

Introducción

Se propone el lenguaje de programación de creación propia llamado KOD. Este lenguaje será un lenguaje orientado a objetos de los cuales procederemos a definir la estructura de los programas, sus identificadores, operadores, etc. La descripción del lenguaje se hará a continuación en prosa, sin el uso de gramáticas y servirá para el diseño de un traductor dentro del contexto del proyecto final.

Estructura del programa

La estructura de un programa en el lenguaje KOD está dada de la siguiente manera:

```
Nombre(){  
    declaraciones;  
    asignaciones;  
    instrucciones;  
}
```

Como podemos observar, los signos “{” y “}” representan el inicio y el fin de cualquier programa en el lenguaje, el cual inicia con el nombre de una determinada función y los paréntesis. Dentro de ambos signos de corchetes, van primero las declaraciones, luego las asignaciones y finalmente las instrucciones. Además, el símbolo “;” indica el fin de cada sentencia, del tipo que sea.

La función principal o “main” del lenguaje KOD se define con la palabra reservada **prin** y signos de apertura de paréntesis y cerradura de paréntesis, los cuales representan la estructura de una función que se detalla más adelante. Ejemplo:

```
prin{  
    declaraciones locales  
    asignaciones locales  
    instrucciones locales  
}
```

Identificadores

Un identificador maneja las diferentes variables dentro de un programa, por lo que tendríamos una serie de tipos de identificadores según lo deseado. Estos identificadores tienen que comenzar con un carácter alfabético y puede estar seguido de cualquier carácter alfanumérico o con guión, como por ejemplo “alo_123”. Existen identificadores para diferentes tipos de entidades, como:

Variables

Son entidades en el programa cuyo valor va cambiando durante la ejecución de este. Son definidas en el lenguaje KOD mediante sentencias donde se especifica primero el tipo de dato y luego un identificador, o lista de identificadores separados por una coma de la siguiente forma:

<tipo de dato> <identificador1>,<identificador2>

Antes de dar ejemplos de identificadores para variables, es necesario conocer los tipos de dato en el lenguaje KOD, los cuales son:

ente : Representa números enteros.

real : Representa números reales.

logic : Representa un valor lógico, verdadero o falso.

carac : Representa caracteres simbólicos

Siguiendo esta lista de tipos de datos, ejemplos de identificadores para variables serían:

Ejemplo 1:

real precio

Donde se está definiendo una variable de tipo real cuyo identificador es “precio”.

Ejemplo 2:

ente codigo0, numero1

En este ejemplo, se define una lista de variables enteras con identificadores “codigo0” y “numero1”

Constante

Una constante es una entidad cuyo valor no varía durante la ejecución de un programa. Se definen en el lenguaje KOD similar a como lo haría una variable, solo añadiendo el uso de la palabra reservada **konst** y un símbolo “=” con el cual le asignaremos un valor. La estructura de una definición de constantes sería tal:

konst <tipo de dato> <identificador> = <valor>

Ejemplo 1:

konst real pi = 3.14

Aquí, se define una constante de tipo real con identificador “pi” cuyo valor es 3.14

Vectores

Los llamados vectores son entidades que agrupan un número determinado de elementos de un mismo tipo de dato. Tienen la siguiente estructura en el lenguaje KOD:

<tipo de dato> <identificador> <tamaño del vector>

En la estructura se observa que es necesario proporcionar el tamaño de un vector y el proporcionarle un identificador, el cual cumple con la condición de ser un conjunto de caracteres alfanuméricos.

Ejemplo:

real boletas [10]

En este ejemplo, se está definiendo un vector que agrupa elementos de tipo real, cuyo tamaño es 10 y tiene como identificador a “boletas”. Se denota cómo el tamaño del vector se encuentra dentro de los símbolos “[“ y “]”.

Operadores

Definición de algunos operadores básicos que utiliza el lenguaje KOD.

+ : suma

- : sustracción

***** : multiplicación

/ : división

% : resto de la división

y : conjunción lógica
o : disyunción lógica
= : desigual
=/ : igual
< : menor que
</ : menor o igual
> : mayor que
>/ : mayor o igual
() : parámetros de método
[] : elemento de vector

Expresiones

Las expresiones son combinaciones de operadores y operandos (identificadores que denotan variables o constantes, funciones, etc). Ejemplo de expresiones en lenguaje KOD:

5 - numero1 + numero2

(id >/ 5) y (id2 + 10 < 30)

Instrucciones

El lenguaje KOD abarca diferentes tipos de instrucciones que se procede a ejemplificar.

Instrucciones declarativas

Como el compilador necesita ser informado de cada identificador existente para su uso en el programa, es necesaria una declaración con la que presentamos este identificador. Esta declaración hace referencia a una determinada entidad de cierto tipo de dato. Tiene la siguiente estructura:

<tipo de dato> <identificador1>,<identificador2>

En la estructura se observa el tipo de dato y una lista de identificadores que van separados por una coma.

Ejemplos:

ente numero1,numero2

real identificacion

logico condicion,estado,resultado

Instrucciones de asignación

Las instrucciones de asignación permiten almacenar en una variable el resultado de alguna expresión. Para definir la estructura de una asignación, el identificador de una variable debe aparecer al lado izquierdo de la instrucción la cual está separada por un carácter “=”. A la derecha de este carácter, debe colocarse una expresión cualquiera que arroje un valor. La estructura es la siguiente:

<identificador> = <valor>

Ejemplos:

variable1 = ide + 24

numero = 25/codigo * 12

Instrucciones selectivas

Para las instrucciones selectivas, se reservan palabras como **ki** y **kino** para instrucciones selectivas dobles y palabras como **kwitch** y **kase** para instrucciones selectivas múltiples.

La estructura para una instrucción selectiva doble es la siguiente:

```
ki(<expresion_logica>) {  
    sentencias  
    ...  
    } kino {  
        sentencias  
        ...  
    }
```

La estructura para una instrucción múltiple es la siguiente:

```
kwitch(<expresion_logica>){
```

```

kase 1:
    sentencias
kase 2:
    sentencias
...
kase n:
    sentencias
}

```

Se observa el uso de la palabra reservada **kwitch** seguido de una expresión, la apertura de llaves y el uso de **kase** seguido de un número entero. Después de los “:” colocados, se procede a escribir las sentencias correspondientes a esa selección. Una vez terminada la escritura de sentencias, se continúa con la siguiente selección.

Ejemplos:

```

ki (K_a </ K_b) {
    ...
}
kino {
    ...
}

```

```

kwitch ( opcion ) {
    kase 1:
        ...
    kase 2:
        ...
}

```

Instrucciones repetitivas

Para las instrucciones repetitivas se reservan palabras como **kor**, **kientras**

Se procede a detallar la estructura para cada posible instrucción repetitiva.

Para “**kor**”.

```
kor(entero <identificador1> = <expresión_entera> ; <expresión  
lógica> ; <instrucción de incremento>) {  
  
    sentencias  
  
}
```

En esta estructura para el **kor**, vemos que dentro de los paréntesis se encuentra una inicialización, donde tenemos una entidad entera con cierto identificador (“identificador1”) a la cual le asignamos el valor de una expresión también entera. Seguido por el punto y coma, colocamos una expresión lógica (de valor verdadero o falso) y por último, seguido de otro punto y coma, una instrucción de incremento para la entidad entera definida al inicio.

Para “**kientras**”

```
kientras(<expresion logica>){  
  
    sentencias  
  
}
```

En esta estructura, para el uso de **kientras**, vemos dentro de los paréntesis una expresión lógica la cual puede tener un valor de falso o verdadero y la apertura de corchetes donde colocaremos las sentencias que se ejecutarán en caso la expresión lógica sea verdadera.

Instrucciones de entrada y salida

Para instrucciones de entrada y salida se reservan palabras como **ingresar** e **imprimir**. Se estructuran usando lo siguiente:

```
ingresar(<identificador>)  
imprimir(<valor>)
```

Subprogramas

Funciones

Su definición posee la siguiente estructura:

```
<tipo de dato> <identificador> (<parámetros>){
```

```

sentencias
...
devuelve <expresión>;
}

```

Las funciones escritas en KOD poseen los parámetros delimitados por paréntesis, los cuales se encontrarán definidos dentro por comas. Se especifica al inicio el tipo de dato que se utilizará, el identificador y las sentencias dentro del corchete. Por tratarse de una función, es necesario el retorno de un valor. En este caso, usando la palabra reservada **devuelve**, somos capaces de devolver una expresión del mismo tipo que el tipo de dato.

Ejemplo:

```

real sumarNumeros (real a ,real b) {
    devuelve a + b ;
}

```

La llamada de una función se realiza con la palabra reservada **fun** de la siguiente forma:

```

fun <identificadorDeFuncion> (<argumentos>)

```

Procedimientos

Poseen la siguiente estructura en su definición

```

sinretorno <identificador> (<parametros>){
    sentencias
    ...
}

```

Los procedimientos escritos en KOD poseen los parámetros delimitados por paréntesis, antepuestos por su identificador y la palabra reservada **sinretorno**, la cual denota que se trata efectivamente de un procedimiento. Dentro de los corchetes, se colocan las sentencias o instrucciones a realizar. No es necesario en esta ocasión, hacer la devolución de algún valor.

Ejemplo:


```

sinretorno impresion(ente a){
    imprimir( a );
}

```

La llamada de un procedimiento se realiza con la palabra reservada **pro** de la siguiente forma:

pro <identificadorDeProcedimiento> (<argumentos>)

Clases y objetos

Las clases son tipos abstractos de datos que el programador crea según conveniencia. En el lenguaje KOD, las clases se definen usando la palabra reservada **klase** y tienen la siguiente estructura:

```

klase <identificador>{

    declaraciones de atributos
    <tipo de dato> metodo1 (<parametros>) {

        sentencias del método

    }

    sinretorno metodo2 (<parametros>) {

        sentencias del método

    }

    ...

}

```

Es posible observar la estructura de una clase, la cual empieza con la palabra reservada **klase**, seguido de un identificador del lenguaje KOD. Dentro de los corchetes se encuentra la declaración de los atributos de la clase y sus respectivos métodos, los cuales pueden ser funciones o procedimientos con la capacidad de recibir parámetros mediante el uso de los paréntesis.

