PROCESADORES DEL LENGUAJE PRÁCTICA FINAL

Construcción de un analizador del lenguaje BASIC

Grado en Ingeniería Informática por la Universidad de Alcalá

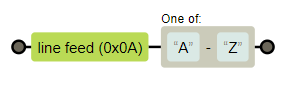
Rodrigo Pueblas Núñez – 09085810M – rodrigo.pueblas@hotmail.com

# ANALIZADOR LÉXICO

Para el desarrollo del presente apartado se ha utilizado la herramienta JFLEX realizando la integración con el entorno de desarrollo Eclipse, aún sin la compatibilidad con CUP, que utilizaré más adelante para enlazar con el resto de módulos del intérprete. Para ello, una vez instalado he creado un archivo de extensión “.lex” con el que defino los distintos tipos de tokens posibles en nuestro programa. Aunque algunos no son utilizados, sí vienen especificados en el lenguaje BASIC, así como en el enunciado de la práctica. Todos los autómatas han sido generados con la herramienta JFLAP, dado que ha sido la elegida por el profesor en el laboratorio.

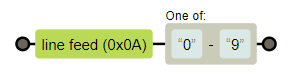
* Letra = [A-Z]

DFA:



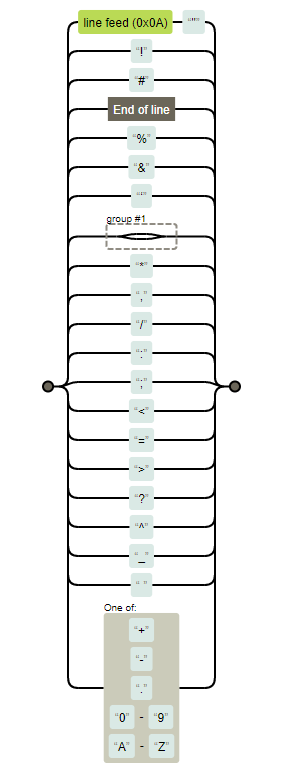
* Digito = [0-9]

DFA:



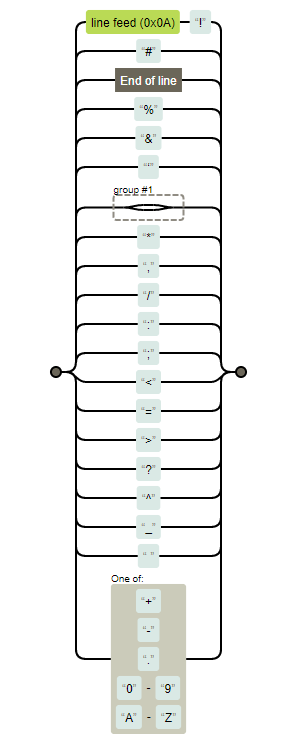
* Car\_Cadena = ” | {Car\_Cad\_Delimitado}

DFA:



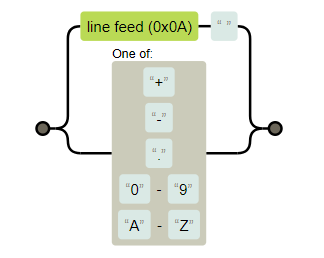
* Car\_Cad\_Delimitado = ! | # | $ | % | & | ‘ | ( | ) | \* | , | / | : | ; | < | = | > | ? | ^ | \_ | Car\_No\_Delimitado

DFA:



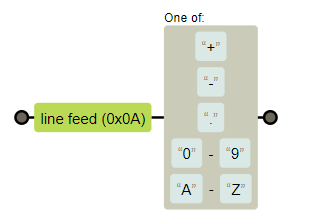
* Car\_No\_Delimitado = “ “ | Car\_Cadena\_Simple

DFA:



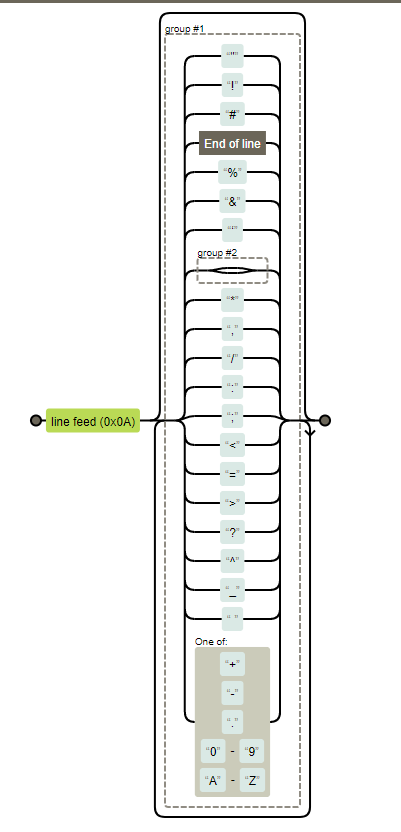
* Car\_Cadena\_Simple = + | - | . | Digito | Letra

DFA:



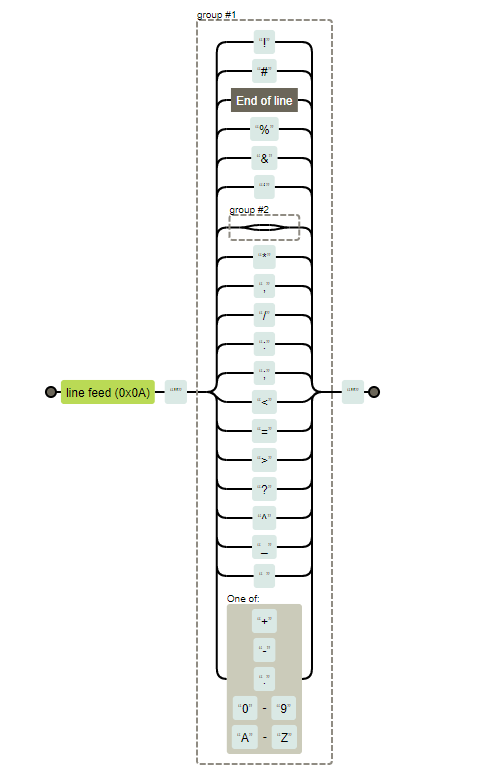
* Cad\_REM = Car\_Cadena\*

DFA:



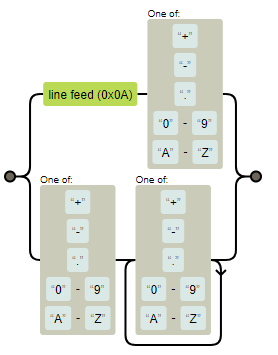
* Cad\_Delimitada = “ Car\_Cad\_Delimitado\* “

DFA:



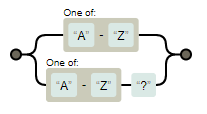
* Car\_No\_Delimitada = Car\_Cadena\_Simple | Car\_Cadena\_Simple Car\_No\_Delimitada\* Car\_Cadena\_Simple

DFA:



* Identificador = [A-Z] | [A-Z]$

DFA:



