# Memoria de Procesador de Lenguajes de JS-PDL (G136)

## Index



## Autores


| Nombre | Matricula | Correo |
| --- | --- | --- |
| Alvaro Cabo | c200172 | alvaro.cabo@alumnos.upm.es |
| Oussama El Hatifi | c200359 | o.elhatifi@alumnos.upm.es |

## Opciones del Grupo

Las opciones obligatorias para el grupo 98 son:

- Sentencia repetitiva (do-while)
- Pre-auto-incremento (++ como prefijo)
- Comentario de bloque (/**/)
- Con comillas dobles (" ")
- Descendente Recursivo

Los elementos opcionales seleccionados por el grupo son*:

- Operador Aritmético %
- Operador Lógico &&
- Operador Relacional >

# Analizador léxico`

## Tokens

<TypeBool,- >
<LoopDo,- >
<FunID,- >
<CondIf,- >
<ResIn,- >
<TypeInt,- >
<ResLet,- >
<ResPrint,- >
<Return,- >
<TypeString,- >
<LoopWhile,- >

<CteInt,num>
<Cad,-”c*”>
<TokT,>
<TokF,>
<ID,num >

<AsValue,- >
<Com,- >
<SemCol,- >
<ParOpen,- >
<ParClose,- >
<KeyOpen,- >
<KeyClose,- >

<MOD,- >
<AND,- >
<GT,- >

<Teof,>

## Gramática Regular

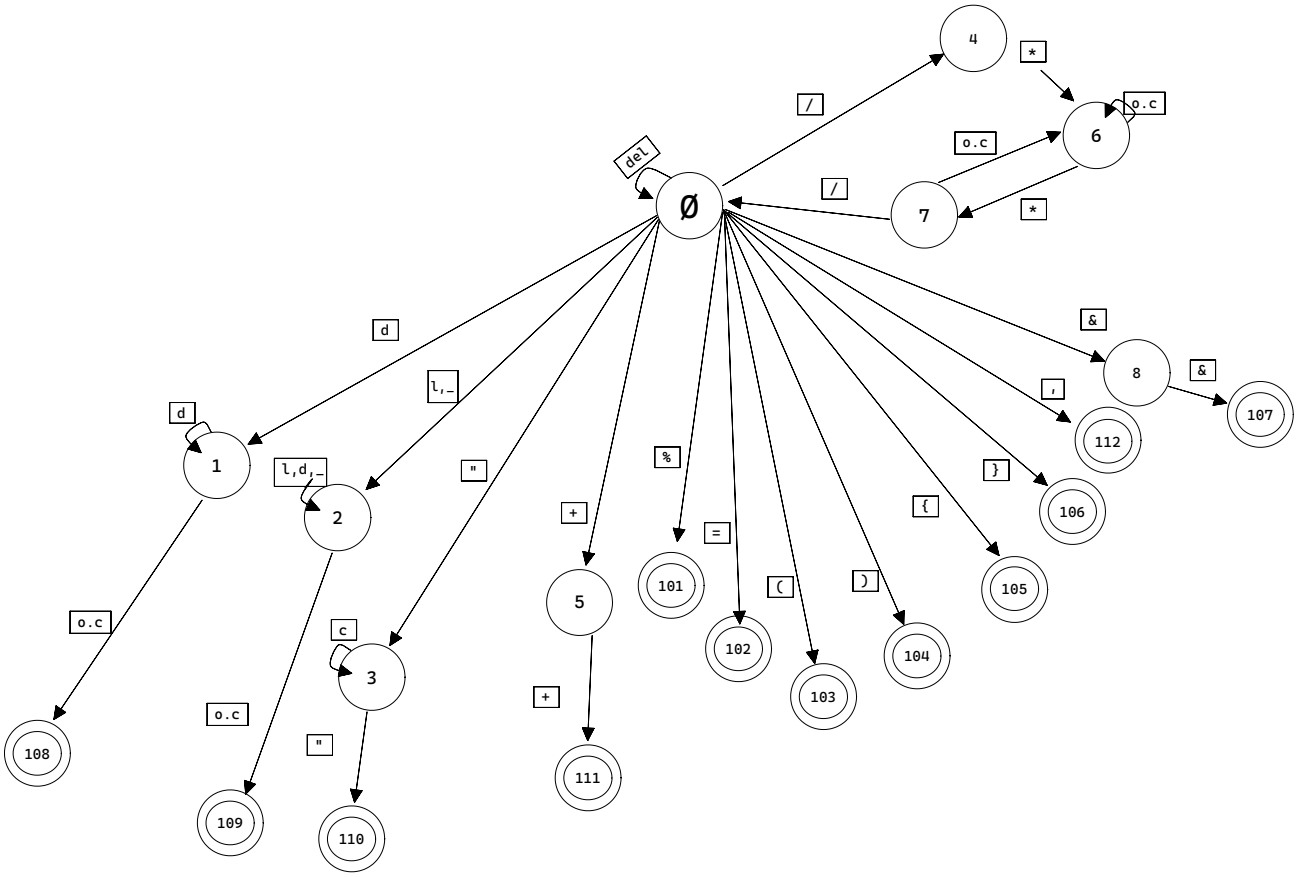
```
S: del S | dA | lB | "C | /D | +E | &F | = | , | ( | ) | { | } | %
A: dA | λ              → cte_entera
B: lB | dB | λ         → ID
C: cC | "              → cadena

D:  *D'
D': λD' | *D"          → Comentario
D": /S

E: + B                 → Pre-Auto-Incremento
F: &                   → Operadores Lógicos: Y lógico ( && )
```

Elementos Validos

```
d:[0..9]
l: ∀ ascci / {Control}
del
o.c → si cambia el car
```