Ejemplo:

```

klase Automovil {
    //Atributos
    carac marca[50];
    real peso;
    ente numAsientos;

    //Métodos
    ente registrarCarro(real a){
        sentencias
        ..
    }
    sinretorno cambiarPeso(){
        sentencias
        ..
    }
}

```

Es importante reconocer también que dentro de un programa, la creación de objetos (instancias de una clase) tienen una estructura definida. Además de su creación, su manipulación también es objeto de descripción para el uso de sus respectivos métodos.

Estructura de la creación de objetos de una clase

<Nombre de la clase> <Identificador1>,<Identificador2>

La creación de un objeto inicia con colocar el nombre de la clase del objeto deseado, seguido de un identificador o lista de identificadores separados por comas.

Ejemplo:

Complejo c1,c2;

Lo que hace esta sentencia es crear dos objetos con identificador c1 y c2 los cuales son de la clase “Complejo”.

Al crear objetos de una clase, uno es capaz de poder acceder a sus métodos correspondientes. Este acceso en el lenguaje KOD está dado por el símbolo “.”

La estructura del uso de método de una clase es la siguiente:

<identificador del objeto>.<identificador del método>(<argumentos>)

Ejemplo:

c1:SumarDivisores();

Aquí, el objeto de clase “Complejo” con identificador “c1”, está haciendo uso de su método con identificador “SumarDivisores” mediante la unión de ambos usando un carácter “.”.

Por último, uno es capaz de acceder a los atributos de un objeto de cierta clase también mediante el uso del símbolo “.”. Un acceso tiene la siguiente estructura:

<identificador del objeto>.<atributo>

Ejemplo:

c1:parteEntera = 14;

En este ejemplo, estamos accediendo al atributo “parteEntera” del objeto c1 y le estamos asignando el valor numérico de 14. Los atributos, según el tipo que sean, pueden funcionar como valores los cuales son posibles de colocar dentro de expresiones lógicas.

Gramática del lenguaje KOD

Para el analizador sintáctico del lenguaje KOD, se decidió utilizar un parser de tipo LL(1), para el cual necesitamos el análisis de la gramática correspondiente.

Gramática del lenguaje

Una gramática está definida por los siguientes elementos:

$$G=(V_n, V_t, P, S)$$

Donde V_n es el conjunto de valores no terminales, V_t el conjunto de valores terminales, P las reglas de producción y S es el símbolo inicial.

Dentro de la descripción de la gramática del lenguaje KOD, proporcionaremos una gramática hasta un nivel sintáctico, excluyendo por el momento el nivel léxico como requerimiento del docente del curso.

$V_n = \{ S, SA, SB, FCP, CL, ATBS, MTS, MT, CFUN, CPROC, PMTS, PMT, RET, INST, INS, DCL, TDCL, DCLA, EOVS, IDS, ASG, IN, OUT, LFUN, LPROC, IF, WH, CTE, FOR, ARGS, VL, EXP, EXPS, EXPA, EXPB, EXPC, EXPD, EXPE, EXPF, EXPG, EXPH, VLEXP, CR, NUM, INT, TD, ID, IDOB \}$

$V_t = \{ klase, id, \{, \}, conretorno, (,), sinretorno, ,, devuelve, :, prin, [,], =, ingresar, imprimir, fun, pro, ki, kino, kientras, konst, kor, ente, +, -, *, / \%, >, !, <, o, y, num, verdadero, falso, c, 0, real, logic, carac, :, \$ \}$

$P = \{$

$S \rightarrow SA\ SB$
 $SA \rightarrow FCP\ SA$
 $SA \rightarrow "$
 $FCP \rightarrow CL$
 $FCP \rightarrow CFUN$
 $FCP \rightarrow CPROC$
 $CL \rightarrow klase\ id\ \{ ATBS\ MTS \}$
 $ATBS \rightarrow DCL\ ATBS$
 $ATBS \rightarrow "$
 $MTS \rightarrow MT\ MTS$
 $MTS \rightarrow "$
 $MT \rightarrow CFUN$
 $MT \rightarrow CPROC$

CFUN -> **conretorno** TD **id** (PMTS) { INST RET }

CPROC-> **sinretorno** **id** (PMTS) { INST }

PMTS -> TD **id** PMT

PMTS -> "

PMT -> , TD **id** PMTS

PMT -> "

RET -> **devuelve** VL ;

SB -> **prin** { INST }

INST -> INST INST

INST -> "

INS -> DCL

DCLV -> TD **id** TDCL ;

TDCL -> DCLA

TDCL -> IDS

DCLA -> [EOVS]

EOVS -> INT

EOVS -> "

IDS -> , **id** IDS

IDS -> "

INS -> ASG ;

INS -> IN ;

INS -> OUT ;

INS -> LPROC ;

INS -> IF

INS -> WH

INS -> CTE ;

INS -> FOR

ASG -> ID = VL

IN -> **ingresar** (ID)

OUT -> **imprimir** (VL)

LFUN -> **fun** ID (ARGS)

LPROC-> **pro** ID (ARGS)

IF -> **ki** (VL) { INST } **kino** { INST }
WH -> **kientras** (EXP) { INST }
CTE -> **konst** TD **id**
FOR -> **kor** (ente ID = INT ; EXP ; ASG) { INST }
ARGS -> VL ARGS
ARGS -> , VL ARGS
ARGS -> "
VL -> CR
VL -> EXP
VL -> LFUN
EXP -> EXPA EXPS
EXPS -> + EXPA EXPS1
EXPS -> - EXPA EXPS
EXPS -> "
EXPA -> EXPC EXPB
EXPB -> * EXPC EXPB1
EXPB -> / EXPC EXPB1
EXPB -> % EXPC EXPB
EXPB -> "
EXPC -> EXPE EXPD
EXPD -> > ! EXPE EXPD
EXPD -> < ! EXPE EXPD
EXPD -> = = EXPE EXPD
EXPD -> ! = EXPE EXPD
EXPD -> "
EXPE -> EXPG EXPF
EXPF -> o EXPG EXPF
EXPF -> "
EXPG -> VLEXP EXPH
EXPH -> y VLEXP EXPH
EXPH -> "
VLEXP -> (EXP)
VLEXP -> num
VLEXP -> verdadero

VLEXP -> falso

VLEXP -> id

CR -> c

NUM -> num

INT -> 0

TD -> ente

TD -> real

TD -> logic

TD -> carac

ID -> id IDOB

IDOB -> : id

IDOB -> "

}

Por último, el símbolo inicial, en este caso

S = <programa>

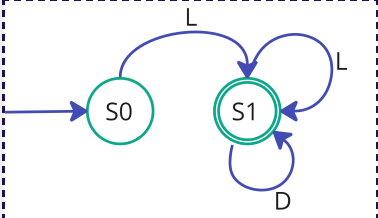
Analizador Léxico del lenguaje KOD

Especificación del analizador léxico para el proyecto de curso

1. Palabras reservadas

Lexemas	Componente léxico	Descripción
sinretorno	palabrasReservadas	Conjunto de palabras reservadas según el lenguaje de programación propuesto (KOD).
fun		
pro		
ente		
real		
logic		
carac		
konst		
ki		
kino		
kwitch		
kase		
kor		
kientras		
ingresar		
imprimir		
devuelve		
klase		
prin		
verdadero		
falso		

2. Identificadores

Lexemas	Componente léxico	Descripción									
<p><u>Expresión regular</u> $L = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z\}$</p> <p>$D = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, _ \}$</p> <p>$L(L/D)^*$</p> <p><u>Autómata finito</u> Diagrama de transición:</p>  <p>Cuadro de transición:</p> <table border="1"> <thead> <tr> <th></th><th>L</th><th>D</th></tr> </thead> <tbody> <tr> <th>S0</th><td>S1</td><td></td></tr> <tr> <th>S1</th><td>S1</td><td>S1</td></tr> </tbody> </table>		L	D	S0	S1		S1	S1	S1	identificadores	Lexemas que sirven para identificar variables, funciones, procedimientos, etc. en el lenguaje de programación KOD.
	L	D									
S0	S1										
S1	S1	S1									

3. Constantes

Lexemas	Componente Léxico	Descripción
<p><u>Expresión regular</u> $D = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$</p> <p>$DD^*$</p> <p><u>Autómata finito</u> Diagrama de transición:</p>	numerosEnteros	Constantes numéricas de tipo entero.

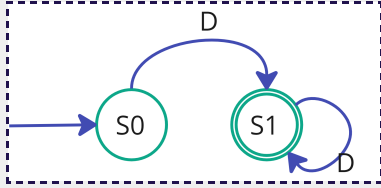


Tabla de transición:

	D
S0	S1
S1	S1

Expresión regular

$D = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

$P = \{.\}$

DD*PDD*

Autómata finito

Diagrama de transición:

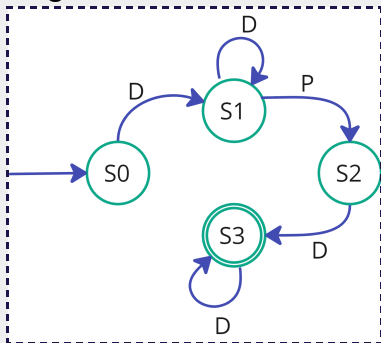


Tabla de transición:

	L	D
S0	S1	
S1	S1	S2
S2	S3	
S3	S3	

numerosReales

Constantes numéricas de tipo real

Expresión regular

$L = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, _ \}$

$D = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

$E = \{ \}$

$S = \{+, -, *, /, \%, \$, \#, !, \cdot, \wedge, \text{?}, =, >, <, [,], \{, \}, \cdot, \cdot, \cdot, \cdot, \& \}$

$(E^*|D^*|L^*|S^*)^*$

Autómata finito

Diagrama de transición:

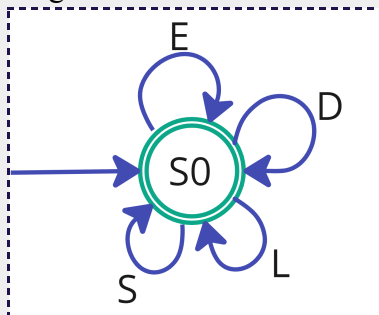


Tabla de transición:

	E	D	L	S
S0	S0	S0	S0	S0

cadenas

Una cadena, conjunto de varios caracteres que están delimitados por comillas doble. Estas comillas no aparecen en el lexema, solo es la delimitación.

