo := L.tipo |
|  | 37.1)if(Q.tipo != 'tipo\_vacio' and Q.tipo != 'tipo\_error')then if(E.tipo != 'tipo\_error')then E.tipo x Q.tipoelse 'tipo\_error' (222)else if(E.tipo != 'tipo\_error' and Q.tipo != 'tipo\_error')then E.tipoelse 'tipo\_error' (223) |
|  | 38.1)L.tipo := 'tipo\_vacio' |
|  | 39.1)Q.tipo := if(Q1.tipo != 'tipo\_vacio' and Q1.tipo != 'tipo\_error')then if(E.tipo != 'tipo\_error')then E.tipo x Q1.tipoelse 'tipo\_error' (224)else if(E.tipo != 'tipo\_error' and Q1.tipo != 'tipo\_error')then E.tipoelse 'tipo\_error' (225) |
|  | 40.1)Q.tipo := 'tipo\_vacio' |
|  | 41.1)X.tipo := E.tipo |
|  | 42.1)X.tipo := 'tipo\_vacio' |
|  | 43.1)T.tipo := 'tipo\_entero';T.tamaño := 2; |
|  | 44.1)T.tipo := 'tipo\_logico';T.tamaño := 2; |
|  | 45.1)T.tipo := 'tipo\_cadena';T.tamaño := 128; |
| 1. {46.4} | 46.1)zona\_decl = true |
|  | 46.2)TSL := crearTabla();desplL = 0; |
|  | 46.3)zona\_decl = false;añadeTipo(id.pos, 'tipo\_funcion');añadeTipoParametros(D.tipo);añadeTipoDevuelto(H.tipo);zona\_decl = false |
|  | 46.4)if (C.tipoRet != H.tipo)then if(C.tipoRet == undefined and H.tipo != 'tipo\_vacio')then error()(226)liberarTabla(TSL);desplL = null; |
|  | 47.1)H.tipo := T.tipo |
|  | 48.1)H.tipo := 'tipo\_vacio' |
|  | 49.1)D.tipo := if(K.tipo != 'tipo\_vacio' and K.tipo != 'tipo\_error')then T.tipo x K.tipo;else if(K.tipo != 'tipo\_error')then T.tipo ;else 'tipo\_error'; (227)añadeTipo(id.pos, T.tipo);añadeDespl(id.pos,desplL);desplL += T.tamaño;if K.listaTT # Comprobar si existe esethen # atributo en Kfor sublista in K.listaTTañadeTipo(sublista[0], sublista[1]);añadeDespl(sublista[0], desplL);desplL += sublista[2]; |
|  | 50.1)D.tipo := 'tipo\_vacio' |
|  | 51.1)K.tipo := if(K1.tipo != 'tipo\_vacio' and K1.tipo != 'tipo\_error')then T.tipo x K1.tipo;else if(K1.tipo != 'tipo\_error')then T.tipo ;else 'tipo\_error'; (228)K.listaTT := [[id.pos, T.tipo, T.tamanho]]if K1.listaTT # Comprobar si existe esethen # atributo en K1for sublista in K1.listaTTK.listaTT += sublista |
|  | 52.1)K.tipo := 'tipo\_vacio' |
|  | 53.1)B.funcion := true |
|  | 53.2)C.tipo := if(B.tipo == C'.tipo == 'tipo\_ok')then 'tipo\_ok';else 'tipo\_error'; (229)C.tipoRet := if( B.tipoRet != undefined and B.tipoRet != 'tipo\_error' )then if (B.tipoRet == C'.tipoRet)then B.tipoRetelse if (C'.tipoRet == undefined)then B.tipoRetelse 'tipo\_error' (230)else if(C'.tipoRet != undefined and C'.tipoRet != 'tipo\_error')then C'.tipoRetelse if (C'.tipoRet != 'tipo\_error' or B.tipoRet != 'tipo\_error')then 'tipo\_error' (231)else undefined |
|  | 54.1)B.funcion := true |
|  | 54.2)C’.tipo := if(B.tipo == C'1.tipo == 'tipo\_ok')then 'tipo\_ok';else 'tipo\_error'; (232)C’.tipoRet := if( B.tipoRet != undefined and B.tipoRet != 'tipo\_error' )then if ( B.tipoRet == C'1.tipoRet)then B.tipoRetelse if (C'1.tipoRet == undefined)then B.tipoRetelse 'tipo\_error' (233)else if(C'1.tipoRet != undefined and C'1.tipoRet != 'tipo\_error')then C'1.tipoRetelse if (C'1.tipoRet != 'tipo\_error' or B.tipoRet != 'tipo\_error')then 'tipo\_error' (234)else undefined |
|  | 55.1)C'.tipo := 'tipo\_ok';C'.tipoRet == undefined; |
|  | 56.1)N.tipo := 'tipo\_vacio' |
|  | 57.1)N.tipo := E.tipo |
|  | 58.1)N.tipo := if (E.tipo == 'tipo\_entero')then E.tipoelse 'tipo\_error' (235) |
|  | 59.1)O.funcion = Z.funcion;O.switch := Z.switch |
|  | 59.2)Z'.funcion = Z.funcion;Z'.switch := Z.switch |
|  | 59.3)Z.tipo := if ( O.tipo == Z'.tipo == 'tipo\_ok' )then 'tipo\_ok'else 'tipo\_error' (236) |
|  | 60.1)O.funcion = Z.funcionO.switch := Z.switch |
|  | 60.2)Z.tipo := if ( O.tipo == 'tipo\_ok' )then 'tipo\_ok'else 'tipo\_error' (237) |
|  | 61.1)B.funcion = O.funcionB.switch := O.switch |
|  | 61.2)O'.funcion = O.funcionO'.switch := O.switch |
|  | 61.3)O.tipo := if ( B.tipo == O'.tipo == 'tipo\_ok' )then 'tipo\_ok'else 'tipo\_error' (238) |
|  | 62.1)O.tipo := 'tipo\_ok' |
|  | 63.1)Z.switch := Z'.switchZ.funcion = Z'.funcionZ'.tipo := if (Z.tipo == 'tipo\_ok')then 'tipo\_ok'else 'tipo\_error' (239) |
|  | 64.1)Z'.tipo := 'tipo\_ok' |

**ERRORES**

# Semánticos

# Código 200: aaaa