

# PDL: Práctica Procesador

Procesador JavaScript-PDL

Serrano,Arrese Francisco Javier  
Cañibano,Lopez Alberto  
Vallejo,Collados Jesús

Grupo 14  
Procesadores de Lenguajes  
Universidad Politécnica de Madrid  
Curso 2020-2021

## PALABRAS RESERVADAS

alert  
 boolean  
 else  
 function  
 if  
 input  
 let  
 number  
 return  
 string  
 while  
 false  
 true  
 do

## TOKENS

alert	<reservedWord, alert >
boolean	<reservedWord, boolean >
else	<reservedWord, else >
function	<reservedWord, function >
if	<reservedWord, if >
input	<reservedWord, input >
let	<reservedWord, let, >
number	<reservedWord, number >
return	<reservedWord, return >
string	<reservedWord, string >
while	<reservedWord, while >
false	<reservedWord, false >
true	<reservedWord, true >
do	<reservedWord, do >
Autoincremento (++)	<autoIncOp, autoInc >
constante entera	<wholeConst, Número>
Posición(Número)	<chain, Posición(Número) >
Identificador	<ID, Número>
=	<asigOp, equal >
,	<separator, colon >
;	<separator, semicolon>
(	<separator,openPar >
)	<separator,closePar >
}	<separator,openBraque >
{	<separator,closeBraque >
Suma (+)	<aritOp,plus>
Resta (-)	<aritOp,minus >
Y lógico (&&)	<logOp,and >
Negación (!)	<logOp,not >
Distinto (!=)	<relOp,notEquals >
Igual (==)	<relOp,equals >

## GRAMÁTICA

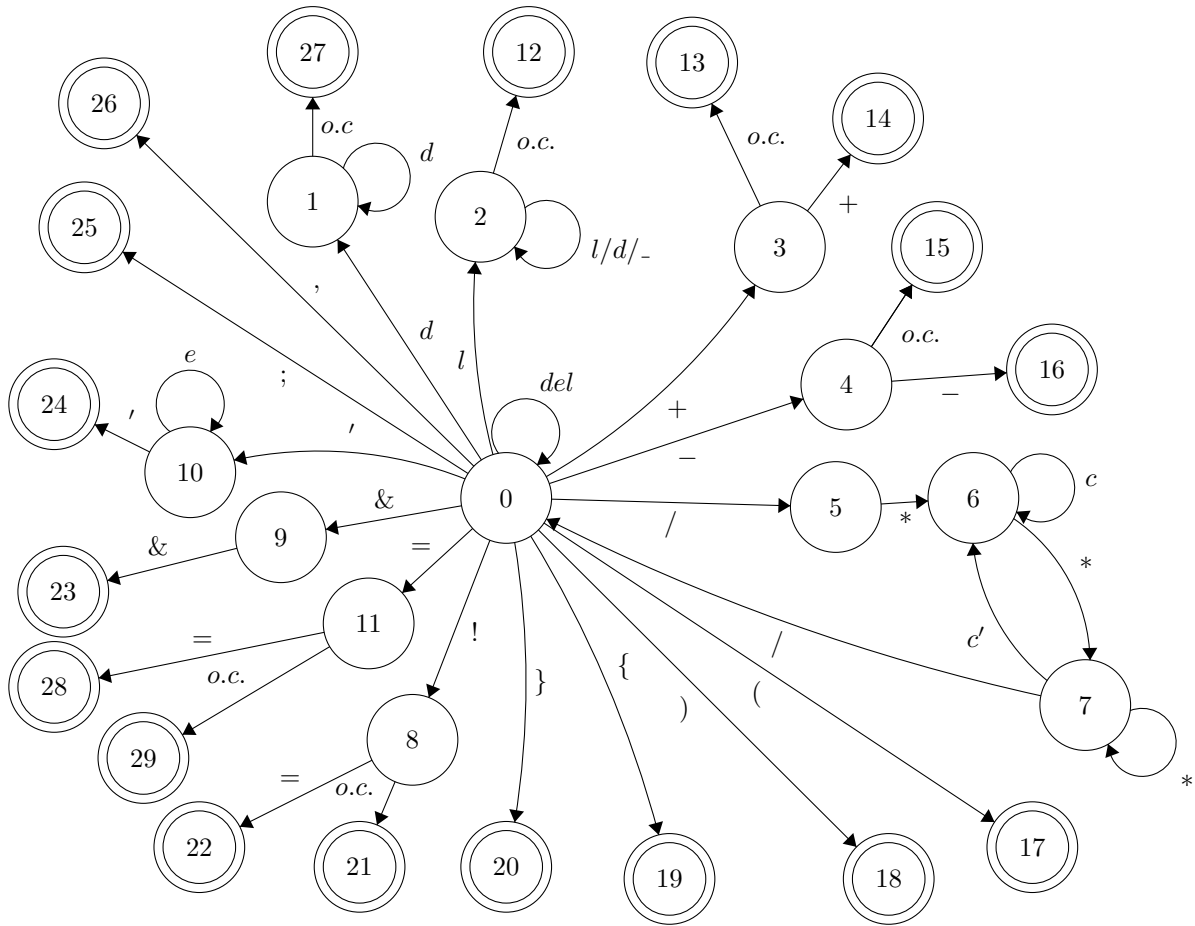
```
S:--> del S | dA | lB | +C | -D | /F | ( | ) | { | } | =G | !G | , | ; | &H | 'J
A:--> dA | lambda
B:--> lB | dB | _B | lambda
C:--> + | lambda
D:--> - | lambda
F:--> *E
E:--> cE | *I | /S
I:--> *I | c'E
G:--> = | lambda
H:--> &
J:--> eJ | '
```