Expresión regular

$L = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, _ \}$

$D = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

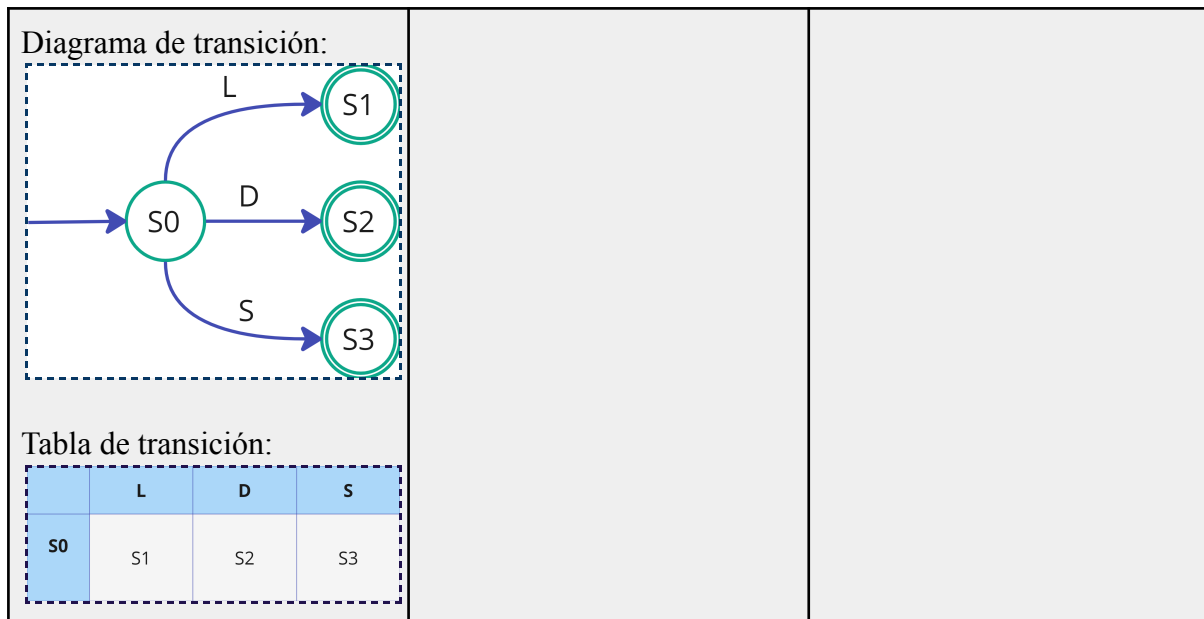
$S = \{+, -, *, /, \%, \$, \#, !, \cdot, \wedge, \text{?}, =, >, <, [,], \{, \}, \cdot, \cdot, \cdot, \cdot, \& \}$

$L|D|S$

Autómata finito

caracteres

Una constante de tipo carácter la cual está delimitada por comillas simples. De la misma forma que las cadenas, esta delimitación no se presenta en el lexema.



4. Símbolos

Lexemas	Componente Léxico	Descripción
+	símbolos	Operadores aritméticos
-		
*		
/		
y		Operadores lógicos
o		
=		Operadores de igualdad o desigualdad
=/		
<		
</		
>		
>/		
(Delimitadores

)		
[
]		
,		Concatenación de identificadores
;		Fin de línea de instrucción
{		Delimitadores de funciones, métodos y clases
}		
'		Delimitadores para cadenas y caracteres
“		
:		Llamada de atributos o métodos de objetos

Analizador Sintáctico del lenguaje KOD

Tabla de análisis sintáctico

Vn	Primeros	Siguientes
S	{λ, prin, klase, conretorno, sinretorno}	{\$}
SA	{λ, klase, conretorno, sinretorno}	{prin}
FCP	{klase, conretorno, sinretorno}	{prin(), klase, conretorno, sinretorno}
CL	{klase}	{prin, klase, conretorno, sinretorno}
ATBS	{λ, ente, real, logic, carac}	{prin, klase, conretorno, sinretorno, }}
MTS	{λ, conretorno, sinretorno}	{}}
MT	{conretorno, sinretorno}	{}, conretorno, sinretorno}
CFUN	{conretorno}	{prin, klase, conretorno, sinretorno, }}
CPROC	{sinretorno}	{prin().klase,conretorno,sinretorno, }}
PMTS	{λ, ente, real, logic, carac}	{}
PMT	{,}	{}
RET	{devuelve}	{}
SB	{prin}	{\$}
INST	{λ, ingresar, imprimir, pro, ki, kientras, konst, kor, ente, real, logic, carac, id}	{devuelve, }}
INS	{ingresar, imprimir, pro, ki, kientras, konst, kor, ente, real, logic, carac, id}	{devuelve, }, ningresar, imprimir, pro, ki, kientras, konst, kor, ente, real, logic, carac, id}
DCL	{ente, real, logic, carac}	{prin, klase, conretorno, sinretorno, }, ente, real, logic, carac, devuelve, ingresar, imprimir, pro, ki, kientras, konst, kor, id}
TDCL	{[, ,, λ}	{}
DCLA	{[]}	{}
EOV	{λ, 0}	{}
IDS	{,, λ}	{}
ASG	{id}	{:,)}
IN	{ingresar}	{}
OUT	{imprimir}	{}

Vn	klase	id	{	}
S	S -> SA SB			
SA	SA -> FCP SA			
FCP	FCP -> CL			
CL	CL -> klase id { ATBS MTS }			
ATBS	ATBS -> λ			ATBS -> λ
MTS				MTS -> λ
MT				
CFUN				
CPROC				
PMTS				
PMT				
RET				
SB				
INST		INST -> INS INST		INST -> λ
INS		INS -> ASG ;		
DCL				
TDCL				
DCLA				
EOV				
IDS				
ASG		ASG -> ID = VL		
IN				
OUT				
LFUN				
LPROC				
IF				
WH				
CTE				
FOR				
ARGS		ARGS -> VL ARGS		
VL		VL -> EXP		
EXP		EXP -> EXPA EXPS		
EXPS		EXPS -> λ		
EXPA		EXPA -> EXPC EXPB		
EXPB		EXPB -> λ		
EXPC		EXPC -> EXPE EXPD		
EXPD		EXPD -> λ		
EXPE		EXPE -> EXPG EXPF		
EXPF		EXPF -> λ		
EXPG		EXPG -> VLEXP EXPH		
EXPH		EXPH -> λ		
VLEXP		VLEXP -> id		
CR				
NUM				
INT				
TD				
ID		ID -> id IDOB		
IDOB				

Vn	conretorno
S	S -> SA SB
SA	SA -> FCP SA
FCP	FCP -> CFUN
CL	
ATBS	ATBS -> λ
MTS	MTS -> MT MTS
MT	MT -> CFUN
CFUN	CFUN -> conretorno TD id (PMTS) { INST RET }
CPROC	
PMTS	
PMT	
RET	
SB	
INST	
INS	
DCL	
TDCL	
DCLA	
EOV	
IDS	
ASG	
IN	
OUT	
LFUN	
LPROC	
IF	
WH	
CTE	
FOR	
ARGS	
VL	
EXP	
EXPS	
EXPA	
EXPB	
EXPC	
EXPD	
EXPE	
EXPF	
EXPG	
EXPH	
VLEXP	
CR	
NUM	
INT	
TD	
ID	
IDOB	

Vn	()
S		
SA		
FCP		
CL		
ATBS		
MTS		
MT		
CFUN		
CPROC		
PMTS		PMTS -> λ
PMT		
RET		
SB		
INST		
INS		
DCL		
TDCL		
DCLA		
EOV		
IDS		
ASG		
IN		
OUT		
LFUN		
LPROC		
IF		
WH		
CTE		
FOR		
ARGS	ARGS -> VL ARGS	ARGS -> λ
VL	VL -> EXP	
EXP	EXP -> EXPA EXPS	
EXPS	EXPS -> λ	EXPS -> λ
EXPA	EXPA -> EXPC EXPB	
EXPB	EXPB -> λ	EXPB -> λ
EXPC	EXPC -> EXPE EXPD	
EXPD	EXPD -> λ	EXPD -> λ
EXPE	EXPE -> EXPG EXPF	
EXPF	EXPF -> λ	EXPF -> λ
EXPG	EXPG -> VLEXP EXPH	
EXPH	EXPH -> λ	EXPH -> λ
VLEXP	VLEXP -> (EXP)	
CR		
NUM		
INT		
TD		
ID		
IDOB	IDOB -> λ	IDOB -> λ

Vn	sinretorno	,
S	S -> SA SB	
SA	SA -> FCP SA	
FCP	FCP -> CPROC	
CL		
ATBS	ATBS -> λ	
MTS	MTS -> MT MTS	
MT	MT -> CPROC	
CFUN		
CPROC	CPROC -> sinretorno id (PMTS) { INST }	
PMTS		
PMT		PMT -> , TD id PMTS
RET		
SB		
INST		
INS		
DCL		
TDCL		TDCL -> IDS
DCLA		
EOV		
IDS		IDS -> , id IDS
ASG		
IN		
OUT		
LFUN		
LPROC		
IF		
WH		
CTE		
FOR		
ARGS		ARGS -> , VL ARGS
VL		
EXP		
EXPS		EXPS -> λ
EXPA		
EXPB		EXPB -> λ
EXPC		
EXPD		EXPD -> λ
EXPE		
EXPF		EXPF -> λ
EXPG		
EXPH		EXPH -> λ
VLEXP		
CR		
NUM		
INT		
TD		
ID		
IDOB		

Vn	devuelve	;	prin
S			S -> SA SB
SA			SA -> λ
FCP			
CL			
ATBS			ATBS -> λ
MTS			
MT			
CFUN			
CPROC			
PMTS			
PMT			
RET	RET -> devuelve VL ;		
SB			SB -> prin { INST }
INST	INST -> λ		
INS			
DCL			
TDCL		TDCL -> IDS	
DCLA			
EOV			
IDS		IDS -> λ	
ASG			
IN			
OUT			
LFUN			
LPROC			
IF			
WH			
CTE			
FOR			
ARGS			
VL			
EXP			
EXPS		EXPS -> λ	
EXPA			
EXPB		EXPB -> λ	
EXPC			
EXPD		EXPD -> λ	
EXPE			
EXPF		EXPF -> λ	
EXPG			
EXPH		EXPH -> λ	
VLEXP			
CR			
NUM			
INT			
TD			
ID			
IDOB			

