



Universidad  
de Alcalá

## PROCESADORES DEL LENGUAJE

**Analizador léxico, sintáctico y semántico.**

GUILLERMO GÓMEZ OLIVARES

Grado en Ingeniería Informática

# INDICE

<b>RESUMEN</b> .....	2
1. Introducción.....	2
2. Diseño del Analizador léxico .....	2
2.1 Métodos .....	2
2.1.1 Teoría del Analizador .....	2
2.1.2 Creación de un proyecto en JFLEX .....	3
2.1.3 Implementación y desarrollo de la solución .....	3
2.2 Análisis y resultados obtenidos.....	12
3. Diseño del analizador sintáctico .....	14
3.1 Métodos .....	14
3.1.1 Teoría del analizador.....	14
3.1.2 Creación de un proyecto en CUP .....	14
3.1.3 Implementación y desarrollo de la solución .....	<b>¡Error! Marcador no definido.</b>
3.2 Tratamiento de errores .....	20
4. Diseño del analizador semántico .....	22
4.1 Métodos .....	22
4.1.1 Teoría del analizador.....	22
4.1.2 Análisis y resultados obtenidos según su implementación (especificaciones Semánticas).....	22
5. Conclusiones .....	29
6. Referencias .....	29
7. Apéndice .....	29
<b>Fichero 1:</b> .....	30
<b>Fichero 2:</b> .....	31
<b>Fichero 3:</b> .....	32
<b>Fichero 4:</b> .....	33
<b>Fichero 5:</b> .....	34
<b>Fichero 6:</b> .....	35
<b>Fichero 7:</b> .....	36
<b>Fichero 8:</b> .....	37
<b>Fichero 9:</b> .....	38
<b>Fichero 10:</b> .....	39

## RESUMEN.

La práctica consiste en la realización de un analizador léxico, sintáctico y semántico, mediante la utilización de un framework llamado Jflex que se utiliza para analizar la parte léxica de nuestros programas esta librería usa archivos con la extensión lex, por ejemplo: procesadores.lex, por otro lado también incorpora librerías para el análisis sintáctico y semántico con la finalidad de reconocer el lenguaje propuesto que se diferencia porque la extensión es cup, por ejemplo: procesadores.cup.

Antes de nada, es necesario tener en cuenta ciertas pautas para empezar a realizar el analizador:

1. Primero tuvimos que desarrollar el analizador léxico, es el encargado de leer letra a letra nuestro archivo y verificar si cumple las condiciones, este proceso genera el token por cada uno de los elementos reconocidos o el error en caso de no reconocerlo.
2. La segunda parte del desarrollo es la implementación de un analizador sintáctico, que es la fase encargada de analizar la estructura de las expresiones en base a gramáticas generadas por el analizador léxico y de determinar si una estructura está correctamente formada.
3. Y el último la comprobación semántica de los resultados finales, comprobar que los argumentos introducidos son válidos para un determinado token y son compatibles entre sí.

En este trabajo se desarrollan todas y cada una de las partes que se involucran en un analizador.

## 1. Introducción

Esta práctica tiene como objetivo tener una visión sobre los conocimientos vistos en clase por eso es complementaria, he podido observar cada uno de los elementos de los que se compone un compilador desde el análisis hasta el tratamiento de errores. La práctica consiste en la realización de un analizador léxico, sintáctico y semántico, para ello utilizamos las herramientas propuestas en clase que son la utilización de archivos lex y cup, estos se encargarán de reconocer el lenguaje propuesto.

Para la realización de esta práctica vamos a usar el IDE eclipse con el plugin instalado para reconocer lex y cup, como éstas están en versión Beta para tener todo más claro me apoyo en el IDE Visual Studio Code, que se puede predeterminar el lenguaje, ya que me ayuda a ver todo más claro y más limpio.

## 2. Diseño del Analizador léxico

### 2.1 Métodos

#### 2.1.1 Teoría del Analizador

Es la primera fase de un compilador consistente en un programa que recibe como entrada el código fuente de otro programa (secuencia de caracteres) y produce una salida compuesta de tokens (componentes léxicos) o símbolos. Estos tokens sirven para una posterior etapa del proceso de traducción, siendo la entrada para el analizador sintáctico.

La especificación de un lenguaje de programación a menudo incluye un conjunto de reglas que definen el léxico. Estas reglas consisten comúnmente en expresiones regulares que indican el conjunto de posibles secuencias de caracteres que definen un token o lexema.

En algunos lenguajes de programación es necesario establecer patrones para caracteres especiales (como el espacio en blanco) que la gramática pueda reconocer sin que constituya un token en sí. Las tareas principales de este son:

### 2.1.2 Creación de un proyecto en JFLEX

El fichero de entrada de JFlex está compuesto de tres secciones, separadas por una línea donde aparece únicamente un '%%'.

Esta sección se utiliza para declarar rutinas que usa la librería de JFLEX tienes varias opciones que se pueden configurar y normalmente muchas son opcionales por ejemplo para poder enlazar con el cup se usa la rutina “%cup”; Si se omite, el segundo `%%' en el fichero de entrada se podría omitir también. La sección de reglas son las que definen las directivas y controlan el comportamiento que debe tener JFlex, macros y nombres de estados.

La sección de Definiciones contiene declaraciones de definiciones de nombres sencillas para simplificar la especificación del escáner, y declaraciones de condiciones de arranque. Para poder enlazar con el analizador sintáctico hace falta importar algunas funcionalidades de nuestro framework como pueden ser **java\_cup.runtime.\*** para que el analizador léxico le proporcione un camino al sintáctico para iniciar la lectura de tokens.

### 2.1.3 Implementación y desarrollo de la solución

Se declaran las opciones que tendrá el archivo lex, como puede ser la importación de componentes Java, si vamos a usar un paquete para generar el analizar y a continuación los parámetros usados por el JFLEX para la gestión de errores, enlazar con el cup y versiones de teclado entre otras.

```

1  package PECL1;
2
3  import java_cup.runtime.*;
4  import java.io.*;
5  import java.util.*;
6  import java.util.ArrayList;
7
8  %%
9
10 | %class Analizador_LexicoLex //Definicion de nombre de la clase java
11 | %unicode
12 | %line           // Detector de lineas
13 | %column         // Detector de columnas
14 | %standalone
15 | %cup            //Enlazador con CUP
16
17

```

FIG1. OPCIONES Y DECLARACIONES

En la parte del código es donde se gestionará la parte en java. En esta parte declaramos las variables que vamos a usar en nuestro caso tenemos una variable de tipo int que hará de contador para los errores y otra de tipo ArrayList para guardar los tokens que se reconocerán.

```

//Codigo
%{
    private static ArrayList<String> tokensList = new ArrayList<String>();
    int err=0;

```

FIG. 2. REPRESENTACIÓN DEL CÓDIGO

### Palabras reservadas

Para el tratamiento de las palabras reservadas de su uso queda restringido y su expresión regular será la siguiente (En la práctica no se ha desarrollado así, es para simplificar la expresión regular):

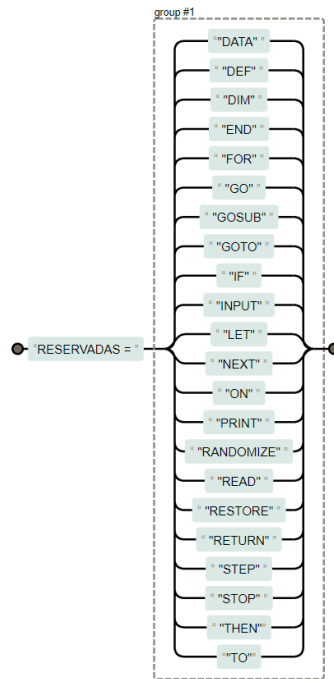


FIG. 3. AFD DE PALABRAS RESERVADAS

### Identificadores

Diferenciamos tres tipos de variables para nuestro programa, los identificadores de cadena (fig 4.1) que se componen de una letra y el símbolo \$, los identificadores numéricos simples (fig 4.2) que se componen de una letra simple o de una letra y un número y los identificadores de variables suscritas (fig 4.3) que se componen de una letra y un número o una expresión entre paréntesis, por otro lado podemos también declarar funciones que están compuestas por las letras FNx donde x es cualquier letra también puede contener parámetros entre paréntesis (fig 4.4).

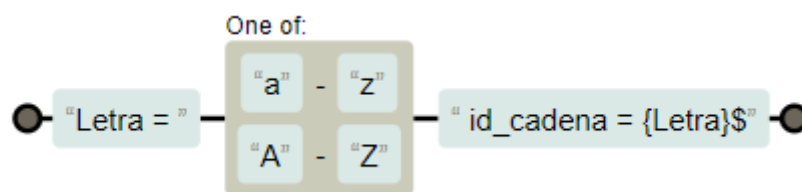


FIG. 4.1. AFD DE IDENTIFICADORES

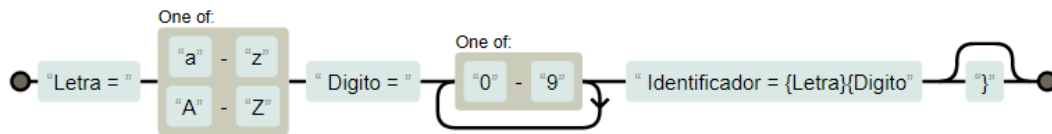


FIG. 4.2. AFD DE IDENTIFICADORES

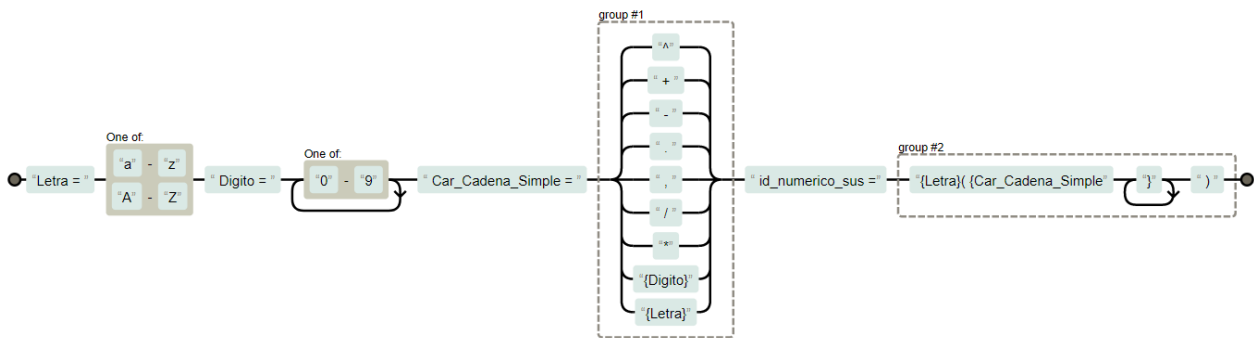


FIG. 4.3. AFD DE IDENTIFICADORES

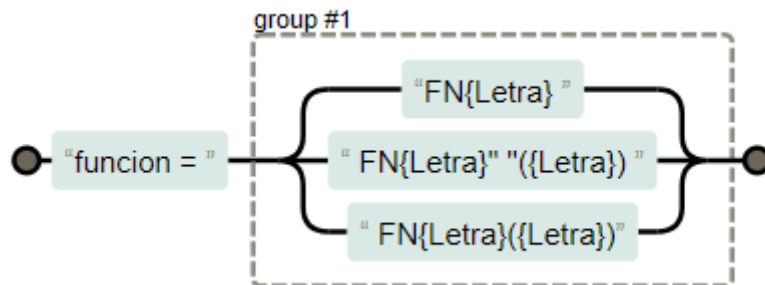


FIG. 4.4. AFD DE IDENTIFICADOR DE FUNCIÓN

### Operador de asignación

Para el operador asignación "=" su expresión regular será la siguiente (En la práctica no se ha desarrollado así, es para simplificar la expresión regular):

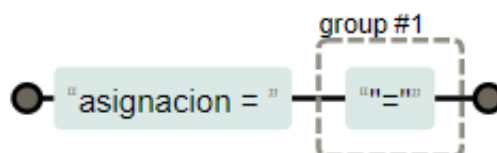


FIG. 5. AFD ASIGNACIÓN

### Operadores de relación

Su AFD sería el siguiente (En la práctica no se ha desarrollado así, es para simplificar la expresión regular):