## LEYENDA

```
d -- digito
l -- letra minuscula
del -- delimitador( blanco, tab, EOL)
c -- caracteres - (*)
c' -- caracteres - (*/)
e -- caracteres - (')
```

# AUTÓMATA FINITO DETERMINISTA

d	Dígito
l	Letra
del	Delimitador
c	Caracteres -{*}
c'	Caracteres -{*/}
e	Caracteres -{'}
o.c.	Otro Caracter



## ACCIONES SEMÁNTICAS

**Leer:** Se lee en todos los estados menos en los que pone o.c.

**Errores:** Cualquier transición no declarada dará error.

### Caso 0-0:

Leer Caracter

### Caso 0-1:

número=valor(d), Leer Caracter

### Caso 1-1:

número=número\*10+d, Leer Caracter

### Caso 1-27:

```
if número<215
    GenerarToken(wholeConst,número)
else
    Error ("Numero fuera de rango, 16 bits")
```

### Caso 0-2:

lexema=1, Leer Caracter

### Caso 2-2:

lexema=lexema+(1|d|'\_''), Leer Caracter

### Caso 2-12:

```
if lexema == reservedWord
    GenerarToken(reservedWord,lexema)
else
    if(not enTablaDeSimbolos(lexema)):
        insertarEnTablaSimbolos(lexema)
    GenerarToken(ID,posicionTablaSimbolos)
```

### Caso 0-3:

Leer Caracter

### Caso 3-13:

GenerarToken(aritOp,plus)

### Caso 3-14:

GenerarToken(autoIncOp,autoinc)

### Caso 0-4:

Leer Caracter

### Caso 4-15:

GenerarToken(aritOp,minus)

**Caso 4-16:**

GenerarToken(autoDecOp,autoDec)

**Caso 0-5:**

Leer Character

**Caso 5-6:**

Leer Character

**Caso 6-6:**

Leer Character

**Caso 6-7:**

Leer Character

**Caso 7-6:**

Leer Character

**Caso 7-0:**

Leer Character

**Caso 0-26:**

GenerarToken(separator,colon)

**Caso 0-25:**

GenerarToken(separator,semicolon)

**Caso 0-20:**

GenerarToken(separator,closeBraq)

**Caso 0-19:**

GenerarToken(separator,openBraq)