Vn	[]	=
S			
SA			
FCP			
CL			
ATBS			
MTS			
MT			
CFUN			
CPROC			
PMTS			
PMT			
RET			
SB			
INST			
INS			
DCL			
TDCL	TDCL -> DCLA		
DCLA	DCLA -> [EOVS]		
EOVS		EOVS -> λ	
IDS			
ASG			
IN			
OUT			
LFUN			
LPROC			
IF			
WH			
CTE			
FOR			
ARGS			
VL			
EXP			
EXPS			
EXPA			
EXPB			
EXPC			
EXPD			EXPD -> == EXPE EXPD
EXPE			
EXPF			EXPF -> λ
EXPG			
EXPH			EXPH -> λ
VLEXP			
CR			
NUM			
INT			
TD			
ID			
IDOB			IDOB -> λ

Vn	ingresar	imprimir	fun
S			
SA			
FCP			
CL			
ATBS			
MTS			
MT			
CFUN			
CPROC			
PMTS			
PMT			
RET			
SB			
INST	INST -> INS INST	INST -> INS INST	
INS	INS -> IN ;	INS -> OUT ;	
DCL			
TDCL			
DCLA			
EOV			
IDS			
ASG			
IN	IN -> ingresar (ID)		
OUT		OUT -> imprimir (VL)	
LFUN			LFUN -> fun ID (ARGS)
LPROC			
IF			
WH			
CTE			
FOR			
ARGS			ARGS -> VL ARGS
VL			VL -> LFUN
EXP			
EXPS			EXPS -> λ
EXPA			
EXPB			EXPB -> λ
EXPC			
EXPD			EXPD -> λ
EXPE			
EXPF			EXPF -> λ
EXPG			
EXPH			EXPH -> λ
VLEXP			
CR			
NUM			
INT			
TD			
ID			
IDOB			

Vn	pro	ki	kino
S			
SA			
FCP			
CL			
ATBS			
MTS			
MT			
CFUN			
CPROC			
PMTS			
PMT			
RET			
SB			
INST	INST -> INS INST	INST -> INS INST	
INS	INS -> LPROC ;	INS -> IF	
DCL			
TDCL			
DCLA			
EOV			
IDS			
ASG			
IN			
OUT			
LFUN			
LPROC	LPROC -> pro ID (ARGS)		
IF		IF -> ki (VL) { INST } kino { INST }	
WH			
CTE			
FOR			
ARGS			
VL			
EXP			
EXPS			
EXPA			
EXPB			
EXPC			
EXPD			
EXPE			
EXPF			
EXPG			
EXPH			
VLEXP			
CR			
NUM			
INT			
TD			
ID			
IDOB			

Vn	kientras	konst
S		
SA		
FCP		
CL		
ATBS		
MTS		
MT		
CFUN		
CPROC		
PMTS		
PMT		
RET		
SB		
INST	INST -> INS INST	INST -> INS INST
INS	INS -> WH	INS -> CTE ;
DCL		
TDCL		
DCLA		
EOV		
IDS		
ASG		
IN		
OUT		
LFUN		
LPROC		
IF		
WH	WH -> kientras (EXP) { INST }	
CTE		CTE -> konst TD id
FOR		
ARGS		
VL		
EXP		
EXPS		
EXPA		
EXPB		
EXPC		
EXPD		
EXPE		
EXPF		
EXPG		
EXPH		
VLEXP		
CR		
NUM		
INT		
TD		
ID		
IDOB		

Vn	kor	ente
S		
SA		
FCP		
CL		
ATBS		ATBS -> DCL ATBS
MTS		
MT		
CFUN		
CPROC		
PMTS		PMTS -> TD id PMT
PMT		
RET		
SB		
INST	INST -> INS INST	INST -> INS INST
INS	INS -> FOR	INS -> DCL
DCL		DCL -> TD id TDCL ;
TDCL		
DCLA		
EOV		
IDS		
ASG		
IN		
OUT		
LFUN		
LPROC		
IF		
WH		
CTE		
FOR	FOR -> kor (ente ID = INT ; EXP ; ASG) { INST }	
ARGS		
VL		
EXP		
EXPS		
EXPA		
EXPB		
EXPC		
EXPD		
EXPE		
EXPF		
EXPG		
EXPH		
VLEXP		
CR		
NUM		
INT		
TD		TD -> ente
ID		
IDOB		

Vn	+	-
S		
SA		
FCP		
CL		
ATBS		
MTS		
MT		
CFUN		
CPROC		
PMTS		
PMT		
RET		
SB		
INST		
INS		
DCL		
TDCL		
DCLA		
EOV		
IDS		
ASG		
IN		
OUT		
LFUN		
LPROC		
IF		
WH		
CTE		
FOR		
ARGS		
VL		
EXP		
EXPS	EXPS -> + EXPA EXPS	EXPS -> - EXPA EXPS
EXPA		
EXPB	EXPB -> λ	EXPB -> λ
EXPC		
EXPD	EXPD -> λ	EXPD -> λ
EXPE		
EXPF	EXPF -> λ	EXPF -> λ
EXPG		
EXPH	EXPH -> λ	EXPH -> λ
VLEXP		
CR		
NUM		
INT		
TD		
ID		
IDOB		

Vn	*	/
S		
SA		
FCP		
CL		
ATBS		
MTS		
MT		
CFUN		
CPROC		
PMTS		
PMT		
RET		
SB		
INST		
INS		
DCL		
TDCL		
DCLA		
EOV		
IDS		
ASG		
IN		
OUT		
LFUN		
LPROC		
IF		
WH		
CTE		
FOR		
ARGS		
VL		
EXP		
EXPS		
EXPA		
EXPB	EXPB -> * EXPC EXPB	EXPB -> / EXPC EXPB
EXPC		
EXPD	EXPD -> λ	EXPD -> λ
EXPE		
EXPF	EXPF -> λ	EXPF -> λ
EXPG		
EXPH	EXPH -> λ	EXPH -> λ
VLEXP		
CR		
NUM		
INT		
TD		
ID		
IDOB		

Vn	%	>
S		
SA		
FCP		
CL		
ATBS		
MTS		
MT		
CFUN		
CPROC		
PMTS		
PMT		
RET		
SB		
INST		
INS		
DCL		
TDCL		
DCLA		
EOV		
IDS		
ASG		
IN		
OUT		
LFUN		
LPROC		
IF		
WH		
CTE		
FOR		
ARGS		
VL		
EXP		
EXPS		
EXPA		
EXPB	EXPB -> % EXPC EXPB	
EXPC		
EXPD	EXPD -> λ	EXPD -> > ! EXPE EXPD
EXPE		
EXPF	EXPF -> λ	EXPF -> λ
EXPG		
EXPH	EXPH -> λ	EXPH -> λ
VLEXP		
CR		
NUM		
INT		
TD		
ID		
IDOB		

Vn	!	<
S		
SA		
FCP		
CL		
ATBS		
MTS		
MT		
CFUN		
CPROC		
PMTS		
PMT		
RET		
SB		
INST		
INS		
DCL		
TDCL		
DCLA		
EOV		
IDS		
ASG		
IN		
OUT		
LFUN		
LPROC		
IF		
WH		
CTE		
FOR		
ARGS		
VL		
EXP		
EXPS		
EXPA		
EXPB		
EXPC		
EXPD	EXPD -> != EXPE EXPD	EXPD -> < ! EXPE EXPD
EXPE		
EXPF	EXPF -> λ	EXPF -> λ
EXPG		
EXPH	EXPH -> λ	EXPH -> λ
VLEXP		
CR		
NUM		
INT		
TD		
ID		
IDOB		

Vn	o	y
S		
SA		
FCP		
CL		
ATBS		
MTS		
MT		
CFUN		
CPROC		
PMTS		
PMT		
RET		
SB		
INST		
INS		
DCL		
TDCL		
DCLA		
EOV		
IDS		
ASG		
IN		
OUT		
LFUN		
LPROC		
IF		
WH		
CTE		
FOR		
ARGS		
VL		
EXP		
EXPS		
EXPA		
EXPB		
EXPC		
EXPD		
EXPE		
EXPF	EXPF -> o EXPG EXPF	
EXPG		
EXPH	EXPH -> λ	EXPH -> y VLEXP EXPH
VLEXP		
CR		
NUM		
INT		
TD		
ID		
IDOB		

Vn	num	VERDADERO
S		
SA		
FCP		
CL		
ATBS		
MTS		
MT		
CFUN		
CPROC		
PMTS		
PMT		
RET		
SB		
INST		
INS		
DCL		
TDCL		
DCLA		
EOV		
IDS		
ASG		
IN		
OUT		
LFUN		
LPROC		
IF		
WH		
CTE		
FOR		
ARGS	ARGS -> VL ARGS	ARGS -> VL ARGS
VL	VL -> EXP	VL -> EXP
EXP	EXP -> EXPA EXPS	EXP -> EXPA EXPS
EXPS	EXPS -> λ	EXPS -> λ
EXPA	EXPA -> EXPC EXPB	EXPA -> EXPC EXPB
EXPB	EXPB -> λ	EXPB -> λ
EXPC	EXPC -> EXPE EXPD	EXPC -> EXPE EXPD
EXPD	EXPD -> λ	EXPD -> λ
EXPE	EXPE -> EXPG EXPF	EXPE -> EXPG EXPF
EXPF	EXPF -> λ	EXPF -> λ
EXPG	EXPG -> VLEXP EXPH	EXPG -> VLEXP EXPH
EXPH	EXPH -> λ	EXPH -> λ
VLEXP	VLEXP -> num	VLEXP -> verdadero
CR		
NUM		
INT		
TD		
ID		
IDOB		

Vn	FALSO	c	9
S			
SA			
FCP			
CL			
ATBS			
MTS			
MT			
CFUN			
CPROC			
PMTS			
PMT			
RET			
SB			
INST			
INS			
DCL			
TDCL			
DCLA			
EOV			
IDS			
ASG			
IN			
OUT			
LFUN			
LPROC			
IF			
WH			
CTE			
FOR			
ARGS	ARGS -> VL ARGS	ARGS -> VL ARGS	
VL	VL -> EXP	VL -> CR	
EXP	EXP -> EXPA EXPS		
EXPS	EXPS -> λ	EXPS -> λ	
EXPA	EXPA -> EXPC EXPB		
EXPB	EXPB -> λ	EXPB -> λ	
EXPC	EXPC -> EXPE EXPD		
EXPD	EXPD -> λ	EXPD -> λ	
EXPE	EXPE -> EXPG EXPF		
EXPF	EXPF -> λ	EXPF -> λ	
EXPG	EXPG -> VLEXP EXPH		
EXPH	EXPH -> λ	EXPH -> λ	
VLEXP	VLEXP -> falso		
CR		CR -> c	
NUM			NUM -> 9
INT			
TD			
ID			
IDOB			