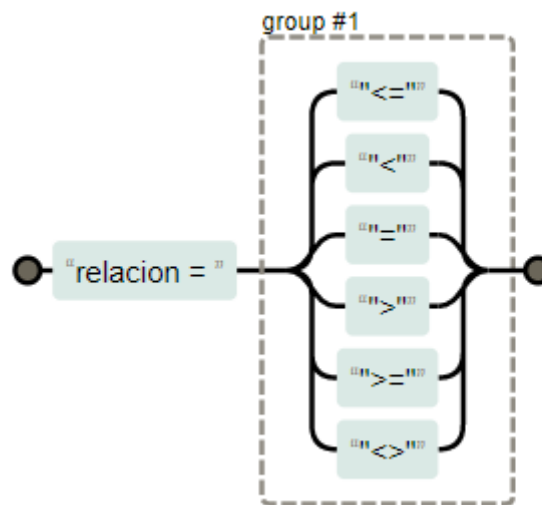


FIG. 6. AFD RELACIÓN

### Operadores

Para los operadores matemáticos será el siguiente AFD:

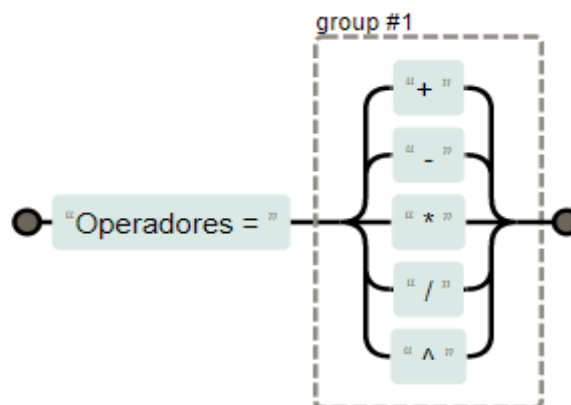


FIG. 7. AFD OPERADORES MATEMÁTICOS

### Constantes

Existen dos tipos de constantes numéricas (Fig. 8.1 y Fig. 8.2) y constantes de cadena (Fig. 8.3):

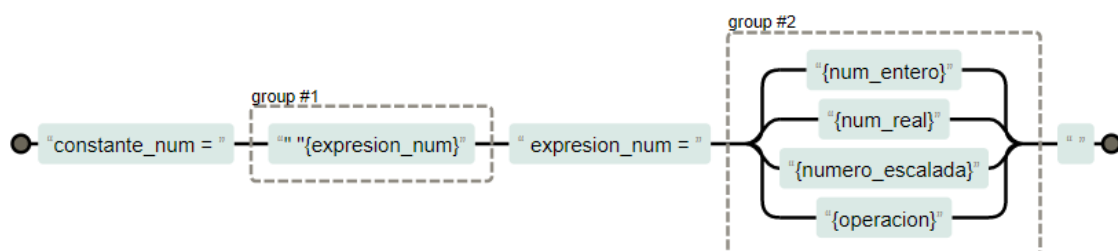


FIG. 8.1. AFD CONSTANTE NUMÉRICA 1



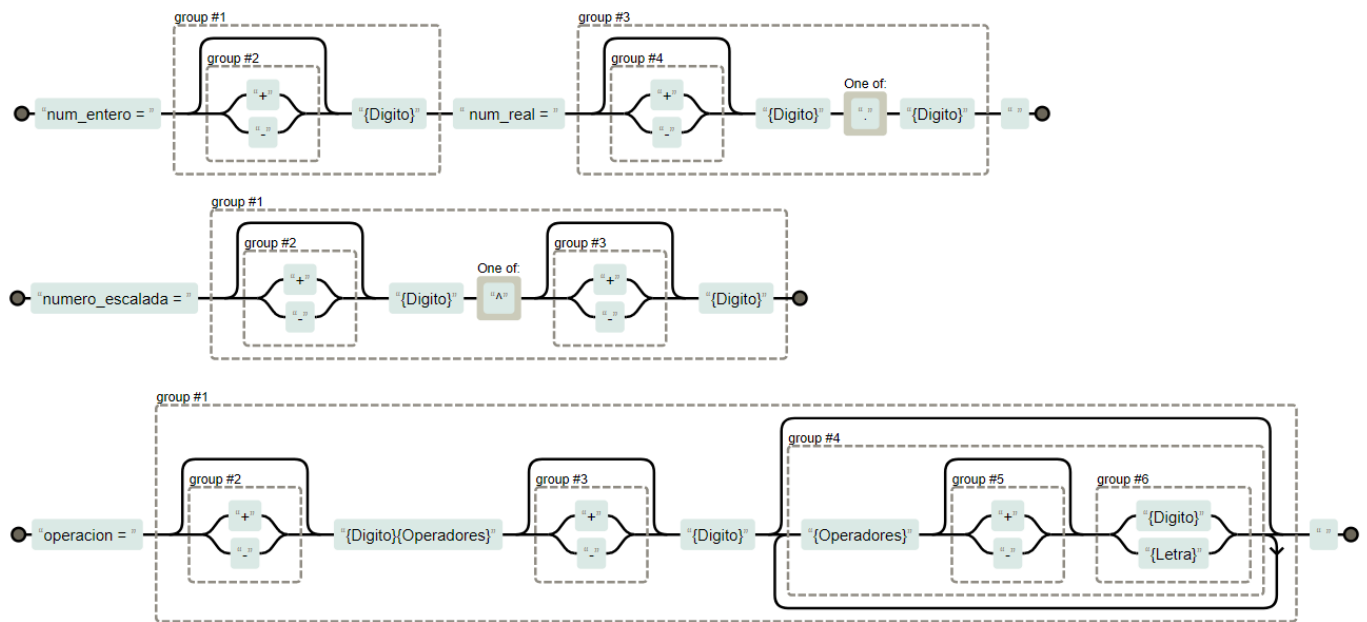


FIG. 8.2. AFD CONSTANTE NUMÉRICA 2

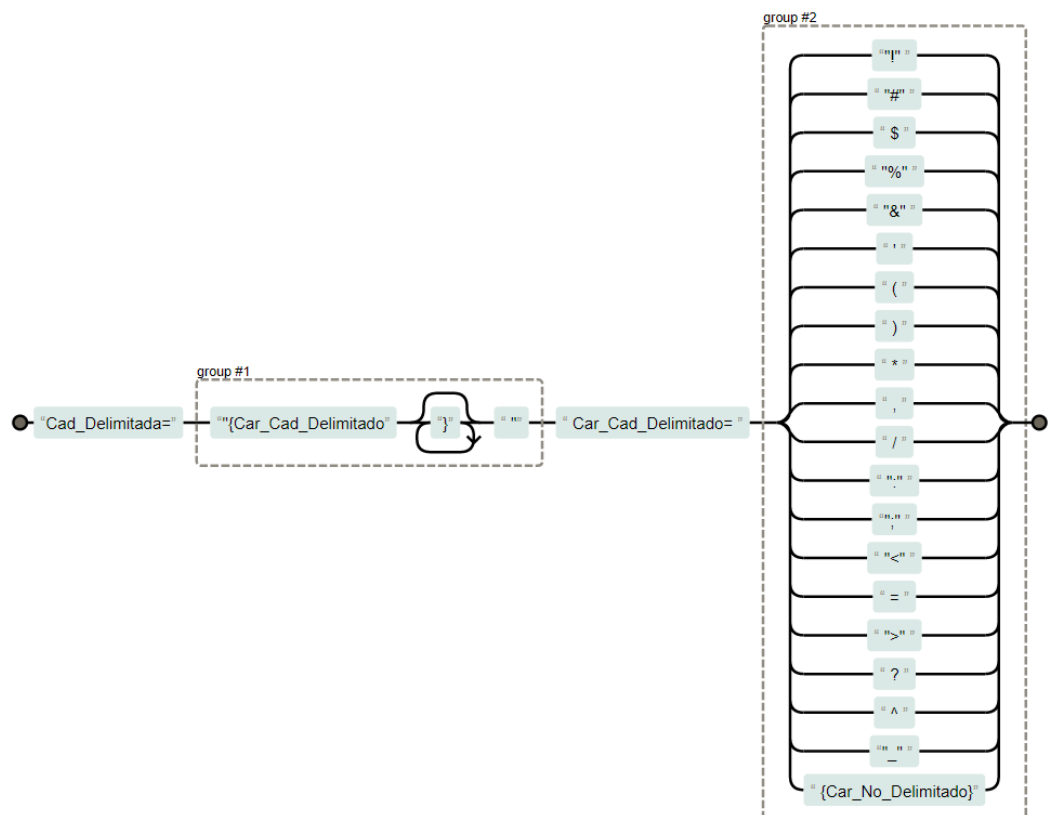


FIG. 8.3. AFD CONSTANTE CADENA

## Funciones matemáticas

Expresión regular para identificar las palabras reservadas para operaciones matemáticas:

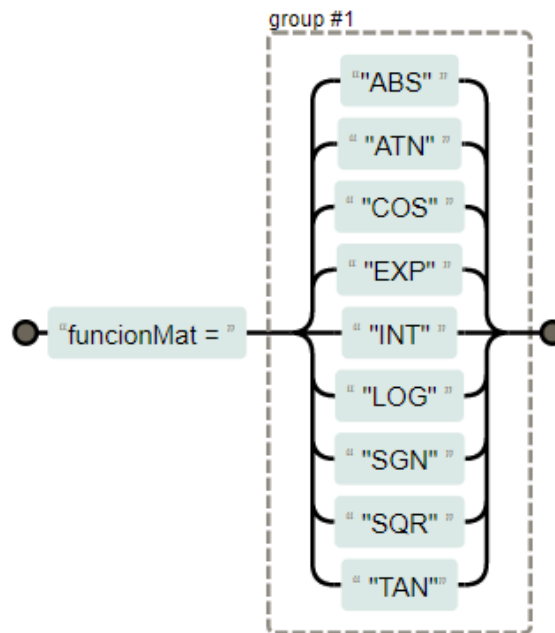


FIG. 9. AFD FUNCIONES MATEMÁTICAS.

## Comentario

Expresión regular que determina si hay un comentario en la línea:

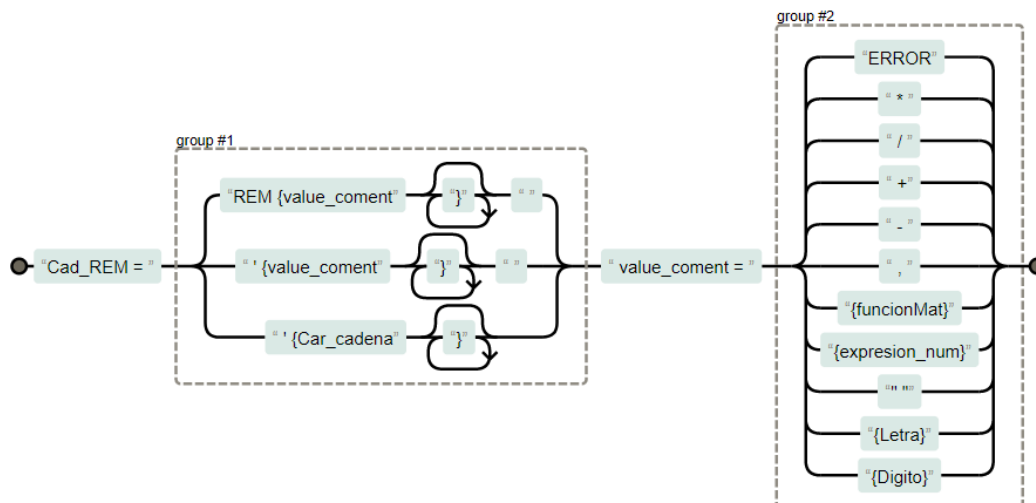


FIG. 10. AFD COMENTARIO

## Errores

Los errores vienen dados por una serie de caracteres no válidos, para el tratamiento de errores su AFD será la siguiente:

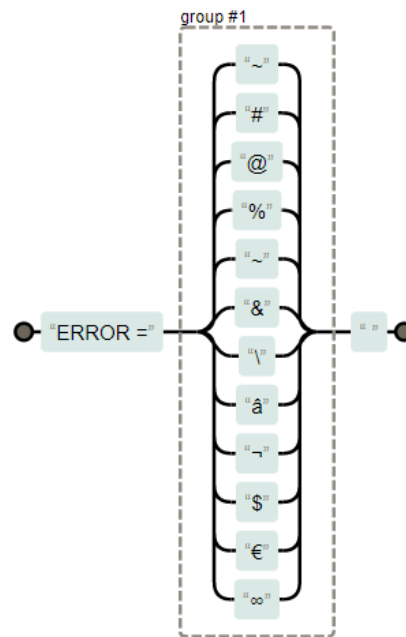


FIG. 11. AFD ERROR

## Acciones:

Como se menciona anteriormente hemos definido una estructura de control de los tokens devueltos por el analizador léxico, en la que utilizaremos un ArrayList que al final de la ejecución imprimimos por pantalla, se define de la siguiente manera:

```
private static ArrayList<String> tokensList = new ArrayList<String>();
```

junto a esta definición también contamos con la definición de un contador de errores:

```
int err=0;
```

```

public int getErr(){
    return err;
}
/* Metodo de escribir en el fichero de salida y por pantalla*/
public void writeOutputFile() throws IOException {
    String filename = "salida.out";
    BufferedWriter out = new BufferedWriter(new FileWriter(filename));
    System.out.print("[");
    out.write("[");

    for (String s : this.tokensList) {
        if(s=="CR LF" || s=="LF"){
            System.out.print(s+",\n");
            out.write(s + ",\n");
        }
        else{
            System.out.print(s+",");
            out.write(s + ",");
        }
    }
    System.out.print("EOF]");
    out.write("EOF]");
    out.close();
}

```

FIG. 12. FUNCIÓN PARA IMPRIMIR Y DEVOLVER ERRORES.