**Caso 0-18:**

GenerarToken(separator,closePar)

**Caso 0-17:**

GenerarToken(separator,openPar)

**Caso 0-10:**

Leer Caracter,lexema = ', contador = 1

**Caso 10-10:**

lexema=lexema+siguienteCaracter

**Caso 10-24:**

GenerarToken(chain,lexema)

**Caso 0-8:**

Leer Caracter

**Caso 8-21:**

GenerarToken(logOp,not)

**Caso 8-22:**

GenerarToken(relOp,notEquals)

**Caso 0-11:**

Leer Caracter

**Caso 11-28:**

GenerarToken(relOp,equals)

**Caso 11-29:**

GenerarToken(asigOp,equal)



**Caso 0-9:**

Leer Character

**Caso 9-23:**

```
if siguienteCaracter == '&'
    GenerarToken(logOp,and)
else
    Error("Syntax error. && expected")
```

**TABLA DE SIMBOLOS**

El valor de los atributos y numero de tabla seran corregidos con el valor real mas adelante.

```
Contenido Tabla Símbolos # N :
* LEXEMA : 'x'
ATRIBUTOS :
+ tipo: unknown
+ despl: unknown
```

Prueba 1: *CASO CORRECTO*

Código:

```

number a = 1;
string pp = 'hola';

/* hola

disculpa*/
if (a && a) {
    a = 2;
}

```

Tokens:

```

<number,>
<ID,a>
<asigOp,equal>
<wholeConst,1>
<separator,semicolon>
<string,>
<ID,pp>
<asigOp,equal>
<chain,'hola'>

```

TS:

Contenido Tabla Símbolos # 0 :

```

* LEXEMA : 'a'
  ATRIBUTOS :
    + tipo: unknown
    + despl: unknown
* LEXEMA : 'pp'
  ATRIBUTOS :
    + tipo: unknown
    + despl: unknown

```

Errores

Prueba 2: *CASO CORRECTO*

Código:

```

function padre(c) {

    let b = c;
    b++;

    c - b

```

```

    return c
}

```

Tokens:

```

<function,>
<ID,padre>
<separator,openPar>
<ID,c>
<separator,closePar>
<separator,openBraq>
<let,>
<ID,b>
<asigOp,equal>
<ID,c>
<separator,semicolon>
<ID,b>
<autoIncOp,autoinc>
<separator,semicolon>
<ID,c>
<aritOp,minus>
<ID,b>
<return,>
<ID,c>
<separator,closeBraq>

```

TS:

```

Contenido Tabla Símbolos # 0 :
* LEXEMA : 'padre'
  ATRIBUTOS :
    + tipo: unknown
    + Despl: -1
* LEXEMA : 'c'
  ATRIBUTOS :
    + tipo: unknown
    + Despl: -1
* LEXEMA : 'b'
  ATRIBUTOS :
    + tipo: unknown
    + Despl: -1

```

Errores

Prueba 3: *CASO CORRECTO*

Código:

```

boolean verdadero = true;
boolean grupo14 = true;
boolean aprobado = true;
do {

```

```

verdadero = false;
if (true) {
    verdadero = true
}

} while (grupo14 = aprobado)

```

Tokens:

```

<boolean,>
<ID,verdadero>
<asigOp,equal>
<true,>
<separator,semicolon>
<boolean,>
<ID,grupo14>
<asigOp,equal>
<true,>
<separator,semicolon>
<boolean,>
<ID,aprovado>
<asigOp,equal>
<true,>
<separator,semicolon>
<do,>
<separator,openBraq>
<ID,verdadero>
<asigOp,equal>
<false,>
<separator,semicolon>
<if,>
<separator,openPar>
<true,>
<separator,closePar>
<separator,openBraq>
<ID,verdadero>
<asigOp,equal>
<true,>
<separator,closeBraq>
<separator,closeBraq>
<while,>
<separator,openPar>
<ID,grupo14>
<asigOp,equal>
<ID,aprovado>
<separator,closePar>