Vn	0	real	logic
S			
SA			
FCP			
CL			
ATBS		ATBS -> DCL ATBS	ATBS -> DCL ATBS
MTS			
MT			
CFUN			
CPROC			
PMTS		PMTS -> TD id PMT	PMTS -> TD id PMT
PMT			
RET			
SB			
INST		INST -> INS INST	INST -> INS INST
INS		INS -> DCL	INS -> DCL
DCL		DCL -> TD id TDCL ;	DCL -> TD id TDCL ;
TDCL			
DCLA			
EOV	EOV -> INT		
IDS			
ASG			
IN			
OUT			
LFUN			
LPROC			
IF			
WH			
CTE			
FOR			
ARGS			
VL			
EXP			
EXPS			
EXPA			
EXPB			
EXPC			
EXPD			
EXPE			
EXPF			
EXPG			
EXPH			
VLEXP			
CR			
NUM			
INT	INT -> 0		
TD		TD -> real	TD -> logic
ID			
IDOB			

Vn	carac	:	\$
S			S -> SA SB
SA			
FCP			
CL			
ATBS	ATBS -> DCL ATBS		
MTS			
MT			
CFUN			
CPROC			
PMTS	PMTS -> TD id PMT		
PMT			
RET			
SB			
INST	INST -> INS INST		
INS	INS -> DCL		
DCL	DCL -> TD id TDCL ;		
TDCL			
DCLA			
EOV			
IDS			
ASG			
IN			
OUT			
LFUN			
LPROC			
IF			
WH			
CTE			
FOR			
ARGS			
VL			
EXP			
EXPS			
EXPA			
EXPB			
EXPC			
EXPD			
EXPE			
EXPF			
EXPG			
EXPH			
VLEXP			
CR			
NUM			
INT			
TD	TD -> carac		
ID			
IDOB		IDOB -> : id	

Pruebas de ejecución

Ejemplo de for

```
prin { kor (ente variable = 34 ; variable < total ; variable = 1+variable) { } }
```

Análisis léxico

-----Lista de Tokens-----		
Palabras Reservadas:		
prin	-----	205
kor	-----	215
ente	-----	206
Identificadores:		
variable	-----	100
total	-----	101
Numeros Enteros:		
34	-----	300
1	-----	301
Símbolos:		
{	-----	519
(-----	513
=	-----	507
;	-----	518
<	-----	509
+	-----	500
)	-----	514
}	-----	520

Análisis sintáctico

```
Pila: $ S | Cadena: prin { kor ( ente id = num ; id < id ; id = num + id ) { } } $ |S-->SA SB
Pila: $ SB SA | Cadena: prin { kor ( ente id = num ; id < id ; id = num + id ) { } } $ |SA-->&
Pila: $ SB | Cadena: prin { kor ( ente id = num ; id < id ; id = num + id ) { } } $ |SB-->prin { INST }
Pila: $ } INST | Cadena: kor ( ente id = num ; id < id ; id = num + id ) { } } $ |INST-->INS INST
Pila: $ } INST INS | Cadena: kor ( ente id = num ; id < id ; id = num + id ) { } } $ |INS-->FOR
Pila: $ } INST FOR | Cadena: kor ( ente id = num ; id < id ; id = num + id ) { } } $ |FOR-->kor ( ente ID = num ; EXP ; ASG ) { INST }
Pila: $ } INST } INST { } ASG ; EXP ; num = ID | Cadena: id = num ; id < id ; id = num + id ) { } } $ |ID-->id IDOB
Pila: $ } INST } INST { } ASG ; EXP ; num = IDOB | Cadena: id = num ; id < id ; id = num + id ) { } } $ |IDOB-->&
Pila: $ } INST } INST { } ASG ; EXP | Cadena: id < id ; id = num + id ) { } } $ |EXP-->EXPA EXPS
Pila: $ } INST } INST { } ASG ; EXPS EXPA | Cadena: id < id ; id = num + id ) { } } $ |EXPA-->EXPC EXPB
Pila: $ } INST } INST { } ASG ; EXPS EXPB EXPC | Cadena: id < id ; id = num + id ) { } } $ |EXPC-->EXPE EXPD
Pila: $ } INST } INST { } ASG ; EXPS EXPB EXPD EXPE | Cadena: id < id ; id = num + id ) { } } $ |EXPE-->EXPG EXPF
Pila: $ } INST } INST { } ASG ; EXPS EXPB EXPD EXPF EXPG | Cadena: id < id ; id = num + id ) { } } $ |EXPG-->VLEXP EXPH
Pila: $ } INST } INST { } ASG ; EXPS EXPB EXPD EXPF EXPH VLEXP | Cadena: id < id ; id = num + id ) { } } $ |VLEXP-->id
Pila: $ } INST } INST { } ASG ; EXPS EXPB EXPD EXPF EXPH | Cadena: < id ; id = num + id ) { } } $ |EXPH-->&
Pila: $ } INST } INST { } ASG ; EXPS EXPB EXPD EXPF | Cadena: < id ; id = num + id ) { } } $ |EXPF-->&
Pila: $ } INST } INST { } ASG ; EXPS EXPB EXPD | Cadena: < id ; id = num + id ) { } } $ |EXPD-->< EXPE EXPD
Pila: $ } INST } INST { } ASG ; EXPS EXPB EXPD EXPE | Cadena: id ; id = num + id ) { } } $ |EXPE-->EXPG EXPF
Pila: $ } INST } INST { } ASG ; EXPS EXPB EXPD EXPF EXPG | Cadena: id ; id = num + id ) { } } $ |EXPG-->VLEXP EXPH
Pila: $ } INST } INST { } ASG ; EXPS EXPB EXPD EXPF EXPH VLEXP | Cadena: id ; id = num + id ) { } } $ |VLEXP-->id
Pila: $ } INST } INST { } ASG ; EXPS EXPB EXPD EXPF | Cadena: ; id = num + id ) { } } $ |EXPH-->&
Pila: $ } INST } INST { } ASG ; EXPS EXPB EXPD EXPF | Cadena: ; id = num + id ) { } } $ |EXPF-->&
Pila: $ } INST } INST { } ASG ; EXPS EXPB | Cadena: ; id = num + id ) { } } $ |EXPD-->&
Pila: $ } INST } INST { } ASG ; EXPS EXPB | Cadena: ; id = num + id ) { } } $ |EXPB-->&
Pila: $ } INST } INST { } ASG ; EXPS | Cadena: ; id = num + id ) { } } $ |EXPS-->&
Pila: $ } INST } INST { } ASG | Cadena: id = num + id ) { } } $ |ASG-->ID = VL
Pila: $ } INST } INST { } VL = ID | Cadena: id = num + id ) { } } $ |ID-->id IDOB
Pila: $ } INST } INST { } VL = IDOB | Cadena: id = num + id ) { } } $ |IDOB-->&
Pila: $ } INST } INST { } VL | Cadena: num + id ) { } } $ |VL-->EXP
Pila: $ } INST } INST { } EXP | Cadena: num + id ) { } } $ |EXP-->EXPA EXPS
```

```

Pila: $ } INST } INST { } EXPS EXPA |Cadena: num + id ) { } } $ |EXPA-->EXPC EXPB
Pila: $ } INST } INST { } EXPS EXPB EXPC |Cadena: num + id ) { } } $ |EXPC-->EXPE EXPD
Pila: $ } INST } INST { } EXPS EXPB EXPD EXPE |Cadena: num + id ) { } } $ |EXPE-->EXPG EXPF
Pila: $ } INST } INST { } EXPS EXPB EXPD EXPF EXPG |Cadena: num + id ) { } } $ |EXPG-->VLEXP EXPH
Pila: $ } INST } INST { } EXPS EXPB EXPD EXPF EXPH VLEXP |Cadena: num + id ) { } } $ |VLEXP-->num
Pila: $ } INST } INST { } EXPS EXPB EXPD EXPF EXPH |Cadena: + id ) { } } $ |EXPH-->&
Pila: $ } INST } INST { } EXPS EXPB EXPD EXPF |Cadena: + id ) { } } $ |EXPF-->&
Pila: $ } INST } INST { } EXPS EXPB EXPD |Cadena: + id ) { } } $ |EXPD-->&
Pila: $ } INST } INST { } EXPS EXPB |Cadena: + id ) { } } $ |EXPB-->&
Pila: $ } INST } INST { } EXPS |Cadena: + id ) { } } $ |EXPS-->+ EXPA EXPS
Pila: $ } INST } INST { } EXPS EXPA |Cadena: id ) { } } $ |EXPA-->EXPC EXPB
Pila: $ } INST } INST { } EXPS EXPB EXPC |Cadena: id ) { } } $ |EXPC-->EXPE EXPD
Pila: $ } INST } INST { } EXPS EXPB EXPD EXPE |Cadena: id ) { } } $ |EXPE-->EXPG EXPF
Pila: $ } INST } INST { } EXPS EXPB EXPD EXPF EXPG |Cadena: id ) { } } $ |EXPG-->VLEXP EXPH
Pila: $ } INST } INST { } EXPS EXPB EXPD EXPF EXPH VLEXP |Cadena: id ) { } } $ |VLEXP-->id
Pila: $ } INST } INST { } EXPS EXPB EXPD EXPF EXPH |Cadena: ) { } } $ |EXPH-->&
Pila: $ } INST } INST { } EXPS EXPB EXPD EXPF |Cadena: ) { } } $ |EXPF-->&
Pila: $ } INST } INST { } EXPS EXPB EXPD |Cadena: ) { } } $ |EXPD-->&
Pila: $ } INST } INST { } EXPS EXPB |Cadena: ) { } } $ |EXPB-->&
Pila: $ } INST } INST { } EXPS |Cadena: ) { } } $ |EXPS-->&
Pila: $ } INST } INST |Cadena: } } $ |INST-->&
Pila: $ } INST |Cadena: } $ |INST-->&
CADENA ACEPTADA