λ-transición a E.F

## AFD




## Acciones semánticas

Esta sección la planteamos con una presentación distinta a la vista en clase, ya que creemos que utilizando una sintaxis estilo pseudocódigo (con varibles globales, funciones y todo lo que incluye la estructura de un programa) podemos plasmar la idea a implementar de una forma más gráfica

## VARIABLES GLOBALES

- STR_MAX_SIZE= 64 (car)

- MAX_POSSIBLE_INT = 32767

## FUNCIONES AUXILIARES

- **NEXT():** Lee el siguiente carácter del fichero fuente. car := leer()
  - **Transiciones:** Todas las que no implican una λ-transición (Representadas por o.c)
  - **Implementación:** Inicializa car <- next() con el caracter recién leido del texto fuente
- **CONCAT(car c):** Concatena el car con el lexema.
  - Se realiza después de Next() para utilizar el car recién leido
    - **Transiciones:** Lo vamos a utilizar en 3 contextos en nuestro autómata:
      - Comentarios: {0:7,7:6,6:4}
      - Identificadores: {0:2, 2:2}
      - Cadenas: {3:3}
- **VALOR(car c):** Similar a la función *atoi(char c)* de C, obtiene el valor numérico de un carácter leido
  - **Transiciones:** {0:1, 1:1}
- ERROR (int code):Devuelve error si se da cualquier caso no contemplado por el programa. Cuenta con algunos tipos de errores definidos:
  - STRING_OVER_MAX_LEGTH: Si se intenta contatenar car con un lexema con size == STR_MAX_SIZE
  - INT_OUT_OF_BOUNDS: Si se intenta generar un token cte_entera con un valor > 32767
- **GENTOKEN(type, @nullable(value)):** Se encarga de generar el token
  - **Funciones auxiliares:**
    - BuscaTS(lex): Comprueba si el lexema es un identificador ya insertado en la tabla de símbolos, devolviendo su posición.
      En caso contrario: NULL

    - AddTS(lex, scope): Inserta un identificador en la tabla de símbolos en la tabla indicada por scope (Global o local)

- **Variables auxiliares**
    - Bool Scope -> indica si se trata de una declaracion, global = true y local = false
    - DirToken [][]:=diccionario de cod_token para aquellos que no necesiten un tratamiento especial (cadenas, nums e IDs)
    - ResWords [][]:= diccionario de cod_token para **palabras reservadas**

DirToken

| Lex | Cod_token |
|---|---|
| { | KeyOpen |
| } | KeyClose |
| ( | ParOpen |
| ) | ParClose |
| = | AsValue |
| , | Com |
| ; | SemiCol |
| ++ | AutoSum |
| % | MOD |
| > | GT |
| && | AND |

ResWords

| Lexema | Cod_token |
|---|---|
| do | LoopDo |
| while | LoopWhile |
| boolean | TypeBool |
| int | TypeInt |
| string | TypeString |
| function | FunID |
| let | ResLet |
| input | ResInput |
| print | ResPrint |
| return | ResReturn |
| if | CondIf |

## Pseudocódigo que genera tokens sin valor

Esta es la casuística para tokens que no solo están compuestos de un key sin un valor variable (Todos menos cad, ID y num)

```
//CASE: palabra reservada
if ( lex in ResWords[][] )
  genToken( lex , - )
else { //CASE: identificador
    p <- buscaTS(lex)
    if ( p != null )
      genToken(id, p)
    else {
      p <- addTS(lex, scope)
```

```
          genToken(id, p) //<ID, puntero en la tabla>
        }
    }
```

### Secuencia de generación de tokes con valor

Este tipo de tokens no se recogen en el DirToken ya que necesitan utilizar la funciones auxiliares anteriormente documentadas

| Tipo | Transiciones | Acción |
|---|---|---|
| **CteInt** | 0->1 | num = valor(car) |
| | 1->1 | num = num*10+valor(car) |
| | 1->108 | if (num < 32768) -> genToken(cteEnt, num) else error(2) |
| **Cad** | 0->3 | lex_init |
| | 3->3 | lex=lex+car |
| | 3->110 | if (len(lex) < ) -> genToken(cad, lex) else error(2) |

# Analizador Sintáctico

La opción del grupo es **Análisis recursivo descendente**

## Gramática

```
Terminales = { lambda eof  let  id  ;  if  (  )  {  }  while % do else
            function return input print int boolean string and >  =
            cad num  ++  , &&  true false  }
NoTerminales = {  START  SENA  CTE  INC  EXP    EXPX  VALUE  XPX  ASIGN  DECL
DECLX  TD  TDX  INOUT
                WILE  SENB  BODY  IFX  FCALL  FCALLX  RX  IDX IFAX   FUN  PARM
PARMX  }

Axioma = START

Producciones = {

START -> SENA START ////1
START -> FUN START ////2
START -> eof ////3

CTE -> cad ////4
CTE -> num ////5
CTE -> true ////6
CTE -> false ////7
```

```
INC -> ++ id //// 8

EXP -> VALUE EXPX ////9
EXP -> INC EXPX ////10

EXPX -> > EXP  ////11
EXPX -> &&  EXP ////12
EXPX -> % EXP ////13
EXPX -> lambda ////14

VALUE -> id XPX ////15
VALUE -> CTE    ////16
VALUE -> ( EXP ) ////17

XPX -> ( FCALL ) ////18
XPX -> lambda   ////19

ASIGN ->  = EXP ; ////factorizable ? //// 20

DECL -> let id TD DECLX ; //// 21

DECLX -> ASIGN //// 22
DECLX -> lambda //// 23

TD -> int //// 24
TD -> string //// 25
TD -> boolean   //// 26

TDX -> TD   //// 27
TDX -> lambda   //// 28

INOUT -> print EXP ;     //// 29
INOUT -> input id ;     //// 30

SENA -> IFX     //// 31
SENA -> DECL    //// 32
SENA -> do { BODY } WILE    //// 33
WILE -> while ( EXP ) ;     //// 34
SENA -> SENB                //// 35

SENB -> id IDX              //// 36
SENB -> INOUT               //// 37
SENB -> return RX ;         //// 38
SENB -> INC ;               //// 39

BODY -> SENA BODY           //// 40
BODY -> lambda              //// 41

IFX -> if ( EXP ) IFAX ;    //// 42
```

```
IFAX -> SENB    //// 43
IFAX -> { SENB } //// 44 Esto en verdad no hace falta


FCALL -> EXP FCALLX     //// 45
FCALL -> lambda         //// 46
FCALLX -> , EXP FCALLX  //// 47
FCALLX -> lambda         //// 48

RX -> EXP               //// 49
RX -> lambda            //// 50

IDX -> ASIGN          //// 51
IDX -> ( FCALL ) ;      //// 52

FUN -> function id TDX ( PARM ) { BODY }    //// 53

PARM -> TD id PARMX     //// 54
PARM -> lambda      //// 55
PARMX -> , TD id PARMX  //// 56
PARMX -> lambda     //// 57

}
```