* Constante = Constante = {ConsNum} | {ConsCad} | {ConsNumDec} | {ConsNumEsc}

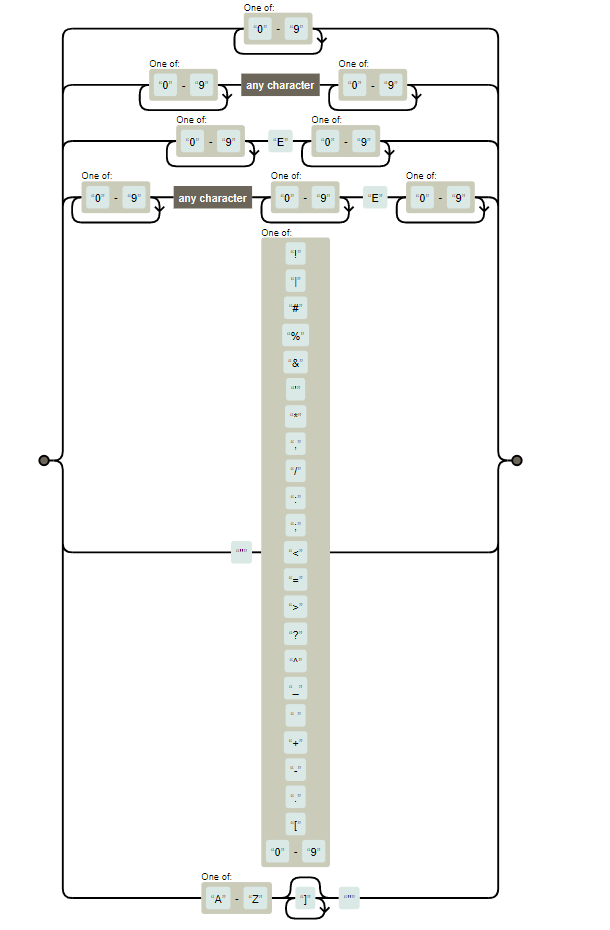
ConsNum = {Digito}+

ConsCad = {Cad\_Delimitada}

ConsNumDec = {Digito}+ \. {Digito}+

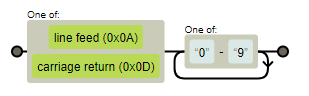
ConsNumEsc = {ConsNum}E{ConsNum} | {ConsNumDec}E{ConsNum}

DFA:



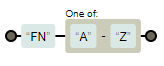
* Entrada = [\n\r] Digito+

DFA:



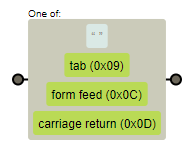
* FNX = FN[A-Z]

DFA:



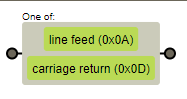
* Espacio = “ “ | \t | \f | \r

DFA:



* Salto = \n | \r | \r\n

DFA:



## Tabla de especificaciones léxicas:

|  |  |  |
| --- | --- | --- |
| EL1 | Cumplido 🗹 | Implementación: Imagen 1.1  Demostración: ficheros 1, 2, 3 |
| EL2 | Cumplido 🗹 | Implementación: Imagen 1.2  Demostración: ficheros 1, 2, 3 |
| EL3 | Cumplido 🗹 | Implementación: Imagen 1.2  Demostración: ficheros 2, 3 |
| EL4 | Cumplido 🗹 | Implementación: Imagen 1.3  Demostración: ficheros 1, 2, 3 |
| EL5 | Cumplido 🗹 | Implementación: Imagen 1.4  Demostración: ficheros 1, 2, 3 |
| EL6 | Cumplido 🗹 | Implementación: Imagen 1.5  Demostración: ficheros 1, 2, 3 |
| EL7 | Cumplido 🗹 | Implementación: Imagen 1.6  Demostración: ficheros 1, 2, 3 |

\*Nota: pese a que más ficheros cumplen los requisitos, se han redactado los tres primeros de forma que cumplan por sí mismos todos los requisitos.

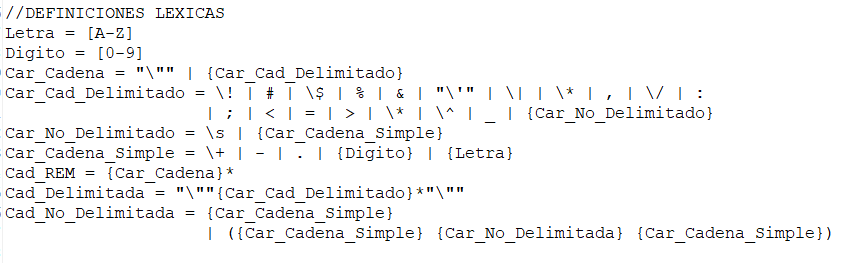


Imagen 1.1

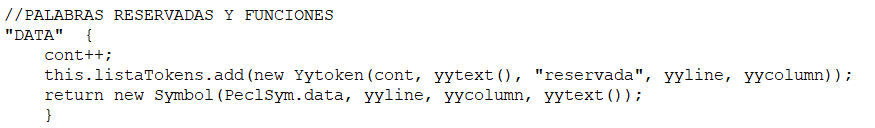


Imagen .2 – Se ha realizado lo mismo con el resto de palabras reservadas y funciones del lenguaje.



Imagen 1.3

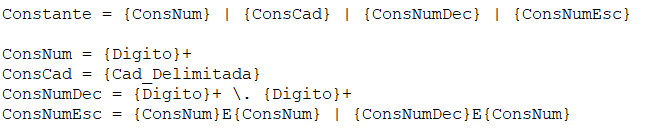


Imagen 1.4



Imagen 1.5

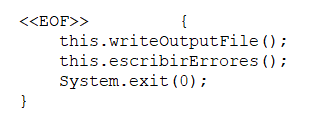


Imagen 1.6

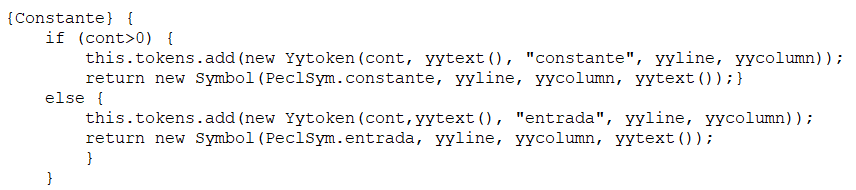
## Justificación del uso o no de estados léxicos:

Se ha decidido no utilizar estados léxicos, ya que el tamaño del léxico a implementar, que es reducido, parecía más sencillo definirlo como hemos indicado y pasar los símbolos directamente, utilizando también un manejador de errores que tiene en cuenta cualquier símbolo que no coincida con una expresión regular dada además de un error de cadena.

## Distinción entre token de entrada (comienzo de fichero) y token de constante numérica

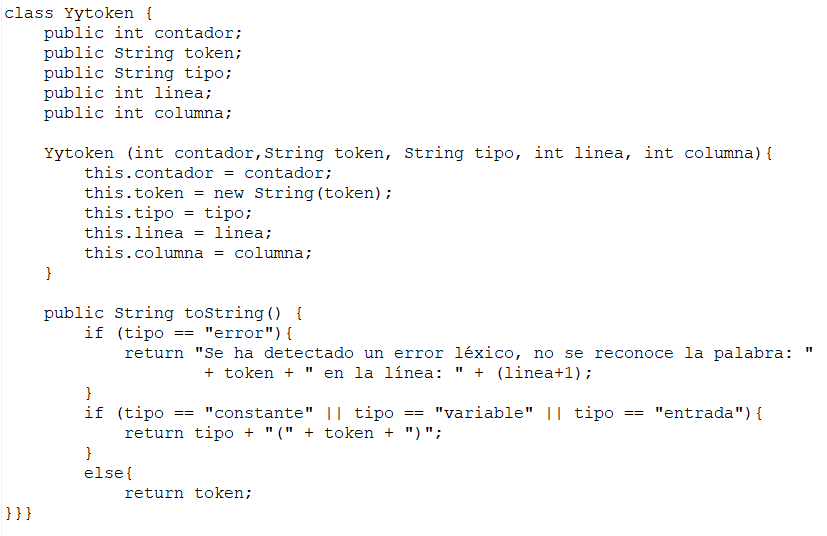
Un problema que se presentó al inicio de la práctica fue el de distinguir dos cadenas que a priori parecen idénticas: el indicador de línea y la constante numérica. Para solucionar esto, he definido “Entrada” como una constante numérica con un retorno de carro o salto de línea antes.