Las declaraciones de la gramática para cada token leído se irán introduciendo en nuestro variable tokenList para luego poder imprimir por pantalla toda la lectura, a continuación de añadir al array devolvemos al analizador sintáctico un token con su valor para que lo pueda reconocer y comprobar si la gramática es correcta, aquí tenemos un ejemplo de todas las macros definidas en mi analizador:

```

"DATA" {this.tokensList.add(yytext()); return new Symbol(Analizador_SintacticoSym.DATA,yyline,yycolumn);}
"DEF" {this.tokensList.add(yytext()); return new Symbol(Analizador_SintacticoSym.DEF,yyline,yycolumn);}
"DIM" {this.tokensList.add(yytext()); return new Symbol(Analizador_SintacticoSym.DIM,yyline,yycolumn);}
"END" {this.tokensList.add(yytext()); return new Symbol(Analizador_SintacticoSym.END,yyline,yycolumn);}
"FOR" {this.tokensList.add(yytext()); return new Symbol(Analizador_SintacticoSym.FOR,yyline,yycolumn);}
"GO" {this.tokensList.add(yytext()); return new Symbol(Analizador_SintacticoSym.GO,yyline,yycolumn);}
"GOSUB" {this.tokensList.add(yytext()); return new Symbol(Analizador_SintacticoSym.GOSUB,yyline,yycolumn);}
"GOTO" {this.tokensList.add(yytext()); return new Symbol(Analizador_SintacticoSym.GOTO,yyline,yycolumn);}
"IF" {this.tokensList.add(yytext()); return new Symbol(Analizador_SintacticoSym.IF,yyline,yycolumn);}
"INPUT" {this.tokensList.add(yytext()); return new Symbol(Analizador_SintacticoSym.INPUT,yyline,yycolumn);}
"LET" {this.tokensList.add(yytext()); return new Symbol(Analizador_SintacticoSym.LET,yyline,yycolumn);}
"NEXT" {this.tokensList.add(yytext()); return new Symbol(Analizador_SintacticoSym.NEXT,yyline,yycolumn);}
"ON" {this.tokensList.add(yytext()); return new Symbol(Analizador_SintacticoSym.ON,yyline,yycolumn);}
"PRINT" {this.tokensList.add(yytext()); return new Symbol(Analizador_SintacticoSym.PRINT,yyline,yycolumn);}
"RANDOMIZE" {this.tokensList.add(yytext()); return new Symbol(Analizador_SintacticoSym.RANDOMIZE,yyline,yycolumn);}
"READ" {this.tokensList.add(yytext()); return new Symbol(Analizador_SintacticoSym.READ,yyline,yycolumn);}
"RESTORE" {this.tokensList.add(yytext()); return new Symbol(Analizador_SintacticoSym.RESTORE,yyline,yycolumn);}
"RETURN" {this.tokensList.add(yytext()); return new Symbol(Analizador_SintacticoSym.RETURN,yyline,yycolumn);}
"STEP" {this.tokensList.add(yytext()); return new Symbol(Analizador_SintacticoSym.STEP,yyline,yycolumn);}
"STOP" {this.tokensList.add(yytext()); return new Symbol(Analizador_SintacticoSym.STOP,yyline,yycolumn);}
"THEN" {this.tokensList.add(yytext()); return new Symbol(Analizador_SintacticoSym.THEN,yyline,yycolumn);}
"TO" {this.tokensList.add(yytext()); return new Symbol(Analizador_SintacticoSym.TO,yyline,yycolumn);}
"RND" {this.tokensList.add(yytext()); return new Symbol(Analizador_SintacticoSym.RND,yyline,yycolumn);}
{funcionMat} {this.tokensList.add(yytext()); return new Symbol(Analizador_SintacticoSym.FUNCION_MAT,yyline,yycolumn, yytext());}
"=" {this.tokensList.add("igual"); return new Symbol(Analizador_SintacticoSym.IGUAL,yyline,yycolumn);}
"+" {this.tokensList.add("mas"); return new Symbol(Analizador_SintacticoSym.MAS,yyline,yycolumn);}
"-" {this.tokensList.add("menos"); return new Symbol(Analizador_SintacticoSym.MENOS,yyline,yycolumn);}
"*" {this.tokensList.add("mul"); return new Symbol(Analizador_SintacticoSym.MUL,yyline,yycolumn);}
"/" {this.tokensList.add("div"); return new Symbol(Analizador_SintacticoSym.DIVISION,yyline,yycolumn);}
"^" {this.tokensList.add("exp"); return new Symbol(Analizador_SintacticoSym.EXPONENTE,yyline,yycolumn);}
"(" {this.tokensList.add("PARENTESIS_IZQ");return new Symbol(Analizador_SintacticoSym.PARENTESIS_IZQ,yyline,yycolumn);}
")" {this.tokensList.add("PARENTESIS_DER");return new Symbol(Analizador_SintacticoSym.PARENTESIS_DER,yyline,yycolumn);}
"," {this.tokensList.add("coma");return new Symbol(Analizador_SintacticoSym.COMA,yyline,yycolumn);}
"<=" {this.tokensList.add("menor_igual"); return new Symbol(Analizador_SintacticoSym.MENOR_IGUAL,yyline,yycolumn);}
">=" {this.tokensList.add("mayor_igual"); return new Symbol(Analizador_SintacticoSym.MAYOR_IGUAL,yyline,yycolumn);}
"<" {this.tokensList.add("menor"); return new Symbol(Analizador_SintacticoSym.MENOR,yyline,yycolumn);}
">" {this.tokensList.add("mayor"); return new Symbol(Analizador_SintacticoSym.MAYOR,yyline,yycolumn);}
"<>" {this.tokensList.add("distinto"); return new Symbol(Analizador_SintacticoSym.DISTINTO,yyline,yycolumn);}
{Digito} {this.tokensList.add("ent(" + yytext() + ")"); return new Symbol(Analizador_SintacticoSym.ENTERO,yyline,yycolumn, yytext());}
{id_cadena} {this.tokensList.add("ide(" + yytext() + ")"); return new Symbol(Analizador_SintacticoSym.ID_CADENA,yyline,yycolumn, yytext());}
{id_numerico_sus} {this.tokensList.add("ide(" + yytext() + ")"); return new Symbol(Analizador_SintacticoSym.ID_NUM_SUS,yyline,yycolumn, yytext());}
{Identificador} {this.tokensList.add("id(" + yytext() + ")"); return new Symbol(Analizador_SintacticoSym.ID_NUM,yyline,yycolumn, yytext());}
{Cad_REM} {this.tokensList.add("value_coment: " + yytext()); return new Symbol(Analizador_SintacticoSym.REM,yyline,yycolumn, yytext());}
{Cad_Delimitada} {
    this.tokensList.add("const(" + yytext().substring(1,yytext().length()-1) + ")");
    return new Symbol(Analizador_SintacticoSym.CONSTANTE,yyline,yycolumn, yytext().substring(1,yytext().length()-1));
}
{CRLF} {this.tokensList.add("CRLF"); return new Symbol(Analizador_SintacticoSym.CRLF,yyline,yycolumn);}
{CR} {this.tokensList.add("CR");}
{LF} {this.tokensList.add("LF");}
{PUNTOYCOMA} {
    this.tokensList.add("PUNTOYCOMA");
    return new Symbol(Analizador_SintacticoSym.PUNTOYCOMA,yyline,yycolumn);
}
{constante} {
    this.tokensList.add("const_num(" + yytext().substring(1)+ ")");
    return new Symbol(Analizador_SintacticoSym.CONSTANTE_NUM,yyline,yycolumn, yytext().substring(1));
}
{funcion} {
    this.tokensList.add(yytext());
    return new Symbol(Analizador_SintacticoSym.FUNCION,yyline,yycolumn, yytext());
}
{ERROR} {System.err.println("Error lexico en línea " + (yyline+1) + " columna " + (yycolumn+1) + ": " + yytext());System.out.println();}
<<EOF>> {return new Symbol(Analizador_SintacticoSym.EOF,yyline,yycolumn);}
. /* Regla lexica para evitar estrellarse cuando se encuentra algo extraño, no hacemos nada tampoco */

```

FIG. 13. LISTADO DE MACROS Y SÍMBOLOS

Para cada regla léxica tomamos algunas acciones muy similares. Vamos añadiendo el token a nuestra estructura ArrayList. Una de las partes más importantes de nuestro analizador es la expresión regular que define el error que nos imprimirá por pantalla en que línea y en columna se encuentra el fallo o el símbolo que no ha podido leer.

## 2.2 Análisis y resultados obtenidos

Los cambios en el fichero .lex en Eclipse generara una estructura del proyecto java como podemos observar en la siguiente figura.

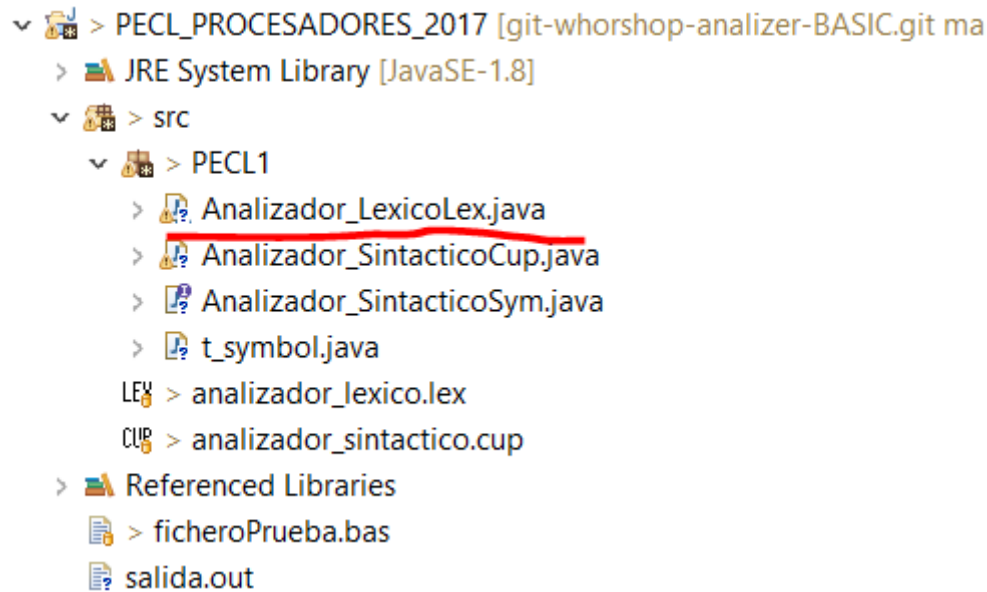


FIG. 14. ESTRUCTURA DEL PROYECTO

Ahora vamos a analizar el fichero expuesto a continuación y la salida producida por nuestro analizador en base a el.

```
110 INPUT "Cual es tu nombre?"; U$
220 PRINT "Hola "; U$
330 INPUT "Cuantas estrellas quieres?"; N
440 LET S$ = ""
550 FOR I = 1 TO N
660 LET S$ = S$ + "*"
770 NEXT (I) @
880 PRINT S$
990 INPUT "Quieres mas estrellas?"; A$
10120 IF A$ = "S" THEN GOTO 30 #
11130 PRINT "Adios !! "; U$
12140 END
```

FIG. 15. FICHERO DE PRUEBA

1) Lo primero que nos mostrara por pantalla serán los errores analizados por el analizador léxico:

```
Error lexico en linea 7 columna 13: @
Error lexico en linea 10 columna 30: #
Numero de errores: 2
```

FIG. 16. RESULTADOS DE ERROR

2) A continuación nos mostrara la cantidad de errores encontradas en el análisis léxico Fig16.  
3) Por último se nos mostrara la lista de los tokens que han sido analizados Fig17.

```
[ent(10),INPUT,const(Cual es tu nombre?),PUNTOYCOMA,ide(U$),CRLF,
ent(20),PRINT,const(Hola ),PUNTOYCOMA,ide(U$),CRLF,
ent(30),INPUT,const(Cuántas estrellas quieres?),PUNTOYCOMA,id(N),CRLF,
ent(40),LET,ide(S$),igual,const(),CRLF,
ent(50),FOR,id(I),igual,const_num(1),TO,id(N),CRLF,
ent(60),LET,ide(S$),igual,ide(S$),mas,const(*),CRLF,
ent(70),NEXT,PARENTESIS_IZQ,id(I),PARENTESIS_DER,CRLF,
ent(80),PRINT,ide(S$),CRLF,
ent(90),INPUT,const(Quieres mas estrellas?),PUNTOYCOMA,ide(A$),CRLF,
ent(120),IF,ide(A$),igual,const(S),THEN,GOTO,const_num(30),CRLF,
ent(130),PRINT,const(Adios !! ),PUNTOYCOMA,ide(U$),CRLF,
ent(140),END,EOF]
```

FIG. 17. ANÁLISIS LÉXICO

## 3. Diseño del analizador sintáctico

### 3.1 Métodos

#### 3.1.1 Teoría del analizador

Es un programa informático que analiza una cadena de símbolos de acuerdo con las reglas de una gramática formal.