## Demostración gramática

Para realizar este parser recursivo descendente, es necesario que la gramática se encuentre en forma LL1 para poder operar con 1 token por iteración.

Vamos a proceder a demostrar que efectivamente nuestra gramática está en forma LL1

1. Tablas First & Follow
   Para depurar nuestra gramática, hemos utilizado esta herramienta online que devuelve ambos sets formateados en una tabla comparativa

| | eof | cad | num | true | false | ++ | id | > | && | % | lambda | ( | ) | = | ; | let |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| START | + | – | – | – | – | + | + | – | – | – | – | – | – | – | – | + |
| CTE | – | + | + | + | + | – | – | – | – | – | – | – | – | – | – | – |
| INC | – | – | – | – | – | + | – | – | – | – | – | – | – | – | – | – |
| EXP | – | + | + | + | + | + | + | – | – | – | – | + | – | – | – | – |
| EXPX | – | – | – | – | – | – | – | + | + | + | + | – | – | – | – | – |
| VALUE | – | + | + | + | + | – | + | – | – | – | – | + | – | – | – | – |

| | eof | cad | num | true | false | ++ | id | > | && | % | lambda | ( | ) | = | ; | let |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| XPX | – | – | – | – | – | – | – | – | – | – | + | + | – | – | – | – |
| ASIGN | – | – | – | – | – | – | – | – | – | – | – | – | – | + | – | – |
| DECL | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | + |
| DECLX | – | – | – | – | – | – | – | – | – | – | + | – | – | + | – | – |
| TD | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – |
| TDX | – | – | – | – | – | – | – | – | – | – | + | – | – | – | – | – |
| INOUT | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – |
| SENA | – | – | – | – | – | + | + | – | – | – | – | – | – | – | – | + |
| WILE | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – |
| SENB | – | – | – | – | – | + | + | – | – | – | – | – | – | – | – | – |
| BODY | – | – | – | – | – | + | + | – | – | – | + | – | – | – | – | + |
| IFX | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – |
| IFAX | – | – | – | – | – | + | + | – | – | – | – | – | – | – | – | – |
| FCALL | – | + | + | + | + | + | + | – | – | – | + | + | – | – | – | – |
| FCALLX | – | – | – | – | – | – | – | – | – | – | + | – | – | – | – | – |
| RX | – | + | + | + | + | + | + | – | – | – | + | + | – | – | – | – |
| IDX | – | – | – | – | – | – | – | – | – | – | – | + | – | + | – | – |
| FUN | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – |
| PARM | – | – | – | – | – | – | – | – | – | – | + | – | – | – | – | – |
| PARMX | – | – | – | – | – | – | – | – | – | – | + | – | – | – | – | – |

| | eof | cad | num | true | false | ++ | id | > | && | % | lambda | ( | ) | = | ; | let |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| START | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – |
| CTE | – | – | – | – | – | – | – | + | + | + | + | – | – | – | – | – |
| INC | – | – | – | – | – | – | – | + | + | + | + | – | – | – | + | – |
| EXP | – | – | – | – | – | – | – | – | – | – | + | – | + | – | + | – |
| EXPX | – | – | – | – | – | – | – | – | – | – | + | – | + | – | + | – |
| VALUE | – | – | – | – | – | – | – | + | + | + | + | – | – | – | – | – |
| XPX | – | – | – | – | – | – | – | + | + | + | + | – | – | – | – | – |
| ASIGN | + | – | – | – | – | + | + | – | – | – | + | – | – | – | + | + |
| DECL | + | – | – | – | – | + | + | – | – | – | + | – | – | – | – | + |