Esto seguía siendo problemático con el primer token de entrada. Para solucionarlo, utilicé el contador de tokens leídos hasta el momento para identificar si la constante numérica es leída como primer token, en cuyo caso se registra como Entrada.



## Clase Yytoken

Para un mayor control sobre los tokens leídos, los errores y facilitar la salida por pantalla de los datos necesarios, he implementado una clase Yytoken en el archivo “.lex”. Es una clase muy parecido a la clase Symbol de CUP, en la que guardaré el contador de tokens en ese momento, la cadena de entrada, el tipo y su posición.



Los tokens se van introduciendo en un ArrayList definido al inicio del programa a medida que se van leyendo, así como los errores se van introduciendo en otro ArrayList similar. Al final, se busca en estas estructuras los tokens a escribir por pantalla.

# ANALIZADOR SINTÁCTICO

## Gramática

Para la realización del análisis sintáctico, he construido la siguiente gramática, sin errores ni advertencias de reducción-desplazamiento, ni reducción-reducción. He eliminado todo fragmento de código correspondiente al analizador semántico para dejar la gramática limpia.

terminal String constante, variable, entrada;

terminal String menor, mayor, menor\_igual, mayor\_igual, igual, no\_igual;

terminal String abs, atn, cos, exp, t\_int, log, sgn, sin, sqr, tan, rnd;

terminal String potencia, multiplicacion, division, suma, resta;

terminal String data, def, dim, end, t\_for, gosub, t\_goto, t\_if;

terminal String input, let, next, on, print, randomize, read, restore, t\_return;

terminal String step, stop, then, to, fnx, rem;

terminal String parentesis\_izq, parentesis\_dch, coma, punto\_coma, eof;

terminal String constante, variable, entrada;

terminal String menor, mayor, menor\_igual, mayor\_igual, igual, no\_igual;

terminal String abs, atn, cos, exp, t\_int, log, sgn, sin, sqr, tan, rnd;

terminal String potencia, multiplicacion, division, suma, resta;

terminal String data, def, dim, end, t\_for, gosub, t\_goto, t\_if;

terminal String input, let, next, on, print, randomize, read, restore, t\_return;

terminal String step, stop, then, to, fnx, rem;

terminal String parentesis\_izq, parentesis\_dch, coma, punto\_coma, eof;

non terminal SentenciaNext, Programa, Comparacion, Funcion, Operador;

non terminal Operacion, Expresion;

non terminal Sentencia, SentenciaDef, SentenciaLet, SentenciaControl, SentenciaGoto;

non terminal SentenciaIf, SentenciaGosub, SentenciaOn, SentenciaStop, SentenciaFor;

non terminal SentenciaPrint, SentenciaInput, SentenciaData, SentenciaRead, SentenciaRestore;

non terminal SentenciaDim, SentenciaRem, SentenciaRandomize;

non terminal S, RestoFunciones, ParametroFuncion, ExtenderExp, CuerpoFuncion, SentenciasReturn;

non terminal MasEntradas, SalidaTexto, MasExpresiones, MasVariables, MasConstantes;

non terminal Matriz2, MasDim, Step, SentenciasNext, InputVar, Operacion1, Operacion2;

start with Programa;

Programa ::= Sentencia Programa | entrada end ;

Sentencia ::= entrada S | error ;

S ::= SentenciaDef | SentenciaLet | SentenciaControl | SentenciaFor | SentenciaPrint | SentenciaInput | SentenciaData | SentenciaRead | SentenciaRestore | SentenciaDim | SentenciaRandomize | SentenciaRem | SentenciaNext | error ;

Comparacion ::= menor | mayor | menor\_igual | mayor\_igual | igual | no\_igual ;

Funcion ::= log parentesis\_izq Operador:o parentesis\_dch | sqr parentesis\_izq Operador parentesis\_dch | RestoFunciones ;

RestoFunciones ::= abs ParametroFuncion ;

| atn ParametroFuncion ;

| cos ParametroFuncion:o ;

| exp ParametroFuncion ;

| t\_int ParametroFuncion ;

| sgn ParametroFuncion ;

| sin ParametroFuncion ;

| tan ParametroFuncion ;

ParametroFuncion::= parentesis\_izq Operador parentesis\_dch ;

Operador ::= constante | variable | rnd | Funcion ;

Operacion ::= potencia | Operacion1 ;

Operacion1 ::= multiplicacion | division | Operacion2 ;

Operacion2 ::= suma | resta ;

Expresion ::= parentesis\_izq Operador ExtenderExp parentesis\_dch;

ExtenderExp ::= Operacion Expresion:e | ;

SentenciaRem ::= rem ;

SentenciaDef ::= def fnx CuerpoFuncion | def error ;

CuerpoFuncion ::= parentesis\_izq variable parentesis\_dch igual Expresion | igual Expresion ;

SentenciaLet ::= let variable igual Expresion | let error ;

SentenciaControl::= SentenciaGoto | SentenciaIf | SentenciaGosub | SentenciaOn | SentenciaStop ;

SentenciaGoto ::= t\_goto constante | t\_goto error ;

SentenciaIf ::= t\_if Expresion Comparacion Expresion then constante | t\_if Expresion Comparacion Expresion then SentenciaGoto;

SentenciaGosub ::= gosub constante SentenciasReturn | gosub error ;

SentenciasReturn::= Sentencia SentenciasReturn | t\_return ;

SentenciaOn ::= on Expresion t\_goto constante:c MasEntradas | on error ;

MasEntradas ::= coma constante | ;

SentenciaStop ::= stop ;

SentenciaPrint ::= print Expresion:e MasExpresiones InputVar:v | print error ;

MasExpresiones ::= coma Expresion | ;

InputVar ::= punto\_coma variable | ;

SentenciaInput ::= input SalidaTexto variable MasVariables | input error ;

MasVariables ::= coma variable | ;

SalidaTexto ::= constante punto\_coma | ;

SentenciaData ::= data constante MasConstantes | data error ;

MasConstantes ::= coma constante | ;

SentenciaRead ::= read variable MasVariables | read error ;

SentenciaRestore::= restore ;

SentenciaDim ::= dim variable parentesis\_izq constante Matriz2 parentesis\_dch MasDim | dim error ;

MasDim ::= coma variable parentesis\_izq constante Matriz2 parentesis\_dch MasDim | ;

Matriz2 ::= coma constante | ;

SentenciaFor ::= t\_for variable igual Expresion to Expresion Step SentenciasNext | t\_for error ;

Step ::= step Expresion | | step error ;

SentenciasNext ::= Sentencia SentenciasNext | entrada next variable ;

SentenciaRandomize::= randomize ;

## Tabla de especificaciones sintácticas:

|  |  |  |
| --- | --- | --- |
| ES1 | Cumplido 🗹 | Implementación: Imagen 2.1  Demostración: ficheros 1, 2, 3 |
| ES2 | Cumplido 🗹 | Implementación: Imagen 2.2  Demostración: ficheros 2, 3 |
| ES3 | Cumplido 🗹 | Implementación: Imagen 2.3  Demostración: ficheros 2 |
| ES4 | Cumplido 🗹 | Implementación: Imagen 2.4  Demostración: ficheros 2 |
| ES5 | Cumplido 🗹 | Implementación: Imagen 2.5  Demostración: ficheros 1, 2, 3 |
| ES6 | Cumplido 🗹 | Implementación: Imagen 2.6  Demostración: ficheros 3 |
| ES7 | Cumplido 🗹 | Implementación: Imagen 2.7  Demostración: ficheros 3 |
| ES8 | Cumplido 🗹 | Implementación: Imagen 2.8  Demostración: ficheros 1 |
| ES9 | Cumplido 🗹 | Implementación: Imagen 2.9  Demostración: ficheros 1, 2, 3 |
| EL10 | Cumplido 🗹 | Implementación: Imagen 2.10  Demostración: ficheros 1, 2, 3 |
| ES11 | Cumplido 🗹 | Implementación: Imagen 2.11  Demostración: ficheros 2 |
| ES12 | Cumplido 🗹 | Implementación: Imagen 2.12  Demostración: ficheros 2 |
| ES13 | Cumplido 🗹 | Implementación: Imagen 2.13  Demostración: ficheros 2 |
| ES14 | Cumplido 🗹 | Implementación: Imagen 2.14  Demostración: ficheros 2 |
| ES15 | Cumplido 🗹 | Implementación: Imagen 2.15  Demostración: ficheros 2 |

\*Nota: pese a que más ficheros cumplen los requisitos, se han redactado los tres primeros de forma que cumplan por sí mismos todos los requisitos.