El análisis sintáctico convierte el texto de entrada en otras estructuras (comúnmente árboles), que son más útiles para el posterior análisis y capturan la jerarquía implícita de la entrada. Un analizador léxico crea tokens de una secuencia de caracteres de entrada y son estos tokens los que son procesados por el analizador sintáctico para construir la estructura de datos, por ejemplo, un árbol de análisis o árboles de sintaxis abstracta.

El análisis sintáctico también es un estado inicial del análisis de frases de lenguaje natural. Es usado para generar diagramas de lenguajes que usan flexión gramatical, como los idiomas romances o el latín. Los lenguajes habitualmente reconocidos por los analizadores sintácticos son los lenguajes libres de contexto. Cabe notar que existe una justificación formal que establece que los lenguajes libres de contexto son aquellos reconocibles por un autómata de pila, de modo que todo analizador sintáctico que reconozca un lenguaje libre de contexto es equivalente en capacidad computacional a un autómata de pila.

Tras realizar la práctica con CUP he observado que el analizador será LALR (1).

#### 3.1.2 Creación de un proyecto en CUP

Lo primero de todo tenemos que configurar el archivo .cup que será el encargado de generar las clases Cup.java y sym.java.

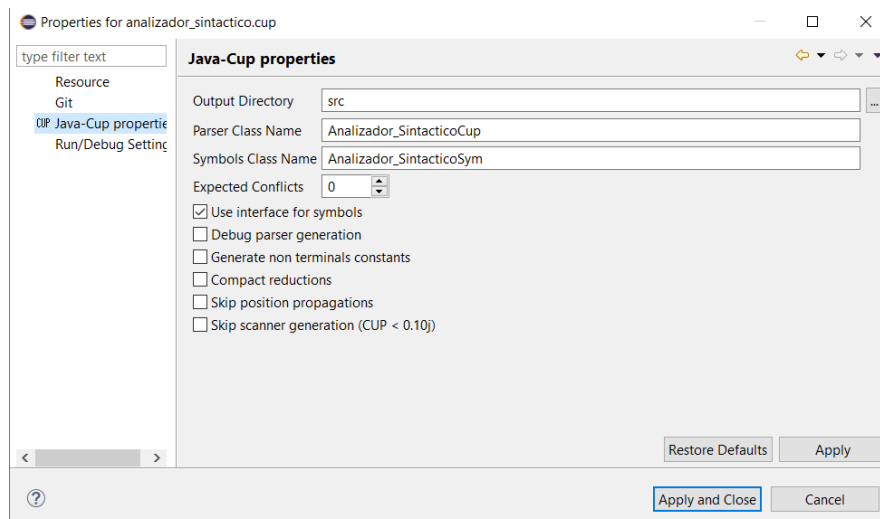


FIG. 18. CONFIGURACIÓN CUP

**Clase Cup.java:** Es la clase principal en la que se genera nuestro programa hace de unión entre léxico y sintactico.

**Clase sym.java:** Es la clase generada por el cup donde se guarda el valor de todos los simbolos terminales que podrá recibir del analizador lexico, añade a esta los símbolos para que puedan ser usados. Se deben implementar para el correcto funcionamiento del cup para el análisis las siguientes librerías:

```
package PECL1;

import java_cup.runtime.*;
import java.io.*;
import java.util.*;
```

El archivo CUP consta de cuatro secciones:

- **Código de Usuario:** Código Java escrito por el usuario para iniciar la lectura y hacer comprobaciones semánticas
- **Nodos terminales y no terminales:** Define la lista de símbolos a utilizar por el analizador sintáctico, asigna cada valor con el devuelto con léxico, es una de las partes más importantes.
- **Declaración de precedencia:** Esta opción es opcional y en el enunciado aconsejan no usarla se declara como “precedence” y es para evitar errores de gramática ambigua.

### 3.1.3 Implementación analizador sintáctico.

Lo primero de todo vendrá dado por las opciones y declaraciones como mostramos en el apartado anterior donde se hace referencia al analizador y la declaración de paquetes de java.

Lo primero que podemos ver al crear el archivo es la parte parse code. En él se encuentra el main donde leerá solo exclusivamente archivos .bas, también mostramos por pantalla el contenido del fichero y al finalizar el parse mostramos los símbolos leídos por el analizador léxico.



En nuestro fichero CUP debemos de pasarle al parser el fichero que deberá leer de esta forma hacemos la unión del lex y el cup.

```
public static void main (String argc[]) throws Exception{
    String extension="";
    String programa=argc[0];

    /***** COMPROBACION EXTENSION *****/
    for (int i=programa.length()-4;i<programa.length();i++){
        extension=extension+programa.charAt(i);
    }
    if(extension.equals(".bas")){
        System.out.println(">>> PRACTICA PROCESADORES 2017/2018 <<<\n");
        System.out.println("Nombre del programa: " + programa+"\n");

        /***** IMPRIMIR EL PROGRAMA A LEER *****/
        String cadena;
        FileInputStream basic =new FileInputStream(programa);
        FileReader program = new FileReader(programa);
        BufferedReader print_program = new BufferedReader(program);
        while((cadena = print_program.readLine())!=null) {
            System.out.println(cadena);
        }
        print_program.close();

        /***** INICIALIZAR EL ANALIZADOR *****/

        Analizador_LexicoLex y=new Analizador_LexicoLex(basic);
        Analizador_SintacticoCup principal=new Analizador_SintacticoCup(y);
        principal.parse();
        System.out.println("\n***** ANALIZADOR LEXICO *****\n");
        y.writeOutputFile();
        System.out.println("\n\nNumero de errores lexicos: "+ y.getErr()+"\n");
    }
    else{
        System.err.println("ERROR. EL ARCHIVO DEBE DE SER .bas");
    }
}
```

FIG. 19. FICHEROS Y VALIDACIONES

## Gramática

El primer paso para realizar nuestro analizador es declarar todos los símbolos terminales que nos pueda devolver el analizador léxico y a continuación los posibles nodos no terminales que se vayan a usar.

## Terminales

```

terminal      DATA, DEF, DIM, END, FOR, GO, GOSUB, GOTO, IF, INPUT, LET, NEXT, ON, PRINT, RND, READ,
              REM, RESTORE, RETURN, STEP, STOP, THEN, TO, IGUAL, MAS, MENOS, MUL, ID_CADENA, ID_NUM_SUS,
              DIVISION, CRLF, PUNTOYCOMA, FUNCION_MAT, PARENTESIS_IZQ, PARENTESIS_DER, ID_NUM,
              RANDOMIZE, FUNCION, EXPONENTE, COMA, MENOR, MAYOR, MAYOR_IGUAL, MENOR_IGUAL, DISTINTO,
              ENTERO, CONSTANTE_NUM;

terminal String CONSTANTE;

```

FIG. 20. TERMINALES

En los terminales recogeremos cada token que declaramos en el analizador léxico. Para solucionar los casos de precedencia.

### No terminales

```

non terminal String inicio, programa, leerLinea, expresion, operacion, operador, operadorParent, opcion2, imprimir;
non terminal String operaciones, funcionMat, variable, asignacion, declaracion, if_then, operando, opcion;
non terminal String goto, comparacion, on_goto, multiple_goto, gosub_return, iniciolinea, mete_entero, print,
              item, nextItem, data, data_var, input_var, input, read, read_var, more_read, for_to,
              variable_num, limit_for, iguallet, variable_if, to_go;

```

FIG. 21. NO TERMINALES

Estos son creados para ayudarnos a evitar conflictos de reducción o desplazamiento, así como de ambigüedad.

### Demostración gramática

Como podemos observar en la imagen (fig. 22) la gramática no tiene problemas de diseño de desplazamiento – reducción o reducción – reducción.

```

----- CUP v0.11a beta 20060608 Parser Generation Summary -----
 0 errors and 0 warnings
 50 terminals, 47 non-terminals, and 130 productions declared,
 producing 199 unique parse states.
 0 terminals declared but not used.
 0 non-terminals declared but not used.
 0 productions never reduced.
 0 conflicts detected (0 expected).
 Code written to "Analizador_SintacticoCup.java", and "Analizador_SintacticoSym.java".
----- (v0.11a beta 20060608)

```

FIG. 22 COMPROBACIÓN GRAMÁTICA

```

inicio      ::= programa{:finish();};

programa    ::= mete_entero END
              {
                if(cont_gosub_return != 0){
                  System.err.println("Error semantico se esperaba una sentencia RETURN para el GOSUB ");
                  numErrS++;
                }
                if(!checkGoSub()){
                  System.err.println("Error semantico la linea definida para el GOSUB no existe ");
                  numErrS++;
                }
                if(cont_for != 0){
                  System.err.println("Error semantico se esperaba una sentencia NEXT para el FOR ");
                  numErrS++;
                }
              };
              | inicioLinea programa
              | mete_entero RETURN:re
              {
                if(cont_gosub_return == 0){
                  System.err.println("Error semantico se esperaba una sentencia GOSUB para el return ");
                  numErrS++;
                }
                else{
                  cont_gosub_return--;
                }
              };
              CRLF programa
              | error programa {:numErr++;};

```

La primera de las producciones tienes que ver con las posibles lecturas del programa se puede ver que todas las líneas deben empezar con un entero seguido de otros tokens, tenemos que tener cuidado con las producciones de END y RETURN porque una marca el final del programa y la otra el final de la sentencia GOSUB, por eso se declaran en esta parte.

```

inicioLinea ::= mete_entero leerLinea CRLF
              | mete_entero REM CRLF
              | mete_entero STOP CRLF;

mete_entero  ::= ENTERO:e
              {
                checkLine(e.toString(), eleft+1);
              };

```

Aquí se analiza la línea que se lee a excepción del END y RETURN, podemos observar que puede haber una sentencia REM y una STOP o la de leer línea que se expone a continuación.

Para concluir la sentencia leerLinea nos ayuda a la hora de tener varias producciones de elementos que componen la gramática.

```

leerLinea ::= DEF expresion
| goto
| on_goto
| if_then
| gosub_return
| print
| read
| data
| input
| for_to
| DATA data
| NEXT ID_NUM:var
{
    int linea = varleft +1;
    if(cont_for == 0){
        System.err.println("Error semantico se esperaba un FOR antes del NEXT ->" + linea );
        numErr++;
    }else if(!var.equals(var_for.get(cont_for-1))){
        System.err.println("Error semantico el NEXT espera la variable del ultimo FOR ->" + linea );
        numErr++;
    }
    cont_for--;
}
:}
| NEXT PARENTESIS_IZQ ID_NUM:var PARENTESIS_DER
{
    int linea = varleft +1;
    if(cont_for == 0){
        System.err.println("Error semantico se esperaba un FOR antes del NEXT ->" + linea );
        numErr++;
    }else if(!var.equals(var_for.get(cont_for-1))){
        System.err.println("Error semantico el NEXT espera la variable del ultimo FOR ->" + linea );
        numErr++;
    }
    cont_for--;
}
:}
| RANDOMIZE
| RESTORE { :contRead=0;:}
| LET asignacion
| DIM declaracion
| error { :numErr++;:};

```

Aquí definimos las posibles palabras reservadas que habrá después de leer una línea con la llamada a su consecuencia, todo este proceso es recursivo hasta que encuentra la sentencia END.