| | eof | cad | num | true | false | ++ | id | > | && | % | lambda | ( | ) | = | ; | let |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| DECLX | – | – | – | – | – | – | – | – | – | – | – | – | – | – | + | – |
| TD | – | – | – | – | – | – | + | – | – | – | + | + | – | + | – | – |
| TDX | – | – | – | – | – | – | – | – | – | – | – | + | – | – | – | – |
| INOUT | + | – | – | – | – | + | + | – | – | – | + | – | – | – | + | + |
| SENA | + | – | – | – | – | + | + | – | – | – | + | – | – | – | – | + |
| WILE | + | – | – | – | – | + | + | – | – | – | + | – | – | – | – | + |
| SENB | + | – | – | – | – | + | + | – | – | – | + | – | – | – | + | + |
| BODY | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – |
| IFX | + | – | – | – | – | + | + | – | – | – | + | – | – | – | – | + |
| IFAX | – | – | – | – | – | – | – | – | – | – | – | – | – | – | + | – |
| FCALL | – | – | – | – | – | – | – | – | – | – | – | – | + | – | – | – |
| FCALLX | – | – | – | – | – | – | – | – | – | – | – | – | + | – | – | – |
| RX | – | – | – | – | – | – | – | – | – | – | – | – | – | – | + | – |
| IDX | + | – | – | – | – | + | + | – | – | – | + | – | – | – | + | + |
| FUN | + | – | – | – | – | + | + | – | – | – | – | – | – | – | – | + |
| PARM | – | – | – | – | – | – | – | – | – | – | – | – | + | – | – | – |
| PARMX | – | – | – | – | – | – | – | – | – | – | – | – | + | – | – | – |

Como se puede observar en la tabla no existen conflicots first/first ni first/follow.

2. Finalemnte, entre los No terminales que derivan en No terminales, no existe recursividad por la izquierda en ningún caso.

> Por lo tanto, estamos ante una gramática en forma LL1

# Análisis semántico

Se encarga de controlar los tipados, operaciones de flujo y administra los símbolos introducido por el léxico a la TS

## Gramática-Semántico

Utilizamos reglas estilo DDS para formalizar las reglas que introduciremos en nuestro analizador semántico:

### TRADUCCIÓN DIRIGIDA POR LA SINTAXIS

```
START -> SENA START
START -> {inFunc := true} FUN START
START -> eof

CTE -> cad { ParseLib.insertOperand() }
CTE -> num { ParseLib.insertOperand() }
CTE -> true { ParseLib.insertOperand() }
CTE -> false { ParseLib.insertOperand() }

INC -> ++ id { ParseLib.setID(), ParseLib.CheckExplicitness(),
ParseLib.insertOperand() }

EXP -> VALUE EXPX
EXP -> INC EXPX

EXPX -> {if (!tk.isOperator()) {
            if (e.getFree() != Insertion.LEFT || e.isEmpty())
                then e.clear()
            else {
                cursor.addChild(new ExpNode(e.clear()))
            }
            if (tk.getType().equals("ParClose")) {
                if (cursor.isRoot()) {
                    for (List<ExpNode> nodesList : expresions.findValidPath()){
                        int i := 1,
                        for (ExpNode node : nodesList) {
       i++,
                        }
                        i :=0,
                    }
                    ParseLib.endTree(),
                } else {
                    cursor := cursor.getParent(),
                }
            }
            return,
         }
         e.insert(tk);  }
EXPX -> > {e.setTipo(Token.Tipado.BOOL) } EXP
EXPX -> &&  {e.setTipo(Token.Tipado.BOOL)} EXP
EXPX -> % { e.setTipo(Token.Tipado.INT) } EXP
EXPX -> lambda

VALUE -> id { ParseLib.setID(), ParseLib.insertOperand()} XPX
VALUE -> CTE
VALUE -> ( {cursor := cursor.addChild(new ExpNode(e.clear()))} EXP )
```

```
XPX -> ( { if (id.getType() != "Function")  then ParseLib.ezError(117) else{tmp
:= id; inFCall := true;
            nArgs := 0;  }}

                FCALL { if(nArgs != tmp.getNumParams()) ParseLib.ezError(204,
tmp.getLexema()),
            ,inFCall := false } )
XPX -> lambda   {ParseLib.CheckExplicitness() }

ASIGN ->  = EXP ;

DECL -> let id TD DECLX ;

DECLX -> {inAss := true;} ASIGN {inAss := false; }
DECLX -> lambda

TD -> int
TD -> string
TD -> boolean

TDX -> TD   {   funcID.setOffset(OffsetG)
            ParseLib.IncOffset(tk.getType()),
            funcID.setReturnType(tk.getType()),  }
TDX -> lambda   {  funcID.setReturnType("Void")  }

INOUT -> print EXP ;    {if (!expresions.isEmpty())  then ParseLib.endTree()  }
INOUT -> input id ;     {if (ParseLib.setID())  then
ParseLib.CheckExplicitness()
                if (id.getType() != "TypeInt" && id.getType() != "TypeString")
then ParseLib.ezError(214)}

SENA -> IFX     {}
SENA -> {inVarDec := true  } DECL
SENA -> do { BODY } WILE
WILE -> while ( EXP ) ;
SENA -> SENB

SENB -> id IDX
SENB -> INOUT
SENB -> return RX ;
SENB -> INC ;       {e.clear()}

BODY -> SENA BODY
BODY -> lambda

IFX -> if ( EXP ) IFAX ;

IFAX -> SENB
IFAX -> { SENB }
```

```
FCALL -> EXP {nArgs++} FCALLX
FCALL -> lambda
FCALLX -> , EXP {nArgs++} FCALLX
FCALLX -> lambda

RX -> EXP
RX -> lambda

IDX -> {    ParseLib.setID(),  ParseLib.CheckExplicitness() }
IDX -> ASIGN
IDX -> ( { inFCall := true } FCALL ) ;

FUN -> function {        funcID := Compiler.ts.lookAtIndex((int) tk.getInfo()),
        if (funcID == null)
            then return;
        TabLex := funcID.getLexema(),
        funcID.setType("Function"), funcID.setOffset(OffsetG)  } id TDX
         {inFunc := true} ( {ParseLib.toLowerSc(), inParms := true} PARM
{inParms := false, funcID.setNumParams(nParams) } ) { {
Compiler.ts.changeScope(true)} BODY }  {OffsetG += OffsetL,
        OffsetL = 0,
        Compiler.ts.changeScope(true),
        inFunc := false,
        nParams := 0  }

PARM -> TD { funcID.addTypesParams(tk.getType()),
            LastType := tk.getType(),

                ParseLib.setID(),
                id.setOffset(OffsetL),
                ParseLib.IncOffset(LastType),
                id.setType(LastType),

                nParams++ }
                 }
            id PARMX
PARM -> lambda
PARMX -> , TD {funcID.addTypesParams(tk.getType()),
                LastType := tk.getType(),
                nParams++}
                id {if (ParseLib.setID()) {
                    id.setOffset(OffsetL),
                    ParseLib.IncOffset(LastType),
                    id.setType(LastType),
                     }
                      } PARMX
```

```
PARMX -> lambda
```