```

Se acepta la cadena

Ejemplo de creación de clase

```
klase automovil{ente codigo; logic prendido; sinretorno prenderAutomovil(){} } prin{}
```

Análisis léxico

```

-----Lista de Tokens-----

Palabras Reservadas:
klase ----- 220
ente ----- 206
logic ----- 208
sinretorno ----- 200
prin ----- 205

Identificadores:
automovil ----- 100
codigo ----- 101
prendido ----- 102
prenderAutomovil ----- 103

Símbolos:
{ ----- 519
; ----- 518
( ----- 513
) ----- 514
} ----- 520

```

Análisis sintáctico

```

Pila: $ S |Cadena: klase id { ente id ; logic id ; sinretorno id ( ) { } } prin { } $ |S-->SA SB
Pila: $ SB SA |Cadena: klase id { ente id ; logic id ; sinretorno id ( ) { } } prin { } $ |SA-->FCP SA
Pila: $ SB SA FCP |Cadena: klase id { ente id ; logic id ; sinretorno id ( ) { } } prin { } $ |FCP-->CL
Pila: $ SB SA CL |Cadena: klase id { ente id ; logic id ; sinretorno id ( ) { } } prin { } $ |CL-->klase id { ATBS MTS }
Pila: $ SB SA } MTS ATBS |Cadena: ente id ; logic id ; sinretorno id ( ) { } } prin { } $ |ATBS-->DCL ATBS
Pila: $ SB SA } MTS ATBS DCL |Cadena: ente id ; logic id ; sinretorno id ( ) { } } prin { } $ |DCL-->TD id TDCL ;
Pila: $ SB SA } MTS ATBS ; TDCL id TD |Cadena: ente id ; logic id ; sinretorno id ( ) { } } prin { } $ |TD-->ente
Pila: $ SB SA } MTS ATBS ; TDCL |Cadena: ; logic id ; sinretorno id ( ) { } } prin { } $ |TDCL-->IDS
Pila: $ SB SA } MTS ATBS ; IDS |Cadena: ; logic id ; sinretorno id ( ) { } } prin { } $ |IDS-->&
Pila: $ SB SA } MTS ATBS |Cadena: logic id ; sinretorno id ( ) { } } prin { } $ |ATBS-->DCL ATBS
Pila: $ SB SA } MTS ATBS DCL |Cadena: logic id ; sinretorno id ( ) { } } prin { } $ |DCL-->TD id TDCL ;
Pila: $ SB SA } MTS ATBS ; TDCL id TD |Cadena: logic id ; sinretorno id ( ) { } } prin { } $ |TD-->logic
Pila: $ SB SA } MTS ATBS ; TDCL |Cadena: ; sinretorno id ( ) { } } prin { } $ |TDCL-->IDS
Pila: $ SB SA } MTS ATBS ; IDS |Cadena: ; sinretorno id ( ) { } } prin { } $ |IDS-->&
Pila: $ SB SA } MTS ATBS |Cadena: sinretorno id ( ) { } } prin { } $ |ATBS-->&
Pila: $ SB SA } MTS |Cadena: sinretorno id ( ) { } } prin { } $ |MTS-->MT MTS
Pila: $ SB SA } MTS MT |Cadena: sinretorno id ( ) { } } prin { } $ |MT-->CPROC
Pila: $ SB SA } MTS CPROC |Cadena: sinretorno id ( ) { } } prin { } $ |CPROC-->sinretorno id ( PMTS ) { INST }
Pila: $ SB SA } MTS } INST { } PMTS |Cadena: ) { } } prin { } $ |PMTS-->&
Pila: $ SB SA } MTS } INST |Cadena: } } prin { } $ |INST-->&
Pila: $ SB SA } MTS |Cadena: } prin { } $ |MTS-->&
Pila: $ SB SA |Cadena: prin { } $ |SA-->&
Pila: $ SB |Cadena: prin { } $ |SB-->prin { INST }
Pila: $ } INST |Cadena: } $ |INST-->&
CADENA ACEPTADA

```

Se acepta la cadena

Ejemplo de creación de función

```
conretorno ente Funcion1(ente i, logic flag){
    ki(id){ flag = falso; }
    kino { }
    devuelve valor;
}
prin{ }
```

Análisis léxico

-----Lista de Tokens-----		
Palabras Reservadas:		
conretorno	-----	222
ente	-----	206
logic	-----	208
ki	-----	211
falso	-----	204
kino	-----	212
devuelve	-----	219
prin	-----	205
Identificadores:		
Funcion1	-----	100
i	-----	101
flag	-----	102
id	-----	103
valor	-----	104
Simbolos:		
(-----	513
,	-----	517
)	-----	514
{	-----	519
=	-----	507
;	-----	518
}	-----	520

Análisis sintáctico

```
Pila: $ S |Cadena: conretorno ente id ( ente id , logic id ) { ki ( id ) { id = falso ; } kino { } devuelve id ; } prin { } $ |S-->SA SB
Pila: $ SB SA |Cadena: conretorno ente id ( ente id , logic id ) { ki ( id ) { id = falso ; } kino { } devuelve id ; } prin { } $ |SA-->FCP SA
Pila: $ SB SA FCP |Cadena: conretorno ente id ( ente id , logic id ) { ki ( id ) { id = falso ; } kino { } devuelve id ; } prin { } $ |FCP-->CFUN
Pila: $ SB SA CFUN |Cadena: conretorno ente id ( ente id , logic id ) { ki ( id ) { id = falso ; } kino { } devuelve id ; } prin { } $ |CFUN-->conretorno
Pila: $ SB SA } RET INST { } PMTS ( id TD |Cadena: ente id ( ente id , logic id ) { ki ( id ) { id = falso ; } kino { } devuelve id ; } prin { } $ |TD-->e
Pila: $ SB SA } RET INST { } PMTS |Cadena: ente id , logic id { ki ( id ) { id = falso ; } kino { } devuelve id ; } prin { } $ |PMTS-->TD id PMT
Pila: $ SB SA } RET INST { } PMT id TD |Cadena: ente id , logic id { ki ( id ) { id = falso ; } kino { } devuelve id ; } prin { } $ |PMT-->, TD id PMTS
Pila: $ SB SA } RET INST { } PMTS |Cadena: id TD |Cadena: logic id { ki ( id ) { id = falso ; } kino { } devuelve id ; } prin { } $ |TD-->logic
Pila: $ SB SA } RET INST { } PMTS |Cadena: ) { ki ( id ) { id = falso ; } kino { } devuelve id ; } prin { } $ |PMTS-->&
Pila: $ SB SA } RET INST |Cadena: ki ( id ) { id = falso ; } kino { } devuelve id ; } prin { } $ |INST-->INS INST
Pila: $ SB SA } RET INST INS |Cadena: ki ( id ) { id = falso ; } kino { } devuelve id ; } prin { } $ |INS-->IF
Pila: $ SB SA } RET INST IF |Cadena: ki ( id ) { id = falso ; } kino { } devuelve id ; } prin { } $ |IF-->ki ( VL ) { INST } kino { INST }
Pila: $ SB SA } RET INST } INST { kino } INST { } VL |Cadena: id { id = falso ; } kino { } devuelve id ; } prin { } $ |VL-->EXP
Pila: $ SB SA } RET INST } INST { kino } INST { } EXP |Cadena: id { id = falso ; } kino { } devuelve id ; } prin { } $ |EXP-->EXPA EXPS
Pila: $ SB SA } RET INST } INST { kino } INST { } EXPS EXPA |Cadena: id { id = falso ; } kino { } devuelve id ; } prin { } $ |EXPA-->EXPC EXPB
Pila: $ SB SA } RET INST } INST { kino } INST { } EXPS EXPB EXPC |Cadena: id { id = falso ; } kino { } devuelve id ; } prin { } $ |EXPC-->EXPE EXPD
Pila: $ SB SA } RET INST } INST { kino } INST { } EXPS EXPB EXPD EXPE |Cadena: id { id = falso ; } kino { } devuelve id ; } prin { } $ |EXPE-->EXPG EXPF
Pila: $ SB SA } RET INST } INST { kino } INST { } EXPS EXPB EXPD EXPF EXPG |Cadena: id { id = falso ; } kino { } devuelve id ; } prin { } $ |EXPG-->VLEX
Pila: $ SB SA } RET INST } INST { kino } INST { } EXPS EXPB EXPD EXPF EXPH VLEXP |Cadena: id { id = falso ; } kino { } devuelve id ; } prin { } $ |VLEXP
Pila: $ SB SA } RET INST } INST { kino } INST { } EXPS EXPB EXPD EXPF EXPH |Cadena: ) { id = falso ; } kino { } devuelve id ; } prin { } $ |EXPH-->&
Pila: $ SB SA } RET INST } INST { kino } INST { } EXPS EXPB EXPD EXPF |Cadena: { id = falso ; } kino { } devuelve id ; } prin { } $ |EXPF-->&
Pila: $ SB SA } RET INST } INST { kino } INST { } EXPS EXPB EXPD |Cadena: { id = falso ; } kino { } devuelve id ; } prin { } $ |EXPD-->&
Pila: $ SB SA } RET INST } INST { kino } INST { } EXPS EXPB |Cadena: { id = falso ; } kino { } devuelve id ; } prin { } $ |EXPB-->&
Pila: $ SB SA } RET INST } INST { kino } INST { } EXPS |Cadena: ) { id = falso ; } kino { } devuelve id ; } prin { } $ |EXPS-->&
Pila: $ SB SA } RET INST } INST { kino } INST |Cadena: id = falso ; } kino { } devuelve id ; } prin { } $ |INST-->INS INST
Pila: $ SB SA } RET INST } INST { kino } INST INS |Cadena: id = falso ; } kino { } devuelve id ; } prin { } $ |INS-->ASG ;
Pila: $ SB SA } RET INST } INST { kino } INST ; ASG |Cadena: id = falso ; } kino { } devuelve id ; } prin { } $ |ASG-->ID = VL
Pila: $ SB SA } RET INST } INST { kino } INST ; VL = ID |Cadena: id = falso ; } kino { } devuelve id ; } prin { } $ |ID-->ID IOOB
Pila: $ SB SA } RET INST } INST { kino } INST ; VL = IDOB |Cadena: = falso ; } kino { } devuelve id ; } prin { } $ |IDOB-->&
Pila: $ SB SA } RET INST } INST { kino } INST ; VL |Cadena: falso ; } kino { } devuelve id ; } prin { } $ |VL-->EXP
Pila: $ SB SA } RET INST } INST { kino } INST ; EXP |Cadena: falso ; } kino { } devuelve id ; } prin { } $ |EXP-->EXPA EXPS
```

```

Pila: $ SB SA } RET INST } INST { kino } INST ; EXPS EXPA |Cadena: falso ; } kino { } devuelve id ; } prin { } $ |EXPA-->EXPC EXPB
Pila: $ SB SA } RET INST } INST { kino } INST ; EXPS EXPB EXPC |Cadena: falso ; } kino { } devuelve id ; } prin { } $ |EXPC-->EXPE EXPD
Pila: $ SB SA } RET INST } INST { kino } INST ; EXPS EXPB EXPD EXPE |Cadena: falso ; } kino { } devuelve id ; } prin { } $ |EXPE-->EXPG EXPF
Pila: $ SB SA } RET INST } INST { kino } INST ; EXPS EXPB EXPD EXPF EXPG |Cadena: falso ; } kino { } devuelve id ; } prin { } $ |EXPG-->VLEXP EXPH
Pila: $ SB SA } RET INST } INST { kino } INST ; EXPS EXPB EXPD EXPF EXPH VLEXP |Cadena: falso ; } kino { } devuelve id ; } prin { } $ |VLEXP-->falso
Pila: $ SB SA } RET INST } INST { kino } INST ; EXPS EXPB EXPD EXPF EXPH |Cadena: ; } kino { } devuelve id ; } prin { } $ |EXPH-->&
Pila: $ SB SA } RET INST } INST { kino } INST ; EXPS EXPB EXPD EXPF |Cadena: ; } kino { } devuelve id ; } prin { } $ |EXPF-->&
Pila: $ SB SA } RET INST } INST { kino } INST ; EXPS EXPB EXPD |Cadena: ; } kino { } devuelve id ; } prin { } $ |EXPD-->&
Pila: $ SB SA } RET INST } INST { kino } INST ; EXPS EXPB |Cadena: ; } kino { } devuelve id ; } prin { } $ |EXPB-->&
Pila: $ SB SA } RET INST } INST { kino } INST ; EXPS |Cadena: ; } kino { } devuelve id ; } prin { } $ |EXPS-->&
Pila: $ SB SA } RET INST } INST { kino } INST |Cadena: } kino { } devuelve id ; } prin { } $ |INST-->&
Pila: $ SB SA } RET INST } INST |Cadena: } devuelve id ; } prin { } $ |INST-->&
Pila: $ SB SA } RET INST |Cadena: devuelve id ; } prin { } $ |INST-->&
Pila: $ SB SA } RET |Cadena: devuelve id ; } prin { } $ |RET-->devuelve VL ;
Pila: $ SB SA } ; VL |Cadena: id ; } prin { } $ |VL-->EXP
Pila: $ SB SA } ; EXP |Cadena: id ; } prin { } $ |EXP-->EXPA EXPS
Pila: $ SB SA } ; EXPS EXPA |Cadena: id ; } prin { } $ |EXPA-->EXPC EXPB
Pila: $ SB SA } ; EXPS EXPB EXPC |Cadena: id ; } prin { } $ |EXPC-->EXPE EXPD
Pila: $ SB SA } ; EXPS EXPB EXPD EXPE |Cadena: id ; } prin { } $ |EXPE-->EXPG EXPF
Pila: $ SB SA } ; EXPS EXPB EXPD EXPF EXPG |Cadena: id ; } prin { } $ |EXPG-->VLEXP EXPH
Pila: $ SB SA } ; EXPS EXPB EXPD EXPF EXPH VLEXP |Cadena: id ; } prin { } $ |VLEXP-->id
Pila: $ SB SA } ; EXPS EXPB EXPD EXPF EXPH |Cadena: ; } prin { } $ |EXPH-->&
Pila: $ SB SA } ; EXPS EXPB EXPD EXPF |Cadena: ; } prin { } $ |EXPF-->&
Pila: $ SB SA } ; EXPS EXPB EXPD |Cadena: ; } prin { } $ |EXPD-->&
Pila: $ SB SA } ; EXPS EXPB |Cadena: ; } prin { } $ |EXPB-->&
Pila: $ SB SA } ; EXPS |Cadena: ; } prin { } $ |EXPS-->&
Pila: $ SB SA |Cadena: prin { } $ |SA-->&
Pila: $ SB |Cadena: prin { } $ |SB-->prin { INST }
Pila: $ } INST |Cadena: } $ |INST-->&