### Tratamiento de errores

Según el tratamiento de errores que se nos pide se deben hacer el control de tres tipos de errores que pueden ocasionar en el analizador sintáctico.

- Reportar Errores: Tipo de errores leídos si se esperan otro token.
- Errores Sintácticos.
- Errores fatales: Si no se puede recuperar el error.

```

//Errores sintacticos
public void syntax_error(Symbol s) {
    System.err.println("Error de sintaxis linea " +(s.left+1)+" en o cerca de "+s.value);
}
//Errores Sintacticos que no se pueden recuperar
public void report_fatal_error(String message, Object info) {
    done_parsing();
    report_error("Error de sintaxis fatal : " + "No se puede recuperar del error y continuar con el analisis - "+ message + " ",(Symbol)info);
    report_error("*** Final del analisis.", null);
    System.exit(1);
}

```

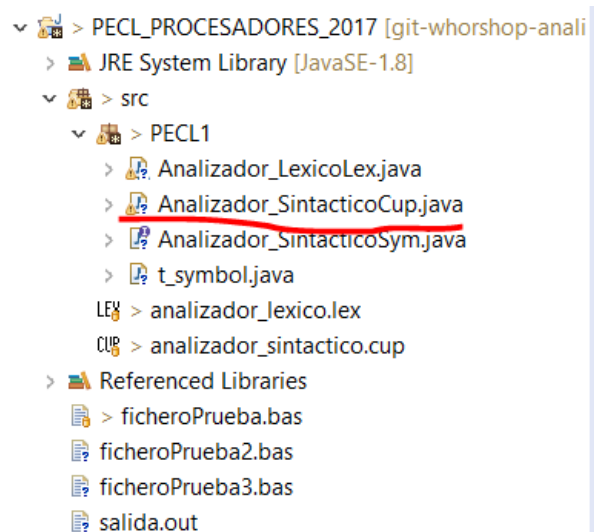
FIG. 23. ERRORES SINTÁCTICOS

### 3.2 Tratamiento de errores

Se presenta el siguiente fichero con errores para analizar los diferentes requisitos léxicos:

```
1 10 INPUT "Cual es tu nombre?"; U$
2 20 PRINT "Hola "; U$
3 30 INPUT "Cuantas estrellas quieres?"; N
4 40 LET S$
5 50 FOR I = 1 TO
6 60 LET S$ = S$ + "*"
7 70 NEXT (I)
8 80 PRINT S$
9 90 INPUT "Quieres mas estrellas?"; A$
10 120 IF A$ = "S" THEN GOTO 30
11 130 PRINT "Adios !! "; U$
12 140 END
```

Una vez guardado los cambios en el fichero .cup que genera el código java del proyecto.



- **RS 1.-** El analizador solamente ha de aceptar ficheros con extensión “.bas”.

```

/***** COMPROBACION EXTENSION *****/
for (int i=programa.length()-4;i<programa.length();i++){
    extension=extension+programa.charAt(i);
}
if(extension.equals(".bas")){
    System.out.println(">>> PRACTICA PROCESADORES 2017/2018 <<<\n");
    System.out.println("Nombre del programa: " + programa+"\n");

    /***** IMPRIMIR EL PROGRAMA A LEER *****/
    String cadena;
    FileInputStream basic =new FileInputStream(programa);
    FileReader program = new FileReader(programa);
    BufferedReader print_program = new BufferedReader(program);
    while((cadena = print_program.readLine())!=null) {
        System.out.println(cadena);
    }
    print_program.close();

    /***** INICIALIZAR EL ANALIZADOR *****/

    Analizador_LexicoLex y=new Analizador_LexicoLex(basic);
    Analizador_SintacticoCup principal=new Analizador_SintacticoCup(y);
    principal.parse();
    System.out.println("\n***** ANALIZADOR LEXICO *****\n");
    y.writeOutputFile();
    System.out.println("\n\nNumero de errores lexicos: "+ y.getErr()+"\n");
}
else{
    System.err.println("ERROR. EL ARCHIVO DEBE DE SER .bas");
}
}

```

- **RS 2.-** El analizador también debe ser capaz de detectar los errores sintácticos que no se ajusten a las especificaciones dadas. Cuando detecte un error, el analizador propuesto deberá mostrar un mensaje que indique la línea donde se ha producido. Al final del análisis, y en caso de haber encontrado errores, deberá mostrar un informe con el número total de errores que se han producido. Por el contrario, si el analizador hubiera terminado sin encontrar ningún error deberá mostrar un mensaje indicando que ha finalizado sin errores.
- **RS 3.-** El analizador debe ser capaz de recuperarse cuando encuentre un error recuperable y continuar analizando el resto del programa.

#### Comprobamos el archivo anterior

```

Error de sintaxis linea 4 en o cerca de null
Error de sintaxis linea 5 en o cerca de null
Error semantico se esperaba un FOR antes del NEXT ->7

*****ANALISIS FINALIZADO*****
Numero de errores sintacticos: 2
Numero de errores semanticos: 1

```

#### Archivo sin errores:

```

10 INPUT "Cual es tu nombre?"; U$
20 PRINT "Hola "; U$
30 INPUT "Cuantas estrellas quieres?"; N
40 LET S$ = "Como te va?"
50 FOR I = 1 TO N
60 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

*****ANALISIS FINALIZADO*****
Numero de errores sintacticos: 0
Numero de errores semanticos: 0

```

## 4. Diseño del analizador semántico

### 4.1 Métodos

#### 4.1.1 Teoría del analizador semántico

El analizador semántico es una fase del compilador que utiliza como entrada el árbol sintáctico este se encarga de detectar el resto de los posibles errores que pueden surgir. Es la fase posterior al análisis sintáctico y la última de la fase de análisis. Es un conjunto de reglas formales que especifican la estructura del lenguaje.

#### 4.1.2 Análisis y resultados obtenidos según su implementación (especificaciones Semánticas)

Para analizar los requisitos semánticos se indicarán en cada punto el ejemplo de una regla de la gramática en concreto.

**ESm 1.-** Todos los identificadores de variable utilizados en las expresiones deberán haber sido declarados previamente.

```
Hashtable<String, t_symbol> tabla_symbol = new Hashtable<String, t_symbol>();

public void checkVarExist(String var, int linea){
    boolean exist = false;
    if(tabla_symbol.isEmpty()){
        System.err.println("Error semantico el operando no se encuentra definido como una variable "+var+" linea -> "+linea);
        numErrs++;
    }else{
        Enumeration e=tabla_symbol.keys();
        while(e.hasMoreElements()){
            char varName = var.charAt(0);
            String key = e.nextElement().toString();
            if(key.length() == 1 || key.charAt(1) != '$'){ //==1 para variables numericas simples el otro para que no coja cadena
                if(key.charAt(0) == varName){
                    exist=true;
                }
            }else{
                if(key.equals(var)){
                    exist=true;
                }
            }
        }
        if(!exist){
            System.err.println("Error semantico el operando no se encuentra definido como una variable "+var+" linea -> "+linea);
            numErrs++;
        }
    }
}
```

Vamos a mostrar algunas comprobaciones con este archivo:

```
110 DIM A(8), B(8,4)
220 DATA 3.15159, 24, 3,"hola", 3^-2, "adi9o"
322 FOR I = 0 TO 2
424 FOR J = 0 TO 2
525 LET C$ = "HOLA"
626 NEXT J
727 NEXT I
830 INPUT "HOLA", N
935 DATA 3578344
1040 DEF FNK (B) = (N/B)+A+C
1145 GOSUB 80
1246 LET Z1 = "HOLA"
1347 GOSUB 70
1448 GOSUB 60
1550 LET S$ = A / 2
1660 STOP
1765 READ J
1870 GOTO 10
1975 READ X,D
2080 IF A <> (D+I/B)+A THEN GOTO 60
21100 ON (A+B) GOTO 10,20,50
22110 RETURN
23115 RETURN
24117 RETURN
25130 PRINT (A+B);"HOLA", S$
26190 END
```

Y nos devuelve los siguientes errores:

Error semantico el operando no se encuentra definido como una variable C linea -> 10  
Error semantico estas intentando guardar una operacion en un variable cadena 12  
Error semantico estas intentando guardar una cadena en un variable numerica 15

\*\*\*\*\*ANALISIS FINALIZADO\*\*\*\*\*

Numero de errores sintacticos: 0

Numero de errores semanticos: 3

**ESm 2.-** Un identificador que aparece como un elemento numérico debe ser una variable numérica.

**ESm 3.-** Un identificador que aparece como un elemento cadena debe ser una variable cadena.

```
public boolean checkNumOrCad(String var){ //funcion para comprobar si es id de cadena o numerico
    boolean numeric = true;
    for(int i = 0; i < var.length(); i++){
        if(var.charAt(i)=='$'){
            numeric = false;
        }
    }
    return numeric;
}
```

En esta función comprobamos si tenemos un numero o una cadena y después desde los tokens devueltos comprobamos si es correcto, por ejemplo, en la palabra reservada LET:

```
//LET
asignacion ::= variable:var IGUAL operacion:op
{
    int lineaErr = varleft+1;
    if(checkNumOrCad(var)){
        checkVar(var, op, lineaErr);
    }else{
        System.err.println("Error semantico estas intentando guardar una cadena en un variable numerica "+ lineaErr);
        numErrS++;
    }
}

| variable:var IGUAL iguallet:op
{
    int lineaErr = varleft+1;
    if(!checkNumOrCad(var)){
        checkVar(var, op, lineaErr);
    }else{
        System.err.println("Error semantico estas intentando guardar una operacion en un variable cadena "+ lineaErr);
        numErrS++;
    }
}

// ASIGNACION LET
iguallet ::= CONSTANTE:var
{ :RESULT = var.toString(); }
| MAS:var iguallet:op { :RESULT = var.toString() + " "+op.toString(); }
| ID_CADENA:var { :RESULT = var.toString(); }
| CONSTANTE:var iguallet:op
{ :RESULT = var.toString() + " "+op.toString(); }
| ID_CADENA:var iguallet:op { :RESULT = var.toString() + " "+op.toString(); }
```

**ESm 4.-** Una expresión suma, resta, multiplicación, división o potenciación que se aplique será de tipo numérica.



```

//EXPRESION NUMERICA
operacion ::= opcion:o { : RESULT = o; : }
           | error opcion ;

opcion ::= operaciones:o { : RESULT = o; : }
         | PARENTESIS_IZQ operaciones:o PARENTESIS_DER opcion2:o2 { : RESULT = "(" + o2; : };

operaciones ::= operador:o operando:op operadorParent:opa opcion2:o2 { :String s=o+" "+opa+" "+o2; RESULT=s; : }
              | operando:o1 operador:o operando:o2 operadorParent:opa opcion2:op2 { :String s=o1+" "+o2+" "+opa+" "+op2; RESULT=s; : }
              | operando:o1 operadorParent:opa { :String s=o1+" "+opa; RESULT = s; : };

opcion2 ::= opcion:o { : RESULT=o; : } | { : RESULT=""; : };

operando ::= CONSTANTE_NUM:id { : RESULT=id.toString(); : }
           | ID_NUM_SUS:id { :checkVarExist(id.toString(), idleft+1); RESULT=id.toString(); : }
           | ID_NUM:id { :checkVarExist(id.toString(), idleft+1); RESULT=id.toString(); : }
           | FUNCION_MAT:fn funcionMat:o { :checkFunMat(fn.toString(), o.toString(), fnleft); String s = fn + " " + o; RESULT = s; : }
           | FUNCION:fn { :checkFunExists(fn.toString(), fnleft+1); RESULT = fn.toString(); : }
           | RND:rn { : RESULT = rn.toString(); : };

operadorParent ::= operador:o { : RESULT=o; : }
                | { : RESULT=""; : };

operador ::= MAS { : RESULT="+"; : }
           | MENOS { : RESULT="-"; : }
           | DIVISION { : RESULT="/"; : }
           | MUL { : RESULT="*"; : }
           | EXPONENTE { : RESULT="^"; : };

```

Esta función solo es llamada por operadores que sean válidos una expresión numérica. El operador “+” es aceptado para unir cadenas.