## Algoritmo de control de expresiones

Sin duda alguna, uno de los mayores desafíos a los que nos hemos enfrentado para llevar a cabo el analizador semántico ha sido la comprobación del tipado de las expresiones.

Para ello hemos creado un algoritmo de ordenación basado en un árbol de expresiones, permitiendo al parser recrear al final de la expresión el camino a seguir en función del orden de operadores y de la asociatividad de izquierda a derecha.

> Ejemplo: s3.js

```
let buleano boolean;
do{
    ++n1;
}
while  (buleano && (5>(n1%10)) && true);
```

## 1. Creación del árbol

1. Se crea una instancia de la API de `Tree` de java que modela la expresión completa -> `ExptTree`
2. Crea la expresión de izquierda a derecha

- Si encuentra un OpenPar '(' -> Coloca el cursor en el nodo actual y sus hijos pasan a ser los nodos de creación

3. Si encuentra una expresión formada completa (ningún miembro == null):
    - Sube de nivel recursivamente según vaya encontrando ')'
4. Si intenta subir al nivel de root -> termina el árbol

```
ROOT
| └ ID AND
|    └ CteInt GT
|      └ EXP_INT
  └ AND TokT
```

## 2. Comprobación del árbol

Se ha implementado el método getValidPath() en la clase ExpTree para intentar construir un camino válido basado en la precedencia de operadores.

```
1) {left=<ID,1>
, right=<CteInt,10>
, op=<MOD,>
, tipo=INT}

2) {left=<CteInt,5>
, right=, op=<GT,>
, tipo=BOOL}

3) {left=<ID,0>
, right=, op=<AND,>
, tipo=BOOL}

1) {left=, right=<TokT,>
, op=<AND,>
, tipo=BOOL}
```

# Tabla de Simbolos

Hemos utilizado el siguiente diseño para la creación de las tablas de simbolos:

| Lexema | ID | Type | NumParams | TypeParams | ReturnType | Offset |
|---|---|---|---|---|---|---|
| Nombre del Identificador | Posición en la tabla | Tipado asignado | Número de parámetros de la función | List[Tipo del elemento] | Tipo de retorno | Número de bytes desde la creación de la tabla ( Desplazamient |

*El atributo ID es privado, no aparece en el fichero resultante: TS.txt*

Esta tabla se va rellenando siguiendo el formato pedido mediante el uso de un objeto de tipo SymbolAt que va guardando los atributos, las tablas se van modificando a lo largo de la ejecucion de los analizadores, creando, destruyendo o modificando el objeto definido por el lexema.

# Control de errores

## Formateo de errores detectados

Se ha implementado una clase: `ErrorAt` para gestionar los errores encontrados por los distintos analizadores, que genera la siguiente plantilla:

```
ERROR FOUND USING THE {$Analizer} Analyzer
###################
Error #{$ErrorCode} @ line {$LineAt}: {$ExtraInfo}
```

# Anexo

## Pruebas Correctas

### Prueba 1

```
let sexto string;
function alert (string msg_)
{
 print msg_;
}
function pideTexto ()
{
 print "Introduce un texto";
 input texto;
}
pideTexto();
  alert
 (texto);
```

---

- **TOKENS**

```
<ResLet,>
<ID,0>
<TypeString,>
<SemCol,>
<FunID,>
<ID,1>
<ParOpen,>
<TypeString,>
<ID,2>
<ParClose,>
<KeyOpen,>
<ResPrint,>
<ID,2>
<SemCol,>
<KeyClose,>
<FunID,>
<ID,3>
```

```
<ParOpen,>
<ParClose,>
<KeyOpen,>
<ResPrint,>
<Cad,"Introduce un texto">
<SemCol,>
<ResIn,>
<ID,4>
<SemCol,>
<KeyClose,>
<ID,3>
<ParOpen,>
<ParClose,>
<SemCol,>
<ID,1>
<ParOpen,>
<ID,4>
<ParClose,>
<SemCol,>
<Teof,>
```

---

- **TABLA DE SIMBOLOS**

```
TABLA PRINCIPAL #1:
--------- ----------
* Lexema:'sexto'
   Atributos:
   + type:'TypeString'
   + offset:0

--------- ----------
* Lexema:'alert'
   Atributos:
   + type:'Function'
   + NumParams:1
   + TypesParams:'[TypeString]'
   + ReturnType:'Void'
   + offset:64

--------- ----------
* Lexema:'pideTexto'
   Atributos:
```

```
   + type:'Function'
   + NumParams:0
   + TypesParams:'[]'
   + ReturnType:'Void'
   + offset:128

--------- ----------
* Lexema:'texto'
   Atributos:
   + type:'TypeInt'
   + offset:129

--------- ----------

TABLA DE LA FUNCION alert #2:
--------- ----------
* Lexema:'msg_'
   Atributos:
   + type:'TypeString'
   + offset:0

--------- ----------

TABLA DE LA FUNCION pideTexto #3:
--------- ----------
```