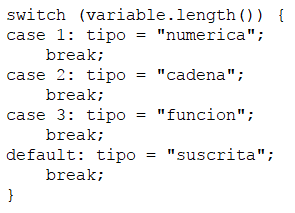


Imagen 2.1 – Dado que registramos un mismo token(variable) para todos los identificadores, distinguimos el tipo en función del tamaño: X es numérico simple, X$ de cadena, FNX de función y X(Y) u otros numérico suscrito.

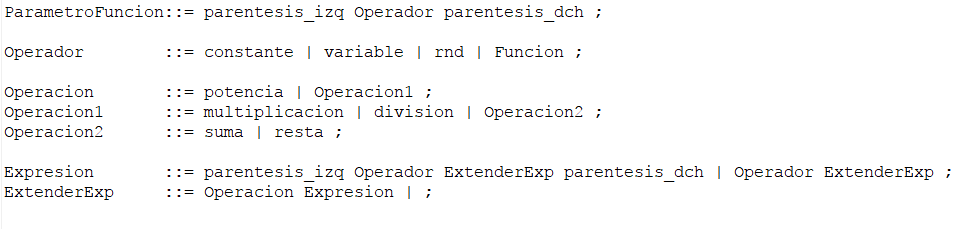


Imagen 2.2 – Se tiene en cuenta la prioridad de los operadores utilizando la propia gramática.

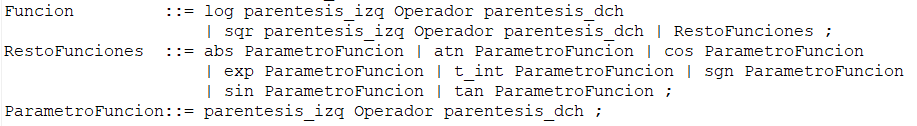


Imagen 2.3

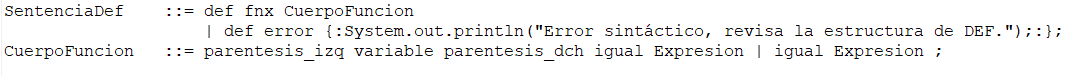


Imagen 2.4



Imagen 2.5



Imagen 2.6

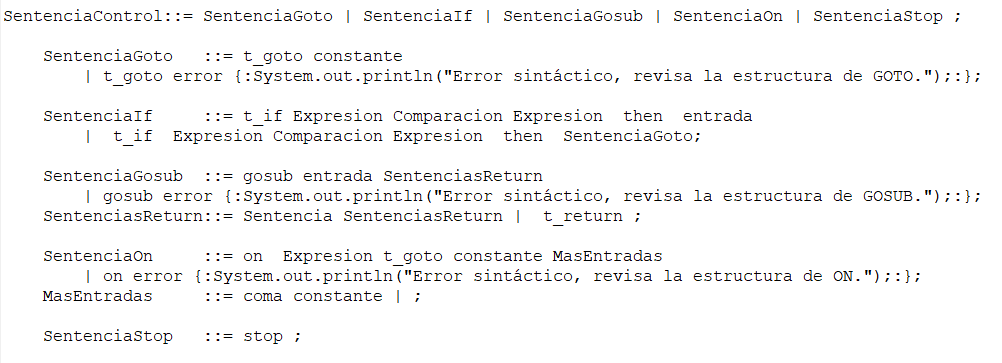


Imagen 2.7

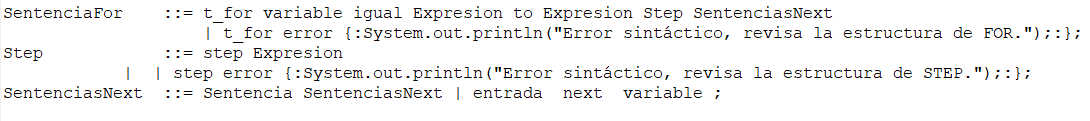


Imagen 2.8

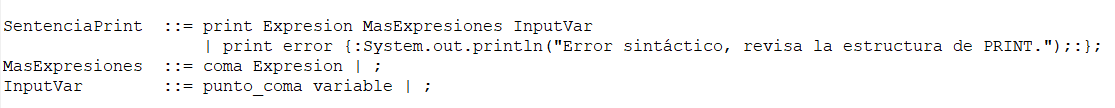


Imagen 2.9

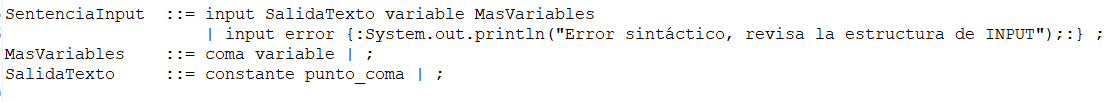


Imagen 2.10

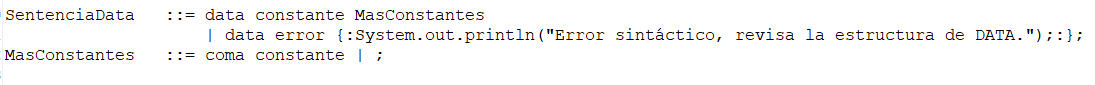


Imagen 2.11

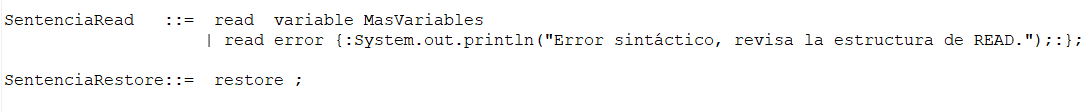


Imagen 2.12

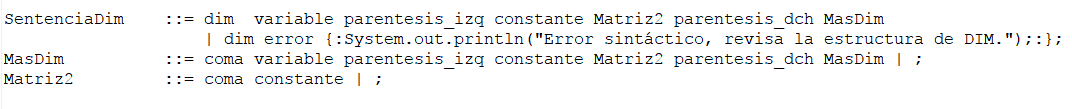


Imagen 2.13

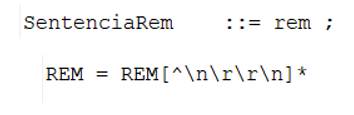


Imagen 2.14 – Arriba, la producción sintáctica. Abajo, la declaración del token léxico. Como se puede observar, para ignorar todos los comentarios hasta el fin de línea se reconoce como un único token REM toda la línea.