**ESm 6.-** La misma letra no debe ser el nombre de una variable simple y una matriz, ni el nombre de una matriz unidimensional y una bidimensional. Esto se comprueba cada vez que intentamos guardar una variable nueva a nuestra tabla de símbolos.

```

public void addVariable(String name, String valor, int linea){
    boolean put = false;
    if(tabla_symbol.isEmpty()){
        if(checkNumOrCad(name)){
            if(name.length() > 1){
                tabla_symbol.put(name, new t_symbol("NUM_SUSCRITA", valor));
            }else{
                tabla_symbol.put(name, new t_symbol("NUMERICA", valor));
            }
        }else{
            tabla_symbol.put(name, new t_symbol("CADENA", valor));
        }
    }else{
        Enumeration e=tabla_symbol.keys();
        while(e.hasMoreElements()){
            char varName = name.charAt(0);
            String key = e.nextElement().toString();
            if(name.length() > 1 && name.charAt(1) == '$'){
                if(tabla_symbol.containsKey(key)){
                    System.err.println("Error semantico al declarar la variable ya existe una con ese nombre "+name+" linea -> "+linea);
                    numErrS++;
                }else{
                    if(valor.length() > 18){
                        System.err.println("Error semantico el valor de la variable cadena es demasiado largo linea -> "+linea);
                        numErrS++;
                    }else{
                        put = true;
                    }
                }
            }else{
                if(key.charAt(0) == varName){
                    System.err.println("Error semantico al declarar la variable ya existe una con ese nombre "+name+" linea -> "+linea);
                    numErrS++;
                }else{
                    put=true;
                }
            }
        }
    }
    if(put){
        if(checkNumOrCad(name)){
            if(name.length() > 1){
                tabla_symbol.put(name, new t_symbol("NUM_SUSCRITA", valor));
            }else{
                tabla_symbol.put(name, new t_symbol("NUMERICA", valor));
            }
        }else{
            tabla_symbol.put(name, new t_symbol("CADENA", valor));
        }
    }
}
}

```

**ESm 7.-** La longitud de la cadena de caracteres asociada con una variable de cadena puede variar durante la ejecución de un programa desde una longitud de cero caracteres (significando la cadena nula o vacía) y puede contener hasta un máximo de 18 caracteres.

Se puede observar que se controla en la función de arriba y también cuando remplazamos el valor de una variable.

```
public void checkVar(String name, String valor, int linea){
    if(tabla_symbol.containsKey(name)){
        if(!checkNumOrCad(name) && valor.length() > 18){
            System.err.println("Error semantico el valor de la variable cadena es demasiado largo linea -> "+linea);
            numErrS++;
        }else{
            t_symbol simbol = tabla_symbol.get(name);
            simbol.setValor(valor);
            tabla_symbol.replace(name, simbol);
        }
    }else{
        addVariable(name, valor, linea);
    }
}
```

**ESm 9.-** El valor del argumento de la función LOG no será cero o negativo. El valor del argumento de la función SQR no será es negativo.

**ESm 10.-** La magnitud del valor de la función exponencial o tangente es mayor que el infinito de la máquina (no fatal, el procedimiento de recuperación recomendado es suministrar el infinito de la máquina con el signo apropiado y continuar).

Para infinito tanto por arriba como por abajo he cogido un número muy grande.

```
public void checkFunMat(String nombre,String valor ,int linea){
    if(nombre.equals("EXP")){
        if(Integer.parseInt(valor) > 999999999 && Integer.parseInt(valor) > -999999999){ //infinito
            System.err.println("Error semantico el valor para la funcion EXP es demasiado grande linea -> "+linea);
            numErrS++;
        }
    } else if(nombre.equals("TAN")){
        if(Integer.parseInt(valor) > 999999999 && Integer.parseInt(valor) > -999999999){ //infinito
            System.err.println("Error semantico el valor para funcion TAN es demasiado grande que 0 linea -> "+linea);
            numErrS++;
        }
    }else if(nombre.equals("ATN")){
        if(Integer.parseInt(valor) > 999999999 && Integer.parseInt(valor) > -999999999){ //infinito
            System.err.println("Error semantico el valor para funcion ATN es demasiado grande que 0 linea -> "+linea);
            numErrS++;
        }
    }else if(nombre.equals("LOG")){
        if(Integer.parseInt(valor) >=0){
            System.err.println("Error semantico la funcion LOG tiene que ser mayor igual que 0 linea -> "+linea);
            numErrS++;
        }
    }else if(nombre.equals("SQR")){
        if(Integer.parseInt(valor) > 0){
            System.err.println("Error semantico la funcion SQR tiene que ser mayor que 0 linea -> "+linea);
            numErrS++;
        }
    }
}
```

**ESm 11.-** Una definición de función, DEFx, debe producirse en una línea numerada inferior a la de la primera referencia a la función. El parámetro que aparece en la lista de parámetros de una definición de funciones local a esa definición, es decir, es distinto de cualquier variable con

el mismo nombre fuera de la función definición. Las variables que no aparecen en la lista de parámetros son las variables del mismo nombre fuera de la definición de la función.

```
public void checkFunExists(String name, int linea){
    String[] nameFun = name.split(" ");
    if(!tabla_symbol.containsKey(nameFun[0])){
        System.err.println("Error semantico la funcion no ha sido definida anteriormente linea -> "+linea);
        numErrs++;
    }
}
```

Comprobamos si existe el nombre de la función en la tabla de símbolos, si no existe mostramos un error.

**ESm 12.-** En una sentencia de asignación, LET, el tipo de la variable en la parte izquierda de la asignación y el tipo de la expresión en la parte derecha debe ser el mismo.

Se ha demostrado en la regla **ESm 3**.

**ESm 13.-** Una instrucción de entrada, INPUT, hace que las variables de la lista de variables se asignen, en orden, a los valores de la entrada-respuesta. Los tipos de la variable en la parte izquierda de la asignación y los tipos de la expresión en la parte derecha debe ser del mismo tipo.

No entiendo esta pregunta ya que los valores son introducidos por un usuario, aun así, se realiza la siguiente comprobación en la parte de las variables mostrada más arriba.

**ESm 14.-** En una sentencia condicional, IF, la expresión debe ser de tipo booleano, mientras que las sentencias que forman parte de la sección THEN será una constante natural.

He contemplado la utilización de expresiones como la de constantes y al igual que antes las comprobaciones se hacen en las variables.

```
//IF THEN
if_then      ::= IF operacion comparacion operacion THEN to_go //Se contempla tambien las variables suscritas y simples
               | IF variable_if comparacion variable_if THEN to_go; //para comparacion con una cadena

variable_if   ::= ID_CADENA: var{:checkVarExist(var.toString(), varleft+1);;}
               | CONSTANTE ;

to_go         ::= goto | gosub_return;
```

**ESm 15.-** La expresión ON-GOTO se evaluará y redondeará para obtener un número entero, cuyo valor se utilizará para seleccionar un número de línea de la lista que sigue al GOTO (los números de línea de la lista se indexan de izquierda a derecha, Empezando por 1). La ejecución del programa continuará en la declaración con el número de línea seleccionado. Todos los números de línea en las instrucciones de control se referirán a las líneas del programa. El número entero obtenido como valor de una expresión en una sentencia ON-GOTO no puede ser menor que uno o mayor que el número de números de línea en la lista.

```
public boolean checkExistLine(String digito){
    int aux=0;
    aux = Integer.parseInt(digito);
    return listaLineas.contains(aux);
}
```

Se comprueba que la línea de referencia existe, si no existe lanza un error semántico.

```

//ON...GOTO
on_goto ::= ON operacion GOTO multiple_goto | ON operacion GO TO multiple_goto;
multiple_goto ::= CONSTANTE_NUM:i
{
    if(!checkExistLine(i.toString())){
        int linea = ileft+1;
        System.err.println("Error semantico la linea para el GOTO no existe "+i.toString()+" linea: " + linea);
        numErr++;
    }
}
| CONSTANTE_NUM:i COMA multiple_goto
{
    if(!checkExistLine(i.toString())){
        int linea = ileft+1;
        System.err.println("Error semantico la linea para el GOTO no existe "+i.toString()+" linea: " + linea);
        numErr++;
    }
}
| ENTERO:i
{
    if(!checkExistLine(i.toString())){
        int linea = ileft+1;
        System.err.println("Error semantico la linea para el GOTO no existe "+i.toString()+" linea: " + linea);
        numErr++;
    }
}
| ENTERO:i COMA multiple_goto
{
    if(!checkExistLine(i.toString())){
        int linea = ileft+1;
        System.err.println("Error semantico la linea para el GOTO no existe "+i.toString()+" linea: " + linea);
        numErr++;
    }
}
};

```

**ESm 16.-** Deberá construir una estructura para el almacenamiento de la tabla de símbolos similar a la descrita.

Este requisito para cumplirlo debemos hacer uso del Hashtable, que nos servirá para la creación de la tabla de símbolos, tengo una clase java llamado t\_símbolo que me ayudara a guardar el valor y el tipo de símbolo. Todos los símbolos son introducidos desde addVariable, addFunc o addWordBook. Para imprimir por pantalla se añadirá una nueva estructura en Java para realizar el volcado del Hashtable de forma correcta.



## 5. Conclusiones

El trabajo realizado en esta práctica representa una gran aportación significativa ya que es bastante interesante ver como realmente trabajan los analizadores en una capa de bajo nivel y esto mucha gente lo desconoce.

En la puesta en práctica, se puede ver como son de completos y algo tan simple como puede ser una gramática puede ocasionar tantos problemas solo por añadir un nodo no terminal o terminal en el lugar equivocado.

En cuanto a la dificultad de la práctica, me ha resultado muy difícil comprender los enunciados, ya que muchas veces no encontraba la coherencia fácilmente, por lo que mi trabajo se veía limitado.

Estoy bastante contento con el trabajo de la práctica ya que me parecía algo imposible cuando empecé, pero al final ha salido casi todo, me hubiese gustado terminar al 100% pero no ha podido ser, aun así, me quedo con lo positivo y espero que el esfuerzo haya merecido la pena.

## 6. Referencias

- 1) PDF proporcionados desde la plataforma
- 2) <https://www.youtube.com/watch?v=mHFBLm4GXnk>
- 3) <https://www.youtube.com/watch?v=XQHivIfKvMk>

## 7. Apéndice

Ahora vamos a analizar los diferentes casos que se pueden presentar a la hora de usar nuestro analizador para ello vamos a comprobar 10 posibles casos, 2 de ellos formados correctamente y 8 con errores léxicos, sintácticos o semánticos controlados según marca la práctica. Los diferentes ficheros se encuentran en la carpeta de la practica en formato txt para que se no haya problemas al acceder a ellos, para poder analizarlos tenemos que copiar el contenido en el fichero "prueba.bas" que se encuentra dentro del proyecto.

Primero se presentarán los ficheros correctos y luego los erróneos:

## Fichero 1:

```
110 DIM A(8), B(8,4)
220 DATA 3.15159, 24, 3,"hola", 3^-2, "adi9o"
322 FOR I = 0 TO 2
424 FOR J = 0 TO 2
525 LET C$ = "HOLA"
626 NEXT J
727 NEXT I
830 INPUT "HOLA", N
935 DATA 3578344
1040 DEF FN(X) = (N/B)+A
1145 GOSUB 80
1246 LET Z1 = FN(X)
1347 GOSUB 70
1448 GOSUB 60
1550 LET S$ = "hola"
1660 STOP
1765 READ J
1870 GOTO 10
1975 READ X,D
2080 IF A <> (D+N/B)+A THEN GOTO 60
2100 ON (A+B) GOTO 10,20,50
2210 RETURN
2315 RETURN
2417 RETURN
25130 PRINT (A+B);"HOLA", S$
26190 END
```

\*\*\*\*\*ANALISIS FINALIZADO\*\*\*\*\*

Numero de errores sintacticos: 0  
Numero de errores semanticos: 0

\*\*\*\*\* TABLA DE SIMBOLOS \*\*\*\*\*

\*\*\*\*\* PALABRAS RESERVADAS \*\*\*\*\*

NOMBRE	TIPO
GOSUB	PALABRA_RESERVADA
INPUT	PALABRA_RESERVADA
READ	PALABRA_RESERVADA
ON_GOTO	PALABRA_RESERVADA
GOTO	PALABRA_RESERVADA
END	PALABRA_RESERVADA
RETURN	PALABRA_RESERVADA
DATA	PALABRA_RESERVADA
IF_THEN	PALABRA_RESERVADA
DEF	PALABRA_RESERVADA
PRINT	PALABRA_RESERVADA
LET	PALABRA_RESERVADA
DIM	PALABRA_RESERVADA
FOR	PALABRA_RESERVADA
NEXT	PALABRA_RESERVADA