```

CADENA ACEPTADA

Se acepta la cadena

Analizador Semántico del lenguaje KOD

El análisis semántico de un procesador de lenguaje es la fase encargada de detectar la validez semántica de las sentencias aceptadas por el analizador sintáctico. Este proceso se realiza mediante la decoración o anotación del análisis sintáctico. Este asigna información adicional a los nodos del análisis sintáctico.

El análisis Sintáctico decorado es una ampliación del análisis sintáctico en que a cada nodo del mismo se le añade atributos indicando las propiedades necesarias de la construcción sintáctica que representan.

Reglas Semánticas

Las relaciones entre los valores de los atributos definidos sobre una gramática atribuida se especifican mediante las reglas semánticas.

	Producción	Regla
1	CL -> klase id { ATBS MTS }	//Añadimos clase como tipo de dato Añade_Clase(id.entrada)
2	CFUN -> conretorno TD id (PMTS) { INST RET }	CFUN.tipo = TD.tipo Si (TD.tipo = RET.tipo) { Añade_Funciones(id.entrada,TD.tipo) id.tipo = TD.tipo } Sino { Escribir("Error de tipo al crear funcion")
3	CPROC-> sinretorno id (PMTS) { INST }	Añade_Procedimiento(id.entrada)
7	DCL -> TD id TDCL ;	Añade_Variables(id.entrada,TD.tipo) TDCL.tipo=TD.tipo id.tipo = TD.tipo
8	TDCL -> IDS	IDS.tipo=TDCL.tipo
9	IDS -> , id IDS1	IDS1.tipo=IDS.tipo Añade_Variables(id.entrada,IDS.tipo) id.tipo = IDS.tipo
10	IDS -> λ	
11	ASG -> ID = VL	Si_igual_tipodedato(ID.tipo,VL.tipo){ ID.valor=VL.valor } Sino { Escribir("Error de compatibilidad de variable") }
12	IN -> ingresar (ID)	IN.tipo=ID.tipo
13	OUT -> imprimir (VL)	

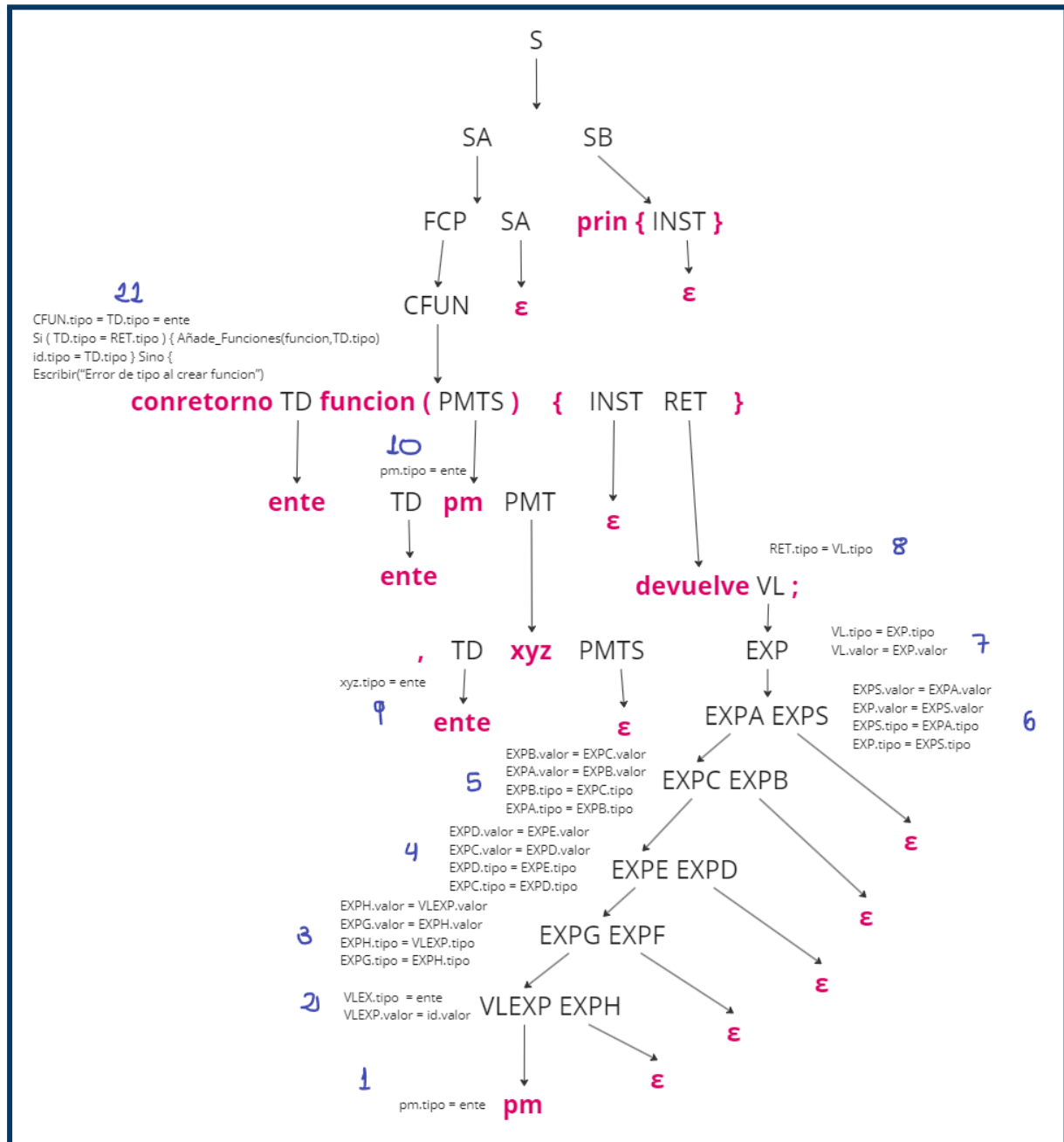
14	LFUN -> fun ID (ARGS)	LFUN.tipo = id.tipo LFUN.valor = id.valor
15	PMTS -> TD id PMT	id.tipo = TD.tipo
16	PMTS -> λ	
17	PMT -> , TD id PMTS	id.tipo = TD.tipo
18	RET -> devuelve VL ;	RET.tipo = VL.tipo
19	CTE -> konst TD id	id.tipo = TD.tipo
20	FOR -> kor (ente ID = INT ; EXP ; ASG) { INST }	ID.tipo=ente ID.valor=INT.valor
21	ARGS -> VL ARGS	AnadeListaArguFunc(VL.valor,VL.tipo)
22	ARGS -> , VL ARGS	AnadeListaArguFunc(VL.valor,VL.tipo)
23	ARGS -> λ	
24	VL -> CR	VL.tipo = CR.tipo VL.valor = CR.valor
25	VL -> EXP	VL.tipo = EXP.tipo VL.valor = EXP.valor
26	VL -> LFUN	VL.tipo = LFUN.tipo VL.valor = LFUN.valor
27	EXP -> EXPA EXPS	EXPS.valor = EXPA.valor EXP.valor = EXPS.valor EXPS.tipo = EXPA.tipo EXP.tipo = EXPS.tipo
28	EXPS -> + EXPA EXPS1	EXPS1.valor = EXPS.valor + EXPA.valor EXPS.valor = EXPS1.valor EXPS1.tipo = mayortipo(EXPS.tipo, EXPA.tipo) EXPS.tipo = EXPS1.tipo
29	EXPS -> - EXPA EXPS1	EXPS1.valor = EXPS.valor - EXPA.valor EXPS.valor = EXPS1.valor EXPS1.tipo = mayortipo(EXPS.tipo, EXPA.tipo) EXPS.tipo = EXPS1.tipo
30	EXPS -> λ	
31	EXPA -> EXPC EXPB	EXPB.valor = EXPC.valor EXPA.valor = EXPB.valor EXPB.tipo = EXPC.tipo EXPA.tipo = EXPB.tipo
32	EXPB -> * EXPC EXPB1	EXPB1.valor = EXPB.valor * EXPC.valor EXPB.valor = EXPB1.valor EXPB1.tipo = mayortipo(EXPB.tipo, EXPC.tipo) EXPB.tipo = EXPB1.tipo

