



ESCUELA TÉCNICA  
SUPERIOR DE INGENIEROS  
INFORMÁTICOS

UNIVERSIDAD POLITÉCNICA DE MADRID

---

# Procesadores de Lenguajes: Memoria del Proyecto

---

ANALIZADOR LÉXICO Y TABLA DE SÍMBOLOS

Jose Luis Prado Sierra - 220070  
Alejandro Gragera Serradilla - 22M043  
Antonio Bielza Díez - 22M049

# 1. Analizador Léxico

Durante el desarrollo del Analizador Léxico hemos descrito una serie de objetos y estructuras matemáticas indispensables para que su diseño, desarrollo y función final cumplan con las expectativas de un Procesador de Lenguajes.

## 1.1. Tokens

Los tokens son duplas que se generan cuando el Analizador Léxico encuentra una concatenación de caracteres que identifica como válida.

Contienen la información necesaria para que las reciba el Analizador Sintáctico y se componen de un código que los identifica y un atributo opcional que puede servir para diferenciarlos de otros tokens con el mismo código o aportar información extra.

Hemos definido los siguientes tokens en función de las necesidades de nuestra práctica:

- Boolean: <BOOLEAN, >
- Else: <ELSE, >
- Float: <FLOAT, >
- Function: <FUNCTION, >
- If: <IF, >
- Int: <INT, >
- Let: <LET, >
- Read: <READ, >
- Return: <RETURN, >
- String: <STRING, >
- Void: <VOID, >
- Write: <WRITE, >
- Constante real: <REALCONST, >
- Constante entera: <INTCONST, >
- Cadena: <STR, c\*"
- Identificador: <ID, posTS>
- Suma con asignación (+=): <PLUSEQ, >
- Igual (=): <EQ, >
- Coma (,): <COMMA, >
- Punto y coma (;): <SEMICOLON, >
- Paréntesis abierto (:): <OPPAR, >
- Paréntesis cerrado ()): <CLPAR, >
- Llave abierta ({): <OPBRA, >

- Llave cerrada (}): <CLBRA, >
- Suma (+): <SUM, >
- Y Lógico (&): <AND, >
- Menor (<): iMINORTHAN, >
- false: <FALSE, >
- true: <TRUE, >
- EOF: <EOF, >

Todos los tokens anteriores conforman todos los obligatorios, los específicos y los opcionales de la práctica.

## 1.2. Gramática

La gramática es la estructura matemática que define el lenguaje a generar, en nuestro caso es para una versión reducida de JS. Definir la gramática es una forma de estructurar por tanto el lenguaje y asegurarnos de que vamos a generar solo lo que queremos.

Hemos definido las siguientes reglas:

$$S \rightarrow delS \mid +A \mid \&B \mid 'C \mid dD \mid lF \mid \_F \mid /G \mid = \mid , \mid ; \mid < \mid ( \mid ) \mid \{ \mid \}$$

$$A \rightarrow = \mid \lambda$$

$$B \rightarrow \&$$

$$C \rightarrow cC \mid '$$

$$D \rightarrow dD \mid .E \mid \lambda$$

$$E \rightarrow dE \mid d$$

$$F \rightarrow lF \mid dF \mid \_F \mid \lambda$$

$$G \rightarrow /H$$

$$H \rightarrow c_2H \mid \backslash nS$$

Los conjuntos definidos para el desarrollo de la gramática y del resto de la práctica son:

Conjunto l: representa cualquier letra.

Conjunto d: representa cualquier dígito.

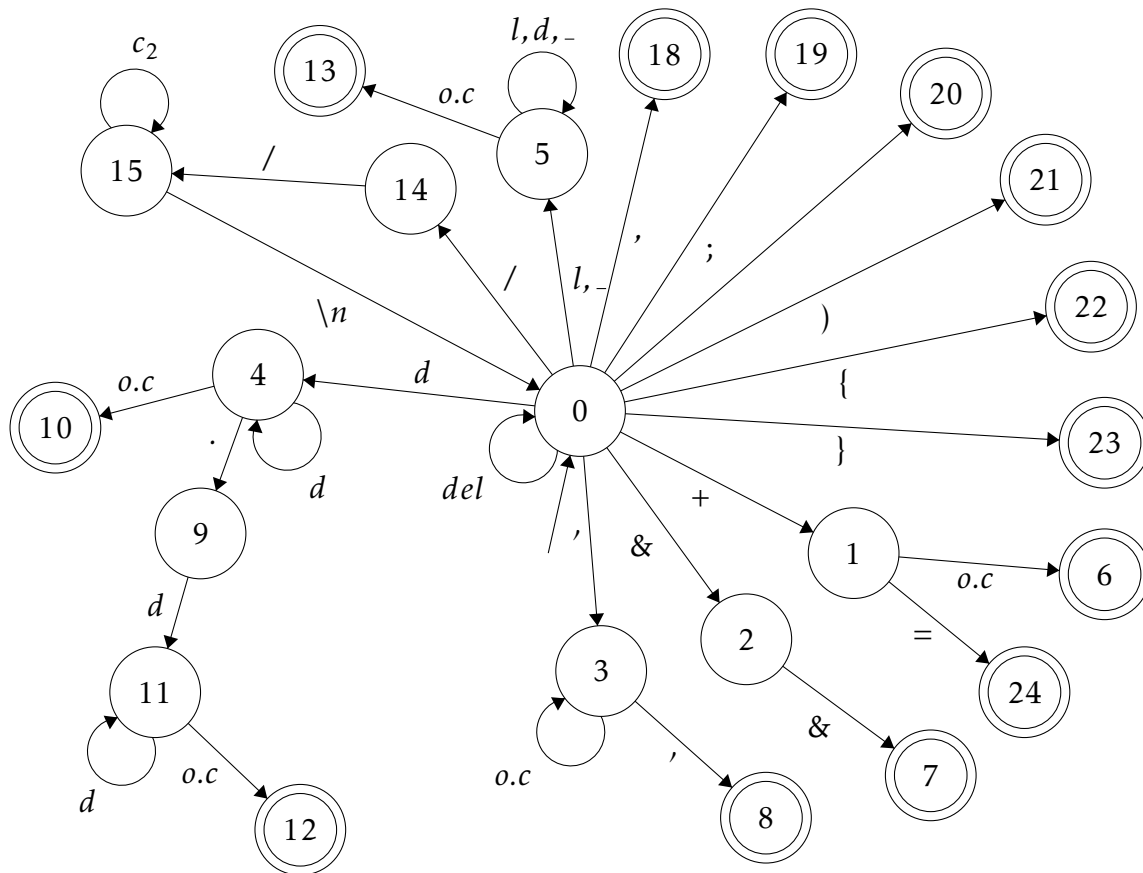
Conjunto c: representa cualquier caracter.