---

- **TRAZA PARSER**

```
D    1 32 21 25 23 2 53 28 54 25 57 40 35 37 29 9 15 19 14 41 2 53 28 55 40 35
37 29 9 16 4 14 40 35 37 30 41 1 35 36 52 46 1 35 36 52 45 9 15 19 14 48 3
```

---

- **ARBOL DEL A.S**

- START (1)
  - SENA (32)
    - DECL (21)
      - let
      - id
      - TD (25)
        - string
      - DECLX (23)
        - lambda
      - ;

- START (2)
  - FUN (53)
    - function
    - id
    - TDX (28)
      - lambda
    - (
    - PARM (54)
      - TD (25)
        - string
      - id
      - PARMX (57)
        - lambda
    - )
    - {
    - BODY (40)
      - SENA (35)
        - SENB (37)
          - INOUT (29)
            - print
            - EXP (9)
              - VALUE (15)
                - id
                - XPX (19)
                  - lambda
              - EXPX (14)
                - lambda
            - ;
      - BODY (41)
        - lambda
    - }
  - START (2)
    - FUN (53)
      - function
      - id
      - TDX (28)
        - lambda
      - (
      - PARM (55)
        - lambda
      - )
      - {
      - BODY (40)
        - SENA (35)
          - SENB (37)
            - INOUT (29)
              - print
              - EXP (9)
                - VALUE (16)
                  - CTE (4)
                    - cad
                - EXPX (14)
                  - lambda
              - ;
        - BODY (40)
          - SENA (35)


---

## Prueba 2

```
let a int;
let   b  int  ;
let bbb boolean ;
a = 3;
b=a
;
let c boolean;

c = a > b;
```

```
if (c) b = 3333;
a = a % b;
print a ;
print b;
```

---

- **TOKENS**

```
<ResLet,>
<ID,0>
<TypeInt,>
<SemCol,>
<ResLet,>
<ID,1>
<TypeInt,>
<SemCol,>
<ResLet,>
<ID,2>
<TypeBool,>
<SemCol,>
<ID,0>
<AsValue,>
<CteInt,3>
<SemCol,>
<ID,1>
<AsValue,>
<ID,0>
<SemCol,>
<ResLet,>
<ID,3>
<TypeBool,>
<SemCol,>
<ID,3>
<AsValue,>
<ID,0>
<GT,>
<ID,1>
<SemCol,>
<CondIf,>
<ParOpen,>
<ID,3>
<ParClose,>
```

```
<ID,1>
<AsValue,>
<CteInt,3333>
<SemCol,>
<ID,0>
<AsValue,>
<ID,0>
<MOD,>
<ID,1>
<SemCol,>
<ResPrint,>
<ID,0>
<SemCol,>
<ResPrint,>
<ID,1>
<SemCol,>
<Teof,>
```

---

- **TABLA DE SIMBOLOS**

```
TABLA PRINCIPAL #1:
--------- ----------
* Lexema:'a'
   Atributos:
   + type:'TypeInt'
   + offset:0

--------- ----------
* Lexema:'b'
   Atributos:
   + type:'TypeInt'
   + offset:1

--------- ----------
* Lexema:'bbb'
   Atributos:
   + type:'TypeBool'
   + offset:2

--------- ----------
* Lexema:'c'
   Atributos:
```

```
    + type:'TypeBool'
    + offset:3
```

---

- **TRAZA PARSER**

```
D    1 32 21 24 23 1 32 21 24 23 1 32 21 26 23 1 35 36 51 20 9 16 5 14 1 35 36
51 20 9 15 19 14 1 32 21 26 23 1 35 36 51 20 9 15 19 11 9 15 19 14 1 31 42 9 15
19 14 44 36 51 20 9 16 5 14 1 35 36 51 20 9 15 19 13 9 15 19 14 1 35 37 29 9 15
19 14 1 35 37 29 9 15 19 14 3
```

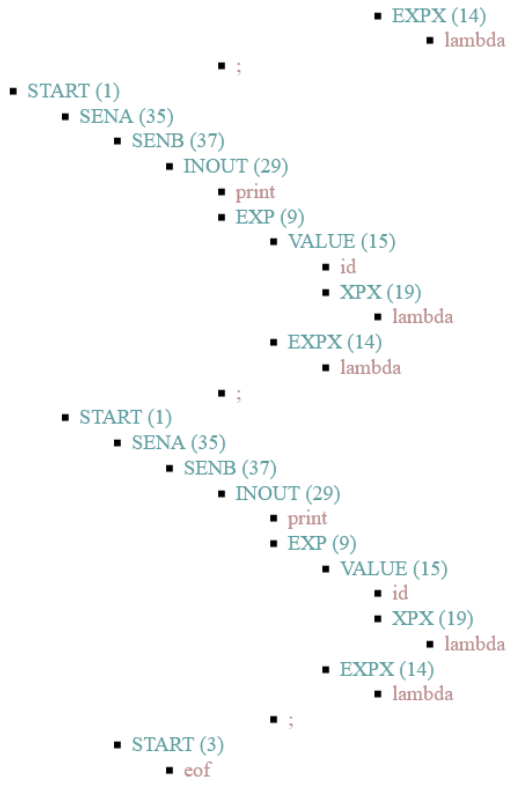
---

- **ARBOL DEL A.S**

- START (1)
  - SENA (32)
    - DECL (21)
      - let
      - id
      - TD (24)
        - int
      - DECLX (23)
        - lambda
      - ;
  - START (1)
    - SENA (32)
      - DECL (21)
        - let
        - id
        - TD (24)
          - int
        - DECLX (23)
          - lambda
        - ;
    - START (1)
      - SENA (32)
        - DECL (21)
          - let
          - id
          - TD (26)
            - boolean
          - DECLX (23)
            - lambda
          - ;
        - START (1)
          - SENA (35)
            - SENB (36)
              - id
              - IDX (51)
                - ASIGN (20)
                  - =
                  - EXP (9)
                    - VALUE (16)
                      - CTE (5)
                        - num
                    - EXPX (14)
                      - lambda
                  - ;
          - START (1)
            - SENA (35)
              - SENB (36)
                - id
                - IDX (51)
                  - ASIGN (20)
                    - =
                    - EXP (9)
                      - VALUE (15)
                        - id
                        - XPX (19)