33	EXPB -> / EXPC EXPB1	EXPB1.valor = EXPB.valor / EXPC.valor no_es_0(EXPC) EXPB.valor = EXPB1.valor EXPB1.tipo = mayortipo(EXPB.tipo, EXPC.tipo) EXPB.tipo = EXPB1.tipo
34	EXPB -> % EXPC EXPB1	EXPB1.valor = modulo(EXPB.valor, EXPC.valor) EXPB.valor = EXPB1.valor EXPB1.tipo = mayortipo(EXPB.tipo, EXPC.tipo) EXPB.tipo = EXPB1.tipo
35	EXPB -> λ	
36	EXPC -> EXPE EXPD	EXPD.valor = EXPE.valor EXPC.valor = EXPD.valor EXPD.tipo = EXPE.tipo EXPC.tipo = EXPD.tipo
37	EXPD -> > ! EXPE EXPD1	EXPD1.valor = mayorOIgual(EXPD.valor , EXPE.valor) EXPD.valor = EXPD1.valor EXPD1.tipo = logico EXPD.tipo = EXPD1.tipo
38	EXPD -> < ! EXPE EXPD1	EXPD1.valor = menorOIgual(EXPD.valor , EXPE.valor) EXPD.valor = EXPD1.valor EXPD1.tipo = logico EXPD.tipo = EXPD1.tipo
39	EXPD -> == EXPE EXPD1	EXPD1.valor = SiIgual(EXPD.valor , EXPE.valor) EXPD.valor = EXPD1.valor EXPD1.tipo = logico EXPD.tipo = EXPD1.tipo
40	EXPD -> != EXPE EXPD1	EXPD1.valor = NoEsIgual(EXPD.valor , EXPE.valor) EXPD.valor = EXPD1.valor EXPD1.tipo = logico EXPD.tipo = EXPD1.tipo
41	EXPD -> λ	
42	EXPE -> EXPG EXPF	EXPF.valor = EXPG.valor EXPE.valor = EXPF.valor EXPF.tipo = EXPG.tipo EXPE.tipo = EXPF.tipo
43	EXPF -> o EXPG EXPF1	EXPF1.valor = conjuncion(EXPF.valor , EXPG.valor) EXPF.valor = EXPF1.valor EXPF1.tipo = logic EXPF.tipo = EXPF1.tipo
44	EXPF -> λ	
45	EXPG -> VLEXP EXPH	EXPH.valor = VLEXP.valor EXPG.valor = EXPH.valor

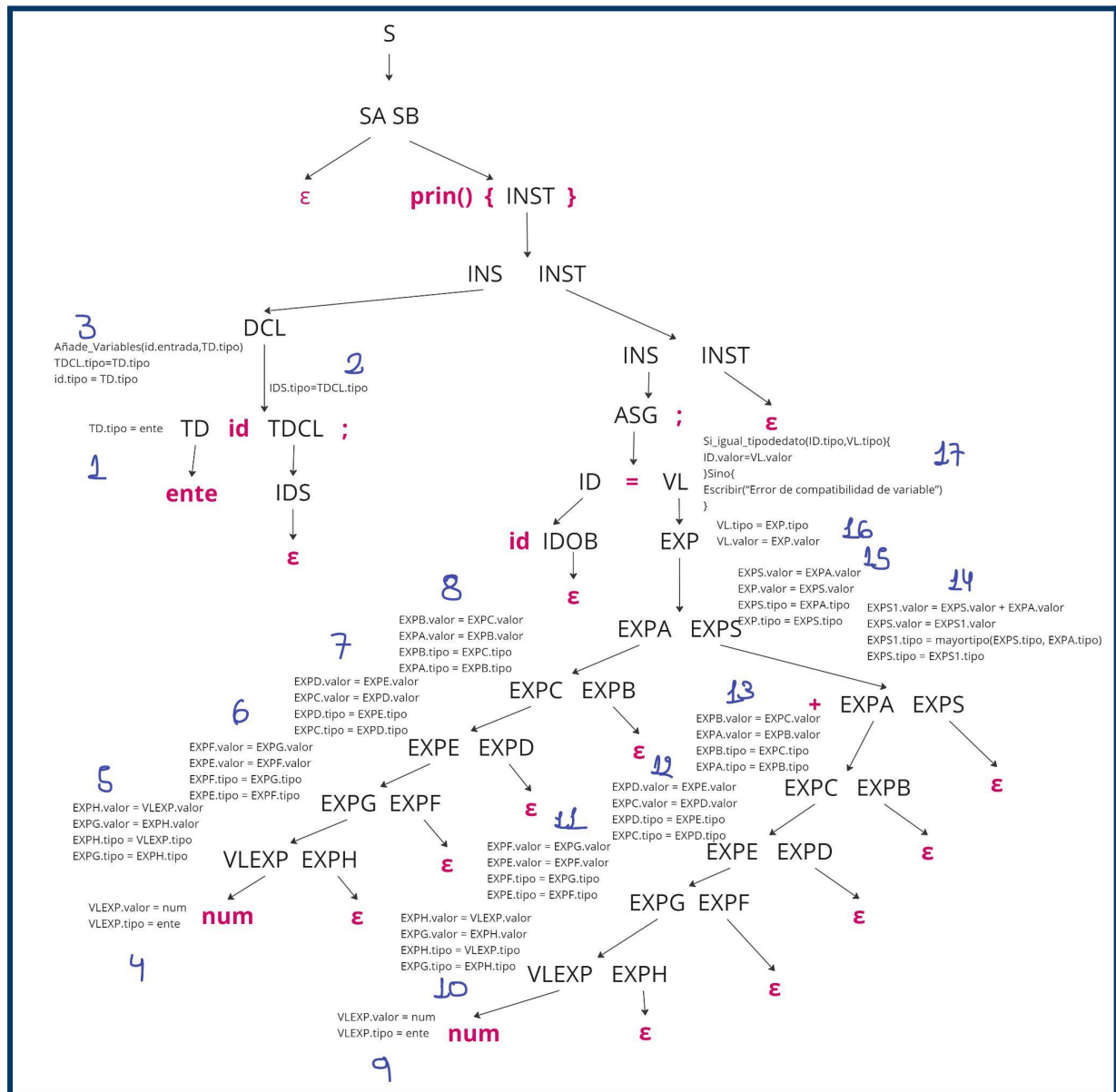
		EXPH.tipo = VLEXP.tipo EXPG.tipo = EXPH.tipo
46	EXPH -> y VLEXP EXPH1	EXPH1.valor =disyuncion(EXPH.valor , EXPG.valor) EXPH.valor = EXPH1.valor EXPH1.tipo = logic EXPH.tipo = EXPH1.tipo
47	EXPH -> λ	
48	VLEXP-> (EXP)	VLEXP.tipo = EXP.tipo VLEXP.valor = EXP.valor
49	VLEXP -> num	VLEXP.valor = num VLEXP.tipo = ente VLEXP.tipo = real
50	VLEXP -> verdadero	VLEXP.valor = verdadero
51	VLEXP -> falso	VLEXP.valor = falso
52	VLEXP -> id	VLEXP.tipo = id.tipo VLEXP.valor = id.valor
53	CR -> c	CR.tipo = carac CR.valor = c
54	NUM -> num	NUM.tipo = ente NUM.tipo = real NUM.valor = num
55	INT -> 0	INT.valor = 0 INT.tipo = ente
56	TD -> ente	TD.tipo = ente
57	TD -> real	TD.tipo = real
58	TD -> logic	TD.tipo = logic
59	TD -> carac	TD.tipo = carac

Árboles sintácticos de prueba

Cadena de prueba: conretorno ente funcion (ente pm , ente xyz) { devuelve pm ; } prin() { }



Cadena de prueba: prin() { ente id ; id = num + num ; }



Cadena de prueba: prin { ente a ; a = 5 ; a = a + 7 ; }