Conjunto  $c_2$ : representa cualquier caracter sin  $\backslash n$ .

### 1.3. Autómata

Un autómata es otro tipo de estructura matemática capaz de, al contrario que una gramática que se centra en generar un lenguaje, un autómata se encarga de comprenderlo.

El autómata que hemos planteado capaz de entender todas las palabras válidas de nuestro lenguaje es el siguiente:



El autómata tiene que ir acompañado de una serie de acciones que realizar mientras se recorre para poder generar los tokens mientras vamos leyendo el archivo, que vamos a plantear en el siguiente punto.

Cuando aparece una transición o.c significa que es cualquier otro caracter distinto a las demás transiciones salientes del vértice.

## 1.4. Acciones Semánticas

Las acciones semánticas son una serie de acciones que se realizan entre las transiciones del autómata que permiten realizar diferentes funciones como por ejemplo ir generando un string para cuando detectamos una cadena.

Las acciones semánticas que hemos definido para nuestro proyecto son:

- 0:0 Leer
- 0:18 Leer, gentoken(<COMMA, >)
- 0:19 Leer, gentoken(<SEMICOLON, >)
- 0:20 Leer, gentoken(<OPPAR, >)
- 0:21 Leer, gentoken(<CLPAR, >)
- 0:22 Leer, gentoken(<OPBRA, >)
- 0:23 Leer, gentoken(<CLBRA, >)
- 0:1 Leer
- 1:6 gentoken(<SUM, >)
- 1:24 Leer, gentoken(<EQ, >)
- 0:2 Leer
- 2:7 Leer, gentoken(<AND, >)
- 0:3 Leer,  $str = ''$
- 3:3 Leer,  $str = str + c$
- 3:8 Leer, gentoken(<STRING, str>)
- 0:4 Leer,  $num = d$
- 4:4 Leer,  $num = num * 10 + d$
- 4:10 gentoken(<INTCONST, num>)
- 4:9 Leer,  $cont = 1$
- 9:11 Leer,  $num = num * 10 + d, cont ++$
- 11:11 Leer,  $num = num * 10 + d, cont ++$
- 11:12  $num = num * 10^{-cont}$ , gentoken(<REALCONST, num>)
- 0:14 Leer
- 0:15 Leer
- 15:15 Leer
- 15:0 Leer

- 0:5 Leer,  $id = c$
- 5:5 Leer,  $id = id + c$
- 5:13

```
if(id in palabrasReservadas)
    postTS = insertTS(id)
    gentoken(<toUpper(id), postTS>)
else
    postTS = insertTS(id)
    gentoken(<ID, postTS>)
```

## 1.5. Errores

Todo buen procesador de lenguajes debe de avisar al usuario de si existe algún error y además es capaz de tratar todos los posibles errores para una ejecución robusta y comprensible.

Sin embargo para esta entrega no es necesario que sean descriptivos pero si han de ser reconocidos y clasificados.

Hemos detectado los siguientes casos capaces de generar errores:

- **Cadena mal cerrada:** Al iniciar una cadena si queda mal cerrada (Ej. 'Hola buenas tardes).
- **ID incorrecto:** Al generar o utilizar un ID si los caracteres que lo componen no son correctos.
- **Caracter no reconocido:** Si se recibe un caracter no válido en el estado inicial.
- **Operador lógico incorrecto:** Si se pone un operador lógico (como `——` o `&&`) de forma incorrecta (Ej. `—` o `&`).
- **Comentario de línea mal iniciado:** Al iniciar un comentario de línea incorrectamente (Ej. `/Hola buenas tardes`).
- **Entero demasiado grande:** Al declarar un entero mayor que 32767.
- **Real demasiado grande:** Al declarar un real mayor que 117549436.
- **Cadena demasiado grande:** Al declarar una cadena de más de 64 caracteres.

## 2. Tabla de Símbolos

La tabla de símbolos es una estructura de datos que contiene información adicional sobre los tokens cuyo atributo es su posición dentro de ella.

El diseño elegido es una pila de diccionarios en la cual cada diccionario contiene la información extra del token (actualmente vacía) y que, al ser insertados, se devuelve la posición que han recibido dentro de ella, para posteriormente generar el token correctamente.

En nuestro caso se asigna la posición dentro de la tabla de forma lineal, el primer elemento recibe la primera posición y la enésima inserción recibe la enésima posición.

### **3. Anexo**

#### **3.1. Casos Correctos**

3.1.1. Caso A

3.1.2. Caso B

3.1.3. Caso C

#### **3.2. Casos Erróneos**

3.2.1. Caso A

3.2.2. Caso B

3.2.3. Caso C



**Figura 1:** A goose.