```

TS:

```

Contenido Tabla Símbolos # 0 :
* LEXEMA : 'verdadero'

```

```

    ATRIBUTOS :
        + tipo: unknown
        + Despl: -1
* LEXEMA : 'grupo14'
    ATRIBUTOS :
        + tipo: unknown
        + Despl: -1
* LEXEMA : 'aprovado'
    ATRIBUTOS :
        + tipo: unknown
        + Despl: -1

```

Errores:

Prueba 4: *CASO INCORRECTO*

Código:

```

    number 1a = 1
string pp = 'hola
numbe;
if (hola
    else {

```

&

Tokens:

```

<number,>
<wholeConst,1>
<ID,a>
<asigOp,equal>
<wholeConst,1>
<string,>
<ID,pp>
<asigOp,equal>
<ID,numbe>
<separator,semicolon>
<if,>
<separator,openPar>
<ID,hola>
<else,>
<separator,openBraq>

```

TS:

```

Contenido Tabla Símbolos # 0 :
* LEXEMA : 'a'

```

```

    ATRIBUTOS :
+ tipo: unknown
+ Despl: -1
* LEXEMA : 'pp'
    ATRIBUTOS :
+ tipo: unknown
+ Despl: -1
* LEXEMA : 'numbe'
    ATRIBUTOS :
+ tipo: unknown
+ Despl: -1
* LEXEMA : 'hola'
    ATRIBUTOS :
+ tipo: unknown
+ Despl: -1

```

Errores:

```

++ Error: ' cadena no se cierra en ningun momento,abierto en caracter: 13 ,linea: 2
++ Error: & esta solo en caracter: 1 ,linea: 8

```

Prueba 5: *CASO INCORRECTO*

Código:

```

1manolo == >

- -

'Esto esta escrito un sabado por la tarde

/*Sin embargo
ha sido un poco tedioso

```

Tokens:

```

<wholeConst,1>
<ID,manolo>
<relOp,equals>
<aritOp,minus>
<aritOp,minus>

```

TS:

```

Contenido Tabla Símbolos # 0 :
* LEXEMA : 'manolo'
  ATRIBUTOS :
    + tipo: unknown
    + Despl: -1

```

Errores:

```
++ Error: Caracter no reconocido:[>] en caracter: 11 ,linea: 1
++ Error: ' cadena no se cierra en ningun momento,abierto en caracter: 5 ,linea: 6
++ Error: /* comentario en bloque no se cierra
```

Prueba 6: *CASO INCORRECTO*

Código:

```
&
!!
===
else
if
{{
    ((
```

Tokens:

```
<logOp,not>
<logOp,not>
<relOp,equals>
<asigOp,equal>
<else,>
<if,>
<separator,openBraqu>
<separator,openBraqu>
<separator,openPar>
<separator,openPar>
```

TS:

Errores:

```
++ Error: & esta solo en caracter: 1 ,linea: 1
```

Expresiones

```
-----  
E -> E && R | R  
R -> R == U | R != U | U  
U -> U + V | U - V | V  
V -> ID | ( E ) | ID ( L ) | ENT | CAD | TRUE | FALSE | !V | V++
```

Sentencias Simples

```
-----  
S -> ID = E ;  
S -> ID ( L ) ;  
S -> ALERT ( E ) ;  
S -> INPUT ( ID ) ;  
S -> RETURN X ;
```

```
L -> E Q | lambda  
Q -> , E Q | lambda  
X -> E | lambda
```

Sentencias Compuestas

```
-----  
B -> IF ( E ) S  
B -> LET T ID  
B -> S  
B -> DO { C } WHILE ( E );
```

```
T -> NUMBER | BOOLEAN | STRING
```

Declaraciones de Funciones

```
-----  
F -> FUNCTION H ID ( A ) { C }
```

```
H -> T | lambda  
A -> T ID K | lambda  
K -> , T ID K | lambda  
C -> B C | lambda
```

Estructura de un programa = Axioma

```
-----  
P -> B P  
P -> F P  
P -> lambda
```