\*\*\*\*\* VARIABLES \*\*\*\*\*

NOMBRE	TIPO	INICIALIZADO
C\$	CADENA	Hola
X	NUMERICA	24
N	NUMERICA	
FN(X)	FUNCION	( N/B ) +A
J	NUMERICA	3.15159
S\$	CADENA	hola
D	NUMERICA	3
Z1	NUM_SUSCRITA	FN(X)
B(8,4)	NUM_SUSCRITA	
A(8)	NUM_SUSCRITA	

\*\*\*\*\* ANALIZADOR LEXICO \*\*\*\*\*

```
[ent(10),DIM,ide(A(8)),coma,ide(B(8,4)),CRLF,
ent(20),DATA,const_num(3.15159),coma,const_num(24),coma,const_num(3),coma,const(hola),coma,const_num(3^-2),coma,const(adi9o),CRLF,
ent(22),FOR,id(I),igual,const_num(0),TO,const_num(2),CRLF,
ent(24),FOR,id(J),igual,const_num(0),TO,const_num(2),CRLF,
ent(25),LET,ide(C$),igual,const(Hola),CRLF,
ent(26),NEXT,id(J),CRLF,
ent(27),NEXT,id(I),CRLF,
ent(30),INPUT,const(Hola),coma,id(N),CRLF,
ent(35),DATA,const_num(3578344),CRLF,
ent(40),DEF, FN(X) = (N/B)+A, PARENTESIS_IZQ,id(N),div,id(B),PARENTESIS_DER,mas,id(A),CRLF,
ent(45),GOSUB,const_num(80),CRLF,
ent(46),LET,id(Z1),igual, FN(X),CRLF,
ent(47),GOSUB,const_num(70),CRLF,
ent(48),GOSUB,const_num(60),CRLF,
ent(50),LET,ide(S$),igual,const(hola),CRLF,
ent(60),STOP,CRLF,
ent(65),READ,id(J),CRLF,
ent(70),GOTO,const_num(10),CRLF,
ent(75),READ,id(X),coma,id(D),CRLF,
ent(80),IF,id(A),distinto,PARENTESIS_IZQ,id(D),mas,id(N),div,id(B),PARENTESIS_DER,mas,id(A),THEN,GOTO,const_num(60),CRLF,
ent(100),ON,PARENTESIS_IZQ,id(A),mas,id(B),PARENTESIS_DER,GOTO,const_num(10),coma,ent(20),coma,ent(50),CRLF,
ent(110),RETURN,CRLF,
ent(115),RETURN,CRLF,
ent(117),RETURN,CRLF,
ent(130),PRINT,PARENTESIS_IZQ,id(A),mas,id(B),PARENTESIS_DER,PUNTOYCOMA,const(Hola),coma,ide(S$),CRLF,
ent(190),END,Eof]
```

Numero de errores lexicos: 0

## Fichero 2:

```
110 INPUT "Cual es tu nombre?"; U$
220 PRINT "Hola "; U$
330 INPUT "Cuantas estrellas quieres?"; N
440 LET S$ = "Como te va?"
550 FOR I = 1 TO N
660 LET S$ = S$ + "*"
770 NEXT (I)
880 PRINT S$
990 INPUT "Quieres mas estrellas?"; A$
10120 IF A$ = "S" THEN GOTO 30
11130 PRINT "Adios !! "; U$
12140 END
```

\*\*\*\*\*ANALISIS FINALIZADO\*\*\*\*\*

Numero de errores sintacticos: 0

Numero de errores semanticos: 0

\*\*\*\*\* TABLA DE SIMBOLOS \*\*\*\*\*

\*\*\*\*\* PALABRAS RESERVADAS \*\*\*\*\*

NOMBRE	TIPO
FOR	PALABRA_RESERVADA
IF_THEN	PALABRA_RESERVADA
NEXT	PALABRA_RESERVADA
INPUT	PALABRA_RESERVADA
PRINT	PALABRA_RESERVADA
LET	PALABRA_RESERVADA
END	PALABRA_RESERVADA
GOTO	PALABRA_RESERVADA

\*\*\*\*\* VARIABLES \*\*\*\*\*

NOMBRE	TIPO	INICIALIZADO
S\$	CADENA	S\$ + *
N	NUMERICA	
A\$	CADENA	
U\$	CADENA	

\*\*\*\*\* ANALIZADOR LEXICO \*\*\*\*\*

```
[ent(10),INPUT,const(Cual es tu nombre?),PUNTOYCOMA,ide(U$),CRLF,
ent(20),PRINT,const(Hola ),PUNTOYCOMA,ide(U$),CRLF,
ent(30),INPUT,const(Cuantas estrellas quieres?),PUNTOYCOMA,id(N),CRLF,
ent(40),LET,ide(S$),igual,const(Como te va?),CRLF,
ent(50),FOR,id(I),igual,const_num(1),TO,id(N),CRLF,
ent(60),LET,ide(S$),igual,ide(S$),mas,const(*),CRLF,
ent(70),NEXT,PARENTESIS_IZQ,id(I),PARENTESIS_DER,CRLF,
ent(80),PRINT,ide(S$),CRLF,
ent(90),INPUT,const(Quieres mas estrellas?),PUNTOYCOMA,ide(A$),CRLF,
ent(120),IF,ide(A$),igual,const(S),THEN,GOTO,const_num(30),CRLF,
ent(130),PRINT,const(Adios !! ),PUNTOYCOMA,ide(U$),CRLF,
ent(140),END,EOF]
```

Numero de errorres lexicos: 0



### Fichero 3:

```
10 INPUT "Cual es tu nombre?"; U$
20 PRINT "Hola "; U$
30 INPUT "Cuantas estrellas quieres?"; N
40 LET S$ = N + 1
50 FOR I = 1 TO N
60 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
```

Error semantico estas intentando guardar una cadena en un variable numerica 4

Error lexico en linea 7 columna 13: @

Error lexico en linea 10 columna 30: #

\*\*\*\*\*ANALISIS FINALIZADO\*\*\*\*\*

Numero de errores sintacticos: 0

Numero de errores semanticos: 1

\*\*\*\*\* ANALIZADOR LEXICO \*\*\*\*\*

```
[ent(10),INPUT,const(Cual es tu nombre?),PUNTOYCOMA,ide(U$),CRLF,
ent(20),PRINT,const(Hola ),PUNTOYCOMA,ide(U$),CRLF,
ent(30),INPUT,const(Cuantas estrellas quieres?),PUNTOYCOMA,id(N),CRLF,
ent(40),LET,ide(S$),igual,id(N),mas,const_num(1),CRLF,
ent(50),FOR,id(I),igual,const_num(1),TO,id(N),CRLF,
ent(60),LET,ide(S$),igual,ide(S$),mas,const(*),CRLF,
ent(70),NEXT,PARENTESIS_IZQ,id(I),PARENTESIS_DER,CRLF,
ent(80),PRINT,ide(S$),CRLF,
ent(90),INPUT,const(Quieres mas estrellas?),PUNTOYCOMA,ide(A$),CRLF,
ent(120),IF,ide(A$),igual,const(S),THEN,GOTO,const_num(30),CRLF,
ent(130),PRINT,const(Adios !! ),PUNTOYCOMA,ide(U$),CRLF,
ent(140),END,EOF]
```

Numero de errores lexicos: 2

#### Fichero 4:

```
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
60 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
```

Error de sintaxis linea 4 en o cerca de null

Error de sintaxis linea 5 en o cerca de null

Error semantico se esperaba un FOR antes del NEXT linea ->7

\*\*\*\*\*ANALISIS FINALIZADO\*\*\*\*\*

Numero de errores sintacticos: 2

Numero de errores semanticos: 1

\*\*\*\*\* ANALIZADOR LEXICO \*\*\*\*\*

```
[ent(10),INPUT,const(Cual es tu nombre?),PUNTOYCOMA,ide(U$),CRLF,
ent(20),PRINT,const(Hola ),PUNTOYCOMA,ide(U$),CRLF,
ent(30),INPUT,const(Cuantas estrellas quieres?),PUNTOYCOMA,id(N),CRLF,
ent(40),LET,ide(S$),CRLF,
ent(50),FOR,id(I),igual,const_num(1),TO,CRLF,
ent(60),LET,ide(S$),igual,ide(S$),mas,const(*),CRLF,
ent(70),NEXT,PARENTESIS_IZQ,id(I),PARENTESIS_DER,CRLF,
ent(80),PRINT,ide(S$),CRLF,
ent(90),INPUT,const(Quieres mas estrellas?),PUNTOYCOMA,ide(A$),CRLF,
ent(120),IF,ide(A$),igual,const(S),THEN,GOTO,const_num(30),CRLF,
ent(130),PRINT,const(Adios !! ),PUNTOYCOMA,ide(U$),CRLF,
ent(140),END,EOF]
```

Numero de errores lexicos: 0

## Fichero 5:

```
10 DIM A(8), B(8,4)
20 DATA 3.15159, 24, 3,"hola", 3^-2, "adi9o"
22 FOR I = 0 TO 2
24 FOR J = 0 TO 2
25 LET C$ = "Hola"
26 NEXT J
27 NEXT I
30 INPUT "Hola", N
35 DATA 3578344
40 DEF FN (B) = (N/B)+A+C
45 GOSUB 80
46 LET Z1 = "Hola"
47 GOSUB 70
48 GOSUB 60
50 LET S$ = A + 2
60 STOP
65 READ J
70 GOTO 10
75 READ X,D
80 IF A <> (D+N/B)+A THEN GOTO 60
100 ON (A+B) GOTO 10,20,50
110 RETURN
115 RETURN
117 RETURN
130 PRINT (A+B);"Hola", S$
190 END
```

Error semantico el operando no se encuentra definido como una variable C linea -> 10  
Error semantico estas intentando guardar una operacion en un variable cadena 12  
Error semantico estas intentando guardar una cadena en un variable numerica 15

\*\*\*\*\*ANALISIS FINALIZADO\*\*\*\*\*  
Numero de errores sintacticos: 0  
Numero de errores semanticos: 3

\*\*\*\*\* ANALIZADOR LEXICO \*\*\*\*\*

```
[ent(10),DIM,ide(A(8)),coma,ide(B(8,4)),CRLF,
ent(20),DATA,const_num(3.15159),coma,const_num(24),coma,const_num(3),coma,const(hola),coma,const_num(3^-2),coma,const(adi9o),CRLF,
ent(22),FOR,id(I),igual,const_num(0),TO,const_num(2),CRLF,
ent(24),FOR,id(J),igual,const_num(0),TO,const_num(2),CRLF,
ent(25),LET,ide(C$),igual,const(Hola),CRLF,
ent(26),NEXT,id(J),CRLF,
ent(27),NEXT,id(I),CRLF,
ent(30),INPUT,const(Hola),coma,id(N),CRLF,
ent(35),DATA,const_num(3578344),CRLF,
ent(40),DEF,FX (B),igual,PARENTESIS_IQ,id(N),div,id(B),PARENTESIS_DER,mas,id(A),mas,id(C),CRLF,
ent(45),GOSUB,const_num(80),CRLF,
ent(46),LET,id(Z1),igual,const(Hola),CRLF,
ent(47),GOSUB,const_num(70),CRLF,
ent(48),GOSUB,const_num(60),CRLF,
ent(50),LET,ide(S$),igual,id(A),mas,const_num(2),CRLF,
ent(60),STOP,CRLF,
ent(65),READ,id(J),CRLF,
ent(70),GOTO,const_num(10),CRLF,
ent(75),READ,id(X),coma,id(D),CRLF,
ent(80),IF,id(A),distinto,PARENTESIS_IQ,id(D),mas,id(N),div,id(B),PARENTESIS_DER,mas,id(A),THEN,GOTO,const_num(60),CRLF,
ent(100),ON,PARENTESIS_IQ,id(A),mas,id(B),PARENTESIS_DER,GOTO,const_num(10),coma,ent(20),coma,ent(50),CRLF,
ent(110),RETURN,CRLF,
ent(115),RETURN,CRLF,
ent(117),RETURN,CRLF,
ent(130),PRINT,PARENTESIS_IQ,id(A),mas,id(B),PARENTESIS_DER,PUNTOYCOMA,const(Hola),coma,ide(S$),CRLF,
```

## Fichero 6:

```
10 DIM A(8), B(8,4)
20 DATA 3.15159, 24, 3,"hola", 3^-2, "adi9o"
30 INPUT "Hola", N
40 DEF FNX (B) = (N/B)+A+C
46 LET Z1 = "Hola"
47 GOSUB 70
50 LET S$ = A + 2
65 READ J
70 GOTO 800
75 READ X,D
80 IF A <> (D+N/B)+A THEN GOTO 60
100 ON (A+B) GOTO 10,20,50
110 RETURN
130 PRINT (A+B);"Hola", S$
190 END
```