- lambda
- EXPX (14)
  - lambda
- ;

- START (1)
  - SENA (32)
    - DECL (21)
      - let
      - id
      - TD (26)
        - boolean
      - DECLX (23)
        - lambda
      - ;
    - START (1)
      - SENA (35)
        - SENB (36)
          - id
          - IDX (51)
            - ASIGN (20)
              - =
              - EXP (9)
                - VALUE (15)
                  - id
                  - XPX (19)
                    - lambda
                - EXPX (11)
                  - >
                  - EXP (9)
                    - VALUE (15)
                      - id
                      - XPX (19)
                        - lambda
                    - EXPX (14)
                      - lambda
              - ;
      - START (1)
        - SENA (31)
          - IFX (42)
            - if
            - (
            - EXP (9)
              - VALUE (15)
                - id
                - XPX (19)
                  - lambda
              - EXPX (14)
                - lambda
            - )
            - IFAX (44)
              - {
              - SENB (36)
                - id
                - IDX (51)
                  - ASIGN (20)
                    - =
                    - EXP (9)
                      - VALUE (16)
                        - CTE (5)
                          - num
                      - EXPX (14)
                        - lambda
                    - ;
              - }
        - START (1)
          - SENA (35)
            - SENB (36)
              - id
              - IDX (51)
                - ASIGN (20)
                  - =
                  - EXP (9)
                    - VALUE (15)
                      - id
                      - XPX (19)
                        - lambda
                    - EXPX (13)
                      - %
                      - EXP (9)
                        - VALUE (15)
                          - id
                          - XPX (19)
                            - lambda




## Prueba 3

```
let n1  int     ;
let l1 boolean ;
let cad    string  ;
let n2 int     ;
let l2 boolean ;
input n1;
l1 = l2;
if (l1&& l2) cad = "hello";
n2 = n1 % 378;

print  33
%
      n1
%
      n2;

function ff boolean(boolean ss)
{
   varglobal = 8;
   if (l1) l2 = ff (ss);
   return ss;
}
```

```
if (ff(l2))
    print varglobal;
```

---

- **TOKENS**

```
<ResLet,>
<ID,0>
<TypeInt,>
<SemCol,>
<ResLet,>
<ID,1>
<TypeBool,>
<SemCol,>
<ResLet,>
<ID,2>
<TypeString,>
<SemCol,>
<ResLet,>
<ID,3>
<TypeInt,>
<SemCol,>
<ResLet,>
<ID,4>
<TypeBool,>
<SemCol,>
<ResIn,>
<ID,0>
<SemCol,>
<ID,1>
<AsValue,>
<ID,4>
<SemCol,>
<CondIf,>
<ParOpen,>
<ID,1>
<AND,>
<ID,4>
<ParClose,>
<ID,2>
<AsValue,>
```

```
<Cad,"hello">
<SemCol,>
<ID,3>
<AsValue,>
<ID,0>
<MOD,>
<CteInt,378>
<SemCol,>
<ResPrint,>
<CteInt,33>
<MOD,>
<ID,0>
<MOD,>
<ID,3>
<SemCol,>
<FunID,>
<ID,5>
<TypeBool,>
<ParOpen,>
<TypeBool,>
<ID,6>
<ParClose,>
<KeyOpen,>
<ID,7>
<AsValue,>
<CteInt,8>
<SemCol,>
<CondIf,>
<ParOpen,>
<ID,1>
<ParClose,>
<ID,4>
<AsValue,>
<ID,5>
<ParOpen,>
<ID,6>
<ParClose,>
<SemCol,>
<Return,>
<ID,6>
<SemCol,>
<KeyClose,>
<CondIf,>
```

```
<ParOpen,>
<ID,5>
<ParOpen,>
<ID,4>
<ParClose,>
<ParClose,>
<ResPrint,>
<ID,7>
<SemCol,>
<Teof,>
```

---

- **TABLA DE SIMBOLOS**

```
TABLA PRINCIPAL #1:
--------- ----------
* Lexema:'n1'
   Atributos:
   + type:'TypeInt'
   + offset:0

--------- ----------
* Lexema:'l1'
   Atributos:
   + type:'TypeBool'
   + offset:1

--------- ----------
* Lexema:'cad'
   Atributos:
   + type:'TypeString'
   + offset:2

--------- ----------
* Lexema:'n2'
   Atributos:
   + type:'TypeInt'
   + offset:66

--------- ----------
* Lexema:'l2'
   Atributos:
   + type:'TypeBool'
```

```
   + offset:67

--------- ----------
* Lexema:'ff'
   Atributos:
   + type:'Function'
   + NumParams:1
   + TypesParams:'[TypeBool]'
   + ReturnType:'TypeBool'
   + offset:68

--------- ----------
* Lexema:'varglobal'
   Atributos:
   + type:'TypeInt'
   + offset:69

--------- ----------

TABLA DE LA FUNCION ff #2:
--------- ----------
* Lexema:'ss'
   Atributos:
   + type:'TypeBool'
   + offset:1

--------- ----------
```