Imagen 2.15

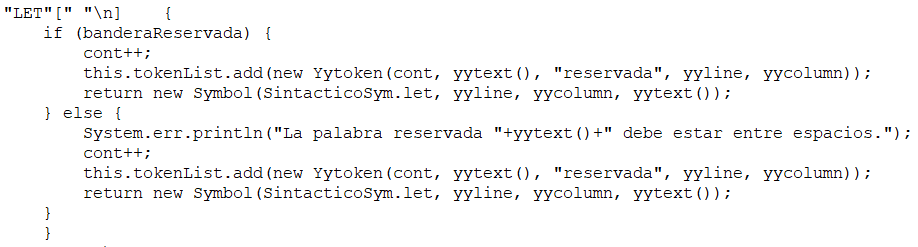
## Comprobación de carácter de espaciado delante y detrás de una palabra reservada:

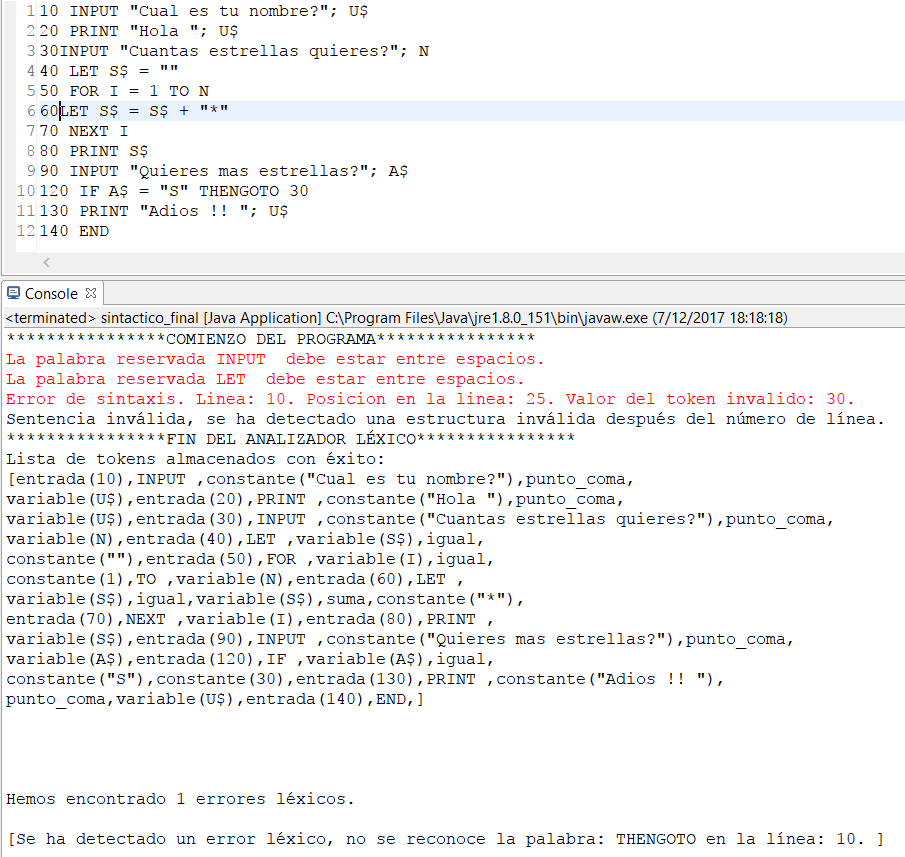
Pese a que este requisito es pedido en el apartado sintáctico, se ha dado como inviable tratarlo de este modo, dado que en ese caso habría que construir una gramática que contemplase todos los espacios posibles.

Sin embargo, se ha visto mucho más asequible tratar esto en el apartado léxico, registrando las palabras registradas con los propios espacios. La primera aproximación a esta solución fue pensada como simplemente incluir un espacio delante y otro detrás de cada palabra reservada. Por ejemplo, la definición “DATA” pasaría a ser “ DATA “. Pero esto no es del todo correcto, por dos motivos:

* Una palabra reservada puede estar al final de línea. Para esto, la nueva definición regular sería “ DATA”[“ “\n] en nuestro ejemplo.
* Pueden haber dos o más palabras reservadas consecutivas, siendo este un error en todo caso sintáctico por una mala ordenación de la sentencia. Para esto, se ha implementado una variable booleana “banderaReservada”, inicializada en “true”, que al registrar cualquier token que no sea una palabra reservada o un espacio pasará a “false”, y volverá a “true” al reconocer palabras reservadas o espacios. Esta bandera nos sirve para que quede en “true” si el último token registrado es un espacio o una palabra reservada, ocasiones en las que una palabra reservada no tendría errores léxicos.

Aquí la implementación y un ejemplo de uso:

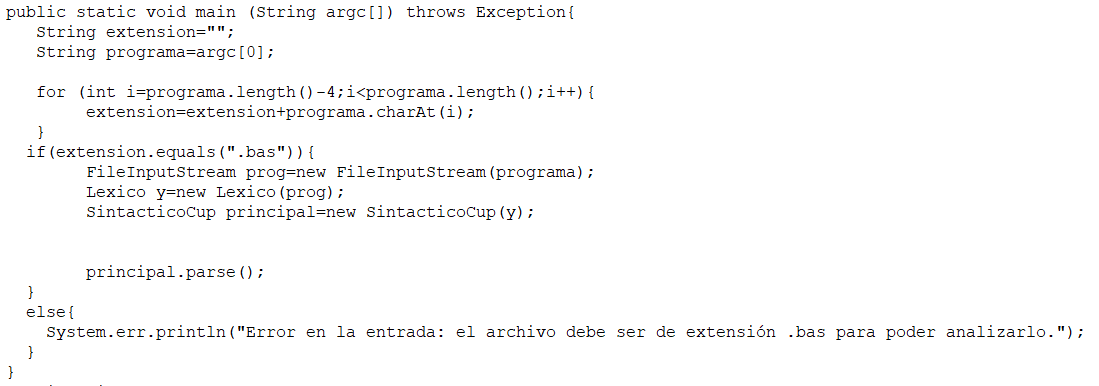




Nótese que el error sintáctico en la línea 10 ocurre porque, al haber un error léxico en el lexema “THENGOTO”, lo ignora, dejando al token 30 en una posición incorrecta. Sin embargo, los errores de espaciado no ignoran ningún token, dado que se entiende que es un problema de espaciado y no pretende provocar errores sintácticos.

## Comprobación de la extensión del fichero:

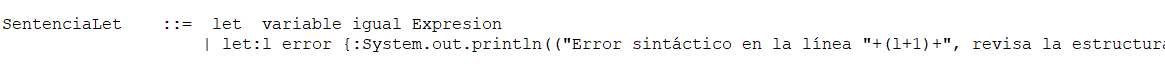
Otro requisito es que el analizador sintáctico no acepte como entrada ficheros con una extensión distinta a “.bas”. Para conseguir esto, simplemente se ha comprobado el argumento de entrada en el método Main:



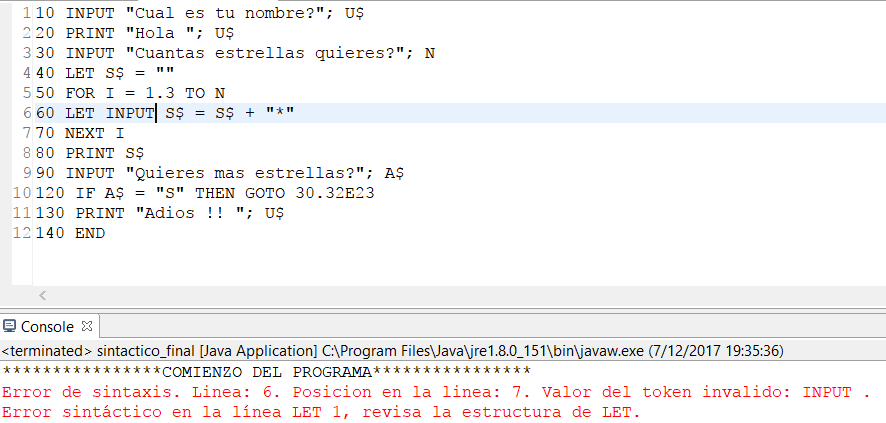
Como podemos observar, el programa sólo continuará si el argumento que pasamos como entrada (“argc[0]”) termina en “.bas”.