Error semantico el operando no se encuentra definido como una variable C linea -> 4

Error semantico estas intentando guardar una operacion en un variable cadena 5

Error semantico estas intentando guardar una cadena en un variable numerica 7

Error semantico la linea para el GOTO no existe 800 linea: 9

Error semantico la linea para el GOTO no existe 60 linea: 11

\*\*\*\*\*ANALISIS FINALIZADO\*\*\*\*\*

Numero de errores sintacticos: 2

Numero de errores semanticos: 3

\*\*\*\*\* ANALIZADOR LEXICO \*\*\*\*\*

```
[ent(10),DIM,ide(A(8)),coma,ide(B(8,4)),CRLF,
ent(20),DATA,const_num(3.15159),coma,const_num(24),coma,const_num(3),coma,const(hola),coma,const_num(3^-2),coma,const(adi9o),CRLF,
ent(30),INPUT,const(Hola),coma,id(N),CRLF,
ent(40),DEF,FNX (B),igual,PARENTESIS_IZQ,id(N),div,id(B),PARENTESIS_DER,mas,id(A),mas,id(C),CRLF,
ent(46),LET,id(Z1),igual,const(Hola),CRLF,
ent(47),GOSUB,const_num(70),CRLF,
ent(50),LET,ide(S$),igual,id(A),mas,const_num(2),CRLF,
ent(65),READ,id(J),CRLF,
ent(70),GOTO,const_num(800),CRLF,
ent(75),READ,id(X),coma,id(D),CRLF,
ent(80),IF,id(A),distinto,PARENTESIS_IZQ,id(D),mas,id(N),div,id(B),PARENTESIS_DER,mas,id(A),THEN,GOTO,const_num(60),CRLF,
ent(100),ON,PARENTESIS_IZQ,id(A),mas,id(B),PARENTESIS_DER,GOTO,const_num(10),coma,ent(20),coma,ent(50),CRLF,
ent(110),RETURN,CRLF,
ent(130),PRINT,PARENTESIS_IZQ,id(A),mas,id(B),PARENTESIS_DER,PUNTOYCOMA,const(Hola),coma,ide(S$),CRLF,
ent(190),END,EOF]
```

Numero de errores lexicos: 0

## Fichero 7:

```
10 DIM A(8), B(8,4)
20 DATA 3.15159, 24, 3, "hola", 3^-2, "adi9o"
30 INPUT "Hola", N
40 DEF FNX (B) = (N/B)+A
46 LET Z1
47 GOSUB
50 LET S$ = "A + 2"
65 READ J
70 GOTO 65
75 READ X,D
80 IF A <> (D+N/B)+A THEN GOTO 50
100 ON (A+B) GOTO 10,20,50
110 RETURN
130 PRINT (A+B); "Hola", S$
190 END
```

Error de sintaxis línea 5 en o cerca de null  
Error de sintaxis línea 6 en o cerca de null  
Error semántico se esperaba una sentencia GOSUB para el return

\*\*\*\*\*ANALISIS FINALIZADO\*\*\*\*\*  
Numero de errores sintacticos: 2  
Numero de errores semanticos: 1

\*\*\*\*\* ANALIZADOR LEXICO \*\*\*\*\*

```
[ent(10),DIM,ide(A(8)),coma,ide(B(8,4)),CRLF,
ent(20),DATA,const_num(3.15159),coma,const_num(24),coma,const_num(3),coma,const(hola),coma,const_num(3^-2),coma,const(adi9o),CRLF,
ent(30),INPUT,const(Hola),coma,id(N),CRLF,
ent(40),DEF,FNX (B),igual,PARENTESIS_IZQ,id(N),div,id(B),PARENTESIS_DER,mas,id(A),CRLF,
ent(46),LET,id(Z1),CRLF,
ent(47),GOSUB,CRLF,
ent(50),LET,ide(S$),igual,const(A + 2),CRLF,
ent(65),READ,id(J),CRLF,
ent(70),GOTO,const_num(65),CRLF,
ent(75),READ,id(X),coma,id(D),CRLF,
ent(80),IF,id(A),distinto,PARENTESIS_IZQ,id(D),mas,id(N),div,id(B),PARENTESIS_DER,mas,id(A),THEN,GOTO,const_num(50),CRLF,
ent(100),ON,PARENTESIS_IZQ,id(A),mas,id(B),PARENTESIS_DER,GOTO,const_num(10),coma,ent(20),coma,ent(50),CRLF,
ent(110),RETURN,CRLF,
ent(130),PRINT,PARENTESIS_IZQ,id(A),mas,id(B),PARENTESIS_DER,PUNTOYCOMA,const(Hola),coma,ide(S$),CRLF,
ent(190),END,EOF]
```

Numero de errores lexicos: 0

## Fichero 8:

```
10 DIM A(8), B(8,4)
20 DATA 3.15159, 24, 3,"hola", 3^-2, "adi9o"
30 INPUT "Hola", N
40 DEF FNX (B) = (N/B)+A
46 LET Z1
47 GOSUB
50 LET S$ = "A + 2"
65 READ J
70 GOTO 65
75 READ X,D
80 IF A <> (D+N/B)+A THEN GOTO 50
100 ON (A+B) GOTO 10,20,50
110 RETURN
130 PRINT (A+B);"Hola", S$
190 END
```

Error de sintaxis linea 5 en o cerca de null  
Error de sintaxis linea 6 en o cerca de null  
Error semantico se esperaba una sentencia GOSUB para el return

\*\*\*\*\*ANALISIS FINALIZADO\*\*\*\*\*  
Numero de errores sintacticos: 2  
Numero de errores semanticos: 1

\*\*\*\*\* ANALIZADOR LEXICO \*\*\*\*\*

```
[ent(10),DIM,ide(A(8)),coma,ide(B(8,4)),CRLF,
ent(20),DATA,const_num(3.15159),coma,const_num(24),coma,const_num(3),coma,const(hola),coma,const_num(3^-2),coma,const(adi9o),CRLF,
ent(30),INPUT,const(Hola),coma,id(N),CRLF,
ent(40),DEF,FNX (B),igual,PARENTESIS_IQZQ,id(N),div,id(B),PARENTESIS_DER,mas,id(A),CRLF,
ent(46),LET,id(Z1),CRLF,
ent(47),GOSUB,CRLF,
ent(50),LET,ide(S$),igual,const(A + 2),CRLF,
ent(65),READ,id(J),CRLF,
ent(70),GOTO,const_num(65),CRLF,
ent(75),READ,id(X),coma,id(D),CRLF,
ent(80),IF,id(A),distinto,PARENTESIS_IQZQ,id(D),mas,id(N),div,id(B),PARENTESIS_DER,mas,id(A),THEN,GOTO,const_num(50),CRLF,
ent(100),ON,PARENTESIS_IQZQ,id(A),mas,id(B),PARENTESIS_DER,GOTO,const_num(10),coma,ent(20),coma,ent(50),CRLF,
ent(110),RETURN,CRLF,
ent(130),PRINT,PARENTESIS_IQZQ,id(A),mas,id(B),PARENTESIS_DER,PUNTOYCOMA,const(Hola),coma,ide(S$),CRLF,
ent(190),END,EOF]
```

Numero de errores lexicos: 0

## Fichero 9:

```
10 DIM A(8), B(8,4)
20 DATA 3.15159, 24, 3,"hola", 3^-2, "adi9o"
30 INPUT "Hola", N
40 DEF FNX (B) = (N/B)+A
46 LET Z1 = 2
47 GOSUB 70
50 LET S$ = "A + 2"
65 READ J
70 GOTO 65
75 READ X,D,F
80 IF A <> (D+N/B)+A THEN GOTO 50
90 RETURN
100 ON (A+B) GOTO 10,20,50
130 PRINT (A+B);"Hola", S$
190 END
```

Error semantico estas intentado guardar una cadena en una variable numerica F linea -> 9

\*\*\*\*\*ANALISIS FINALIZADO\*\*\*\*\*

Numero de errores sintacticos: 0

Numero de errores semanticos: 1

\*\*\*\*\* ANALIZADOR LEXICO \*\*\*\*\*

```
[ent(10),DIM,ide(A(8)),coma,ide(B(8,4)),CRLF,
ent(20),DATA,const_num(3.15159),coma,const_num(24),coma,const_num(3),coma,const(hola),coma,const_num(3^-2),coma,const(adi9o),CRLF,
ent(30),INPUT,const(Hola),coma,id(N),CRLF,
ent(40),DEF,FNX (B),igual,PARENTESIS_IQZ,id(N),div,id(B),PARENTESIS_DER,mas,id(A),CRLF,
ent(46),LET,id(Z1),igual,const_num(2),CRLF,
ent(47),GOSUB,const_num(70),CRLF,
ent(50),LET,ide(S$),igual,const(A + 2),CRLF,
ent(65),READ,id(J),CRLF,
ent(70),GOTO,const_num(65),CRLF,
ent(75),READ,id(X),coma,id(D),coma,id(F),CRLF,
ent(80),IF,id(A),distinto,PARENTESIS_IQZ,id(D),mas,id(N),div,id(B),PARENTESIS_DER,mas,id(A),THEN,GOTO,const_num(50),CRLF,
ent(90),RETURN,CRLF,
ent(100),ON,PARENTESIS_IQZ,id(A),mas,id(B),PARENTESIS_DER,GOTO,const_num(10),coma,ent(20),coma,ent(50),CRLF,
ent(130),PRINT,PARENTESIS_IQZ,id(A),mas,id(B),PARENTESIS_DER,PUNTOYCOMA,const(Hola),coma,ide(S$),CRLF,
ent(190),END,EOF]
```

Numero de errores lexicos: 0

## Fichero 10:

```
10 DIM A(8), B(8,4)
15 DIM B(3)
20 DATA 3.15159, 24, 3,"hola", 3^-2, "adi9o"
30 INPUT "Hola", N
40 DEF FN (B) = (N/B)+A
46 LET Z1 = 2
47 GOSUB 70
50 LET S$ = "A + 2"
65 READ J
70 GOTO 65
75 READ X,D,F$, G$, H$, M
80 IF A <> (D+N/B)+A THEN GOTO 50
90 RETURN
100 ON (A+B) GOTO 10,20,50
130 PRINT (A+B);"Hola", S$
190 END
```

Error semantico al declarar la variable ya existe una con ese nombre B(3) línea -> 1  
Error semantico hay demasiadas variables declaras en el READ  
Error semantico hay demasiadas variables declaras en el READ

\*\*\*\*\*ANALISIS FINALIZADO\*\*\*\*\*  
Numero de errores sintacticos: 0  
Numero de errores semanticos: 3

\*\*\*\*\* ANALIZADOR LEXICO \*\*\*\*\*

```
[ent(10),DIM,ide(A(8)),coma,ide(B(8,4)),CRLF,
ent(15),DIM,ide(B(3)),CRLF,
ent(20),DATA,const_num(3.15159),coma,const_num(24),coma,const_num(3),coma,const(hola),coma,const_num(3^-2),coma,const(adi9o),CRLF,
ent(30),INPUT,const(Hola),coma,id(N),CRLF,
ent(40),DEF,FN (B),igual,PARENTESIS_IZQ,id(N),div,id(B),PARENTESIS_DER,mas,id(A),CRLF,
ent(46),LET,id(Z1),igual,const_num(2),CRLF,
ent(47),GOSUB,const_num(70),CRLF,
ent(50),LET,ide(S$),igual,const(A + 2),CRLF,
ent(65),READ,id(J),CRLF,
ent(70),GOTO,const_num(65),CRLF,
ent(75),READ,id(X),coma,id(D),coma,ide(F$),coma,ide(G$),coma,ide(H$),coma,id(M),CRLF,
ent(80),IF,id(A),distinto,PARENTESIS_IZQ,id(D),mas,id(N),div,id(B),PARENTESIS_DER,mas,id(A),THEN,GOTO,const_num(50),CRLF,
ent(90),RETURN,CRLF,
ent(100),ON,PARENTESIS_IZQ,id(A),mas,id(B),PARENTESIS_DER,GOTO,const_num(10),coma,ent(20),coma,ent(50),CRLF,
ent(130),PRINT,PARENTESIS_IZQ,id(A),mas,id(B),PARENTESIS_DER,PUNTOYCOMA,const(Hola),coma,ide(S$),CRLF,
ent(190),END,EOF]
```

Numero de errores lexicos: 0