## Detección y recuperación de errores sintácticos:

Hemos utilizado el siguiente formato para tratar los errores de uso incorrecto de una estructura de palabra reservada:



De esta forma, si una sentencia comienza por dicha palabra reservada (en este caso LET) pero no cumple su sintaxis, nos indicará que hay un mal uso de la misma en la línea indicada. Podemos observar aquí un ejemplo de uso:



De igual forma, hemos ido implementando errores específicos a cada posible fallo de sintaxis que hemos tenido en cuenta.

## Árbol de derivación:

No se ha finalizado la realización del árbol de derivación por falta de tiempo. He creado la estructura de un árbol general con clases de Java, he creado el objeto árbol y el objeto nodo raíz al inicio del programa, pero no ha dado tiempo a crear el resto de nodos en todas las producciones de la gramática. Desde el punto actual, habría que crear un objeto nodo intermedio en el resto de producciones no terminales y un nodo hoja en los terminales, añadiendo todo al árbol ya creado. Una vez en ese punto, sólo habría que imprimir el árbol por pantalla.

# ANALIZADOR SEMÁNTICO

## Tabla de especificaciones semánticas:

|  |  |  |
| --- | --- | --- |
| ESe1 | Cumplido 🗹 | Implementación: Imagen 3.1  Demostración: ficheros 1, 2, 3 |
| ESe2 | Cumplido 🗹 | Implementación: Imagen 3.2  Demostración: ficheros 2, 3 |
| ESe3 | Cumplido 🗹 | Implementación: Imagen 3.2  Demostración: ficheros 1, 3 |
| ESe4 | Cumplido 🗹 | Implementación: Imagen 3.3  Demostración: ficheros 2, 3 |
| ESe5 | No corresponde a la fase de análisis |  |
| ESe6 | Incompleto ⌧ |  |
| ESe7 | Cumplido 🗹 | Implementación: Imagen 3.4  Demostración: ficheros 3 |
| ESe8 | Incompleto ⌧ |  |
| ESe9 | Cumplido 🗹 | Implementación: Imagen 3.5  Demostración: ficheros 3 |
| ESe10 | Cumplido 🗹 | Implementación: Imagen 3.6  Demostración: ficheros 3 |
| ESe11 | Cumplido 🗹 | Implementación: Imagen 3.7  Demostración: ficheros 2 |
| ESe12 | Cumplido 🗹 | Implementación: Imagen 3.8  Demostración: ficheros 1, 2, 3 |
| ESe13 | Cumplido 🗹 | Implementación: Imagen 3.9  Demostración: ficheros 1, 2, 3 |
| ESe14 | Cumplido 🗹 | Implementación: Imagen 3.10  Demostración: ficheros 3 |
| ESe15 | Incompleto ⌧ |  |
| ESe16 | Cumplido 🗹 | Implementación: Imagen 3.11  Demostración: ficheros 1, 2 |
| ESe17 | Cumplido 🗹 | Implementación: Imagen 3.12  Demostración: ficheros 1, 2 |

\*Nota: pese a que más ficheros cumplen los requisitos, se han redactado los tres primeros de forma que cumplan por sí mismos todos los requisitos.

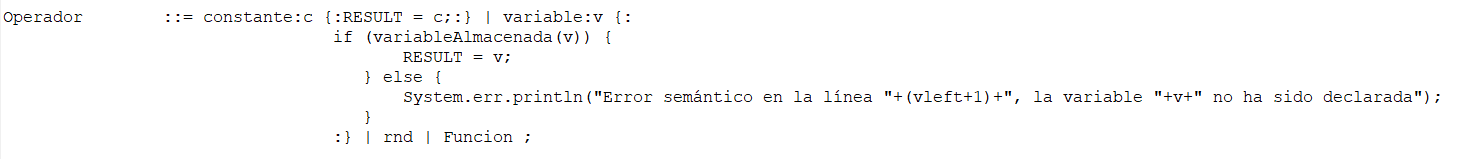


Imagen 3.1

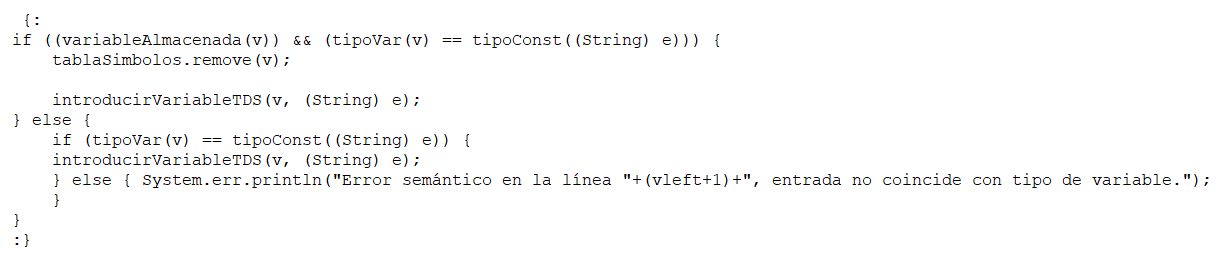


Imagen 3.2

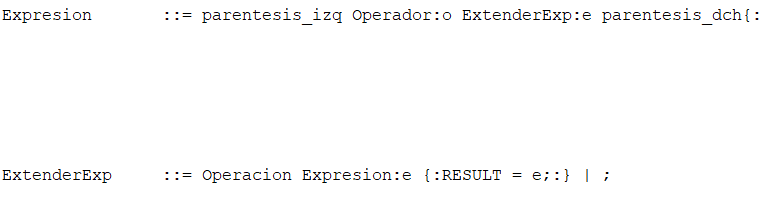


Imagen 3.3

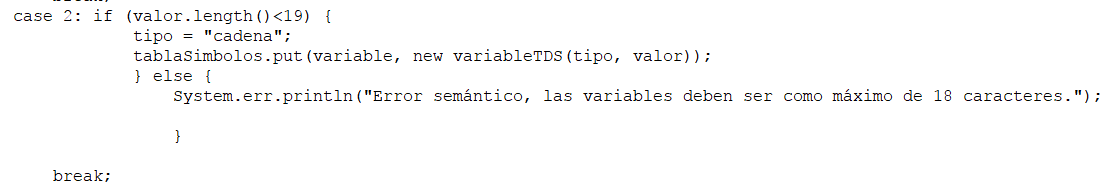


Imagen 3.4

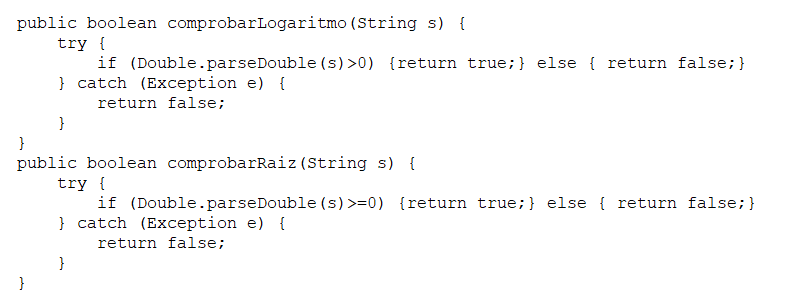


Imagen 3.5

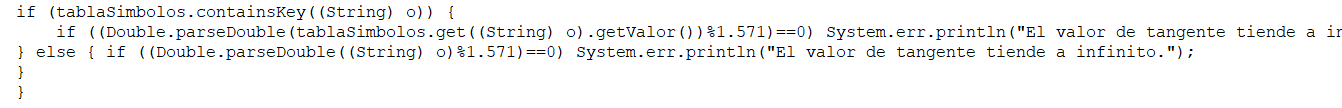


Imagen 3.6 – Nota: se toma 1.571 como el número de radianes equivalentes a 90º

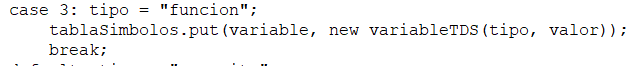


Imagen 3.7

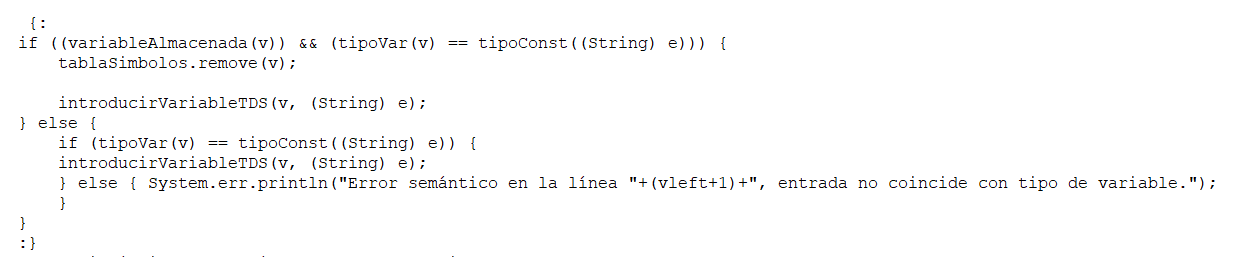


Imagen 3.8

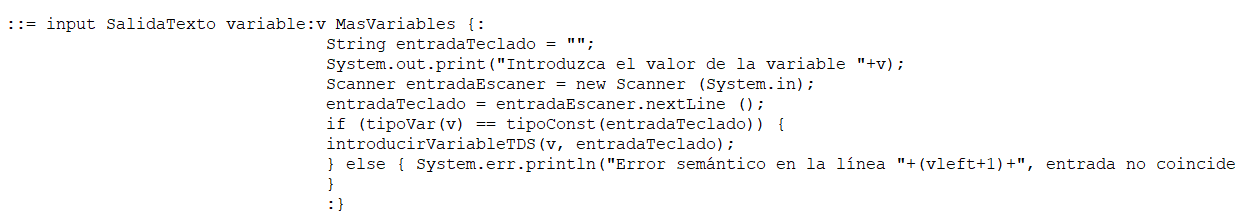


Imagen 3.9



Imagen 3.10

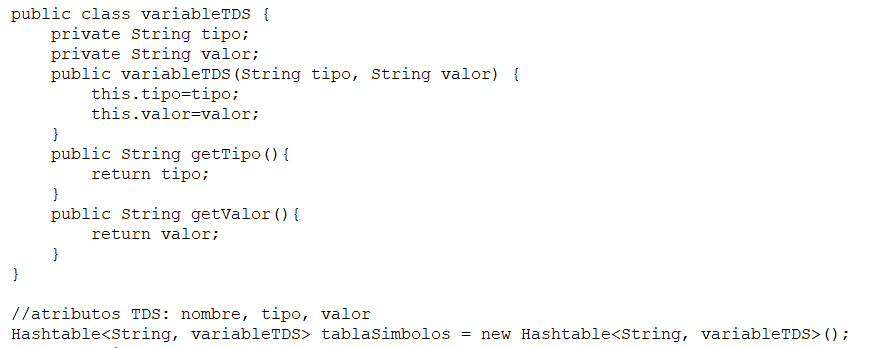


Imagen 3.11

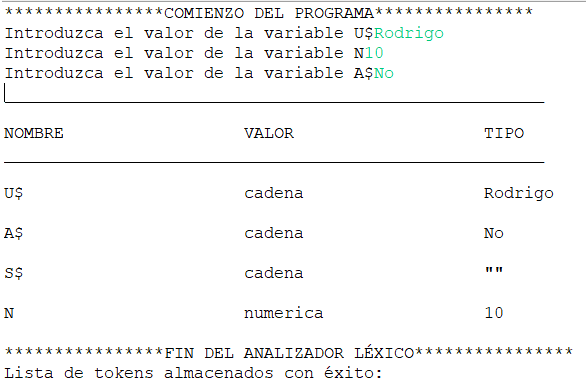


Imagen 3.12

## Tratamiento de errores semánticos:

Los errores semánticos se han tratado de la siguiente forma en general: almacenando el valor de los terminales de forma “terminal:t”, y utilizando el valor de “t” en el código colocado entre “{::}”. Con la sentencia “RESULT = t” se puede subir el valor de dicho terminal a niveles más cercanos a la raíz de la gramática.

Dada la cantidad de código escrito, explicaré las funciones externas (en el action code) en las que se apoya el código escrito en la gramática pero no dicho código, ya que tiene menos relevancia.

* lineaValida(): comprueba si una nueva entrada de línea es válida, comparando su valor con la última línea válida. Si se encuentra que el valor es igual o menor a la última línea correcta, notificará un error semántico indicando el error con su motivo y línea correspondientes.
* comprobarEntero(): comprueba si el valor de un String puede ser casteado a valor numérico. Esto nos sirve para determinar si una constante es numérica o de cadena, dado que si intentamos castear una cadena con la instrucción “Long.parseLong()” saltará una excepción que controlaremos.
* tipoVar: Para determinar el tipo de las variables he tenido en cuenta su longitud, dado que una variable de una sola letra determina una variable numérica simple, una variable con dos letras será siempre una letra continuada del símbolo de dólar, que determinará una variable de cadena, una variable de tres letras determinará siempre una función de la forma “FNX” y una variable con cuatro o más dígitos determinará una variable numérica suscrita.
* comprobarLogaritmo(): comprueba si el valor de un logaritmo es mayor que cero, retornando un error semántico en caso de ser un tipo inválido o un tipo numérico menor o igual que cero.
* comprobarRaiz(): realiza la misma función que el método anterior, sólo que en este caso si el valor es cero también será válido.
* esEntero(): determina si la entrada es de tipo numérica. Dado que puede ser una constante o una variable, si falla el casteo a tipo numérico sólo devolverá “true” si la entrada está en la tabla de símbolos y su tipo es numérico.
* finAnalisis(): método ejecutado al final del análisis, que imprime los datos de interés por pantalla.

## Implementación de la Tabla de Símbolos:

Para la implementación de la Tabla de Símbolos, o TDS en algunas partes del código, se ha utilizado una estructura HashTable. De esta forma, tendremos un atributo único que nos servirá de llave para encontrar su valor asociado. Como nos interesa tener varios datos asociados a cada llave, y podría en un futuro interesarnos más información de los mismos, he creado una clase variableTDS donde sus atributos almacenan el tipo de la variable (numérica, cadena, función o suscrita) y su valor.

Métodos relacionados:

* variableAlmacenada(): busca con la entrada como llave en la tabla si la variable ha sido declarada, devolviendo “true” en caso afirmativo o “false” en caso negativo.
* introducirVariableTDS(): nótese que no hace falta pedir como parámetro de esta función el atributo tipo, dado que como se ha explicado anteriormente dependiendo de la longitud de la variable podemos determinar esta información.

# ANEXO – CASOS DE PRUEBA

# Prueba 1 – Correcto con toda la información.

### Fichero: prueba1.bas

10 INPUT "Cual es tu nombre?"; U$

20 PRINT "Hola "; U$

30 INPUT "Cuantas estrellas quieres?"; N

40 LET S$ = ""

50 FOR I = 1 TO N

60 REM LET S$ = S$ + "\*"

70 NEXT I

80 PRINT S$

90 INPUT "Quieres mas estrellas?"; A$

120 IF A$ = "S" THEN GOTO 30

130 PRINT "Adios !! "; U$

140 END

### Salida:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*COMIENZO DEL PROGRAMA\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Introduzca el valor de la variable U$Rodrigo

Introduzca el valor de la variable N5

Introduzca el valor de la variable A$No

\*\*\*\*\*\*\*\*\*\*\*\*\*\*FIN DEL ANALIZADOR SINTÁCTICO\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

NOMBRE VALOR TIPO

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

U$ cadena Rodrigo

A$ cadena No

S$ cadena ""

N numerica 5

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*FIN DEL ANALIZADOR LÉXICO\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Lista de tokens almacenados con éxito:

[entrada(10),INPUT ,constante("Cual es tu nombre?"),punto\_coma,

variable(U$),entrada(20),PRINT ,constante("Hola "),punto\_coma,

variable(U$),entrada(30),INPUT ,constante("Cuantas estrellas quieres?"),punto\_coma,

variable(N),entrada(40),LET ,variable(S$),igual,

constante(""),entrada(50),FOR ,variable(I),igual,

constante(1),TO ,variable(N),entrada(60),REM LET S$ = S$ + "\*",

entrada(70),NEXT ,variable(I),entrada(80),PRINT ,

variable(S$),entrada(90),INPUT ,constante("Quieres mas estrellas?"),punto\_coma,

variable(A$),entrada(120),IF ,variable(A$),igual,

constante("S"),THEN ,GOTO ,constante(30),entrada(130),

PRINT ,constante("Adios !! "),punto\_coma,variable(U$),entrada(140),

END,]

No hay errores léxicos.

## Prueba 2 – Correcto.

### Fichero: prueba2.bas

10 INPUT "Introduzca el primer operador"; N

20 INPUT "Introduzca el segundo operador"; M

30 PRINT "Multiplicacion: "; M\*N

40 LET M = 5

50 REM ESTO ES UN COMENTARIO

60 LET V = 7

70 LET M = V

80 PRINT ABS(1); V

90 DEF FNX (M) = M \* V

100 DATA 100, 50

110 READ M, V

120 RESTORE

130 DIM V(10)

140 RANDOMIZE

150 END

## Prueba 3 – Incorrecto.

### Fichero prueba3.bas

10 INPUT "Introduzca el primer operador"; N$

20 INPUT "Introduzca el segundo operador"; M

25 INPUT "Introduzca"; R

30 PRINT "Multiplicacion: "; M\*N

35 IF M>R THEN 20

40 LET U = 3.43E52

20 LET M = 5

70 LET M = "Hola"

80 LET N$ = LOG(0)

90 PRINT cadenainvalida; VVV

100 LET N$ = "1234567890123456789"

130 DIM V(10)

140 RANDOMIZE

150 END

### Salida:

Introduzca el valor de la variable N$prueba

Introduzca el valor de la variable M42

Introduzca el valor de la variable R3

Error semántico en la línea 4, la variable N no ha sido declarada

El indicador de línea 20 en la línea 7 no es secuencial.

Error semántico en la línea 8, entrada no coincide con tipo de variable.

Error semántico en línea 9, la función logaritmo no acepta números menores o iguales a 0

Error semántico en la línea 9, entrada no coincide con tipo de variable.

Error de sintaxis. Linea: 10. Posicion en la linea: 23. Valor del token invalido: ;.

Error sintáctico en la línea PRINT 1, revisa la estructura de PRINT.

Error semántico, las variables deben ser como máximo de 18 caracteres.

Error semántico en la línea 12, variable V no declarada.

Hemos encontrado 2 errores léxicos.

[Se ha detectado un error léxico, no se reconoce la palabra: cadenainvalida en la línea: 7. Se ha detectado un error léxico, no se reconoce la palabra: VVV en la línea: 7. ]

## Prueba 4 – Incorrecto.

### Fichero: prueba4.bas

10 PRINT "Imprimir A"; A

20 LET A = 4

30 PRINT "Imprimir A"; A

40 LET A = "Hola"

50 END

### Salida:

Error semántico en la línea 1, la variable A no ha sido declarada

Error semántico en la línea 1, variable "Imprimir A" no declarada.

Error semántico en la línea 4, entrada no coincide con tipo de variable.

No hay errores léxicos.

## Prueba 5 – Incorrecto.

### Fichero prueba5.bas

10 INPUT "Imprimir A"; A

20 LET S$ = "Hola"

30 LET A = S$

40 IF A>5 THEN GOTO 30.40

50 END

### Salida:

Introduzca el valor de la variable A5

Error semántico en la línea 3, entrada no coincide con tipo de variable.

Error semántico, la línea referida en la línea 4 debe ser un entero positivo.

No hay errores léxicos.

## Prueba 6 – Incorrecto.

### Fichero prueba6.bas

10 INPUT "Imprimir A"; A

20 LET S$ = "Hola"

30 LET A = 23.23E542

40 IF A>5 THEN GOTO 30

50 END

60 INPUT ERRORLEXICO

### Salida:

Error de sintaxis. Linea: 5. Posicion en la linea: 0. Valor del token invalido: 60.

Hemos encontrado 1 errores léxicos.

[Se ha detectado un error léxico, no se reconoce la palabra: ERRORLEXICO en la línea: 6. ]

## Prueba 7 – Incorrecto.

### Fichero prueba7.bas

10 INPUT "Imprimir A"; A

20LET S$ = "Hola"

30 LET A = 23.23E542

40 IF A>5 THENGOTO 30

50 END

### Salida:

Introduzca el valor de la variable A5

## La palabra reservada LET debe estar entre espacios.

Error de sintaxis. Linea: 4. Posicion en la linea: 19. Valor del token invalido: 30

Hemos encontrado 1 errores léxicos.

[Se ha detectado un error léxico, no se reconoce la palabra: THENGOTO en la línea: 4. ]

### \*notar cómo se recupera del error léxico en la palabra LET y acepta la estructura sintáctica pese a señalar el error.

## Prueba 8 – Incorrecto.

### Fichero prueba8.bas

10 LET C$ = "password"

20 INPUT "Introduzca la contrasea"; S$

30 IF C$ <> S$ THEN GOTO 20

40 PRINT "Enhorabuena, era "; C$

50 LET S$ = N$

50 LET S$ = "asignacion"

60 END

### Salida:

Introduzca el valor de la variable S$password

Error semántico en la línea 5, la variable N$ no ha sido declarada

Error semántico en la línea 5, entrada no coincide con tipo de variable.

El indicador de línea 50 en la línea 6 no es secuencial.

No hay errores léxicos.

## Prueba 9 – Incorrecto.

### Fichero prueba9.bas

10 LET C$ = "yanosemeocurrenmasejemplosestacadenaesdemasiadolarga"

20 PRINT "Era demasiado larga"; C$

30 INPUT "Introducela a mano "; C$

40 PRINT "Esto es una typo;" C$

50 PRINT "Esto ya esta bien"; C$

60 END

### Salida:

Error semántico, las variables deben ser como máximo de 18 caracteres.

Error semántico en la línea 2, la variable C$ no ha sido declarada

Error semántico en la línea 2, variable "Era demasiado larga" no declarada.

Introduzca el valor de la variable C$cadenacorta

Error de sintaxis. Linea: 4. Posicion en la linea: 29. Valor del token invalido: C$.

No hay errores léxicos.

## Prueba 10 – Incorrecto.

### Fichero prueba10.bas

10 INPUT "Por fin acabamos con los ejemplos, introduce la nota"; N

20 IF N<5 THEN GOTO 10

30 PRINT "Enhorabuena, has aprobado, tu nota es "; N

40 LET I = 1

50 REM esto es un comentario aleatorio para rellenar

60 FOR I = 1 TO N

70 PRINT "De verdad, no se que mas hacer, aqui hay un error por no declarar"; S$

80 NEXT I

90 END

### Salida:

Introduzca el valor de la variable N9

Error semántico en la línea 7, la variable S$ no ha sido declarada