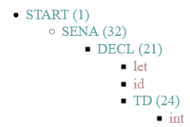
---

- **TRAZA PARSER**

```
D    1 32 21 24 23 1 32 21 26 23 1 32 21 25 23 1 32 21 24 23 1 32 21 26 23 1 35
37 30 1 35 36 51 20 9 15 19 14 1 31 42 9 15 19 12 9 15 19 14 44 36 51 20 9 16 4
14 1 35 36 51 20 9 15 19 13 9 16 5 14 1 35 37 29 9 16 5 13 9 15 19 13 9 15 19 14
2 53 27 26 54 26 57 40 35 36 51 20 9 16 5 14 40 31 42 9 15 19 14 44 36 51 20 9
15 18 45 9 15 19 14 48 14 40 35 38 49 9 15 19 14 41 1 31 42 9 15 18 45 9 15 19
14 48 14 44 37 29 9 15 19 14 3
```

---

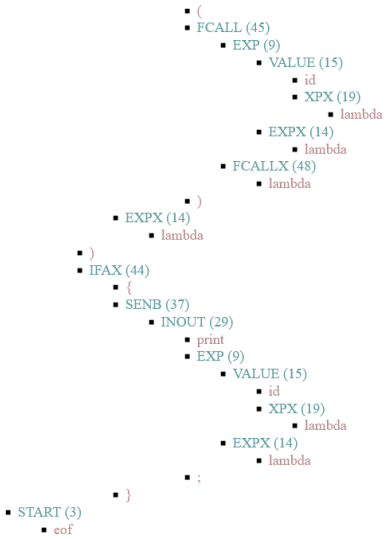
- **ARBOL DEL A.S**



- DECLX (23)
  - lambda
- ;

- START (1)
  - SENA (32)
    - DECL (21)
      - let
      - id
      - TD (26)
        - boolean
      - DECLX (23)
        - lambda
      - ;
    - START (1)
      - SENA (32)
        - DECL (21)
          - let
          - id
          - TD (25)
            - string
          - DECLX (23)
            - lambda
          - ;
        - START (1)
          - SENA (32)
            - DECL (21)
              - let
              - id
              - TD (24)
                - int
              - DECLX (23)
                - lambda
              - ;
            - START (1)
              - SENA (32)
                - DECL (21)
                  - let
                  - id
                  - TD (26)
                    - boolean
                  - DECLX (23)
                    - lambda
                  - ;
                - START (1)
                  - SENA (35)
                    - SENB (37)
                      - INOUT (30)
                        - input
                        - id
                        - ;
                  - START (1)
                    - SENA (35)
                      - SENB (36)
                        - id
                        - IDX (51)
                          - ASIGN (20)
                            - =
                            - EXP (9)
                              - VALUE (15)
                                - id
                                - XPX (19)
                                  - lambda
                              - EXPX (14)
                                - lambda
                            - ;
                    - START (1)
                      - SENA (31)
                        - IFX (42)
                          - if
                          - (
                          - EXP (9)
                            - VALUE (15)
                              - id
                              - XPX (19)
                                - lambda
                            - EXPX (12)
                              - &&
                              - EXP (9)
                                - VALUE (15)
                                  - id
                                - XPX (19)
                                  - lambda
                                - EXPX (14)
                                  - lambda
                          - )
                          - IFAX (44)
                            - {
                            - SENB (36)
                              - id
                              - IDX (51)
                                - ASIGN (20)
                                  - =
                                  - EXP (9)
                                    - VALUE (16)
                                      - CTE (4)
                                        - cad
                                    - EXPX (14)
                                      - lambda
                                  - ;
                            - }
                        - START (1)
                          - SENA (35)
                            - SENB (36)
                              - id
                              - IDX (51)
                                - ASIGN (20)
                                  - =
                                  - EXP (9)
                                    - VALUE (15)
                                      - id
                                      - XPX (19)
                                        - lambda
                                    - EXPX (13)
                                      - %
                                      - EXP (9)
                                        - VALUE (16)
                                          - CTE (5)
                                            - num
                                        - EXPX (14)
                                          - lambda
                                  - ;


START (1)
SENA (35)
SENB (37)
INOUT (29)
print
EXP (9)
VALUE (16)
CTE (5)
num
EXPX (13)
%
EXP (9)
VALUE (15)
id
XPX (19)
lambda
EXPX (13)
%
EXP (9)
VALUE (15)
id
XPX (19)
lambda
EXPX (14)
lambda
;
START (2)
FUN (53)
function
id
TDX (27)
TD (26)
boolean
(
PARM (54)
TD (26)
boolean
id
PARMX (57)
lambda
)
{
BODY (40)
SENA (35)
SENB (36)
id
IDX (51)
ASIGN (20)
=
EXP (9)
VALUE (16)
CTE (5)
num
EXPX (14)
lambda
;
BODY (40)
SENA (31)
IFX (42)
if
(
EXP (9)
VALUE (15)
id
XPX (19)
lambda
EXPX (14)
lambda
)
IFAX (44)
{
SENB (36)
id
IDX (51)
ASIGN (20)
=
EXP (9)
VALUE (15)
id
XPX (18)
(
FCALL (45)
EXP (9)
VALUE (15)
id
XPX (19)
lambda
EXPX (14)
lambda
FCALLX (48)
lambda
)
EXPX (14)
lambda
;
}
BODY (40)
SENA (35)
SENB (38)
return
RX (49)
EXP (9)
VALUE (15)
id
XPX (19)
lambda
EXPX (14)
lambda
;
BODY (41)
lambda
}
START (1)
SENA (31)
IFX (42)
if
(
EXP (9)
VALUE (15)
id
XPX (18)




---

## Prueba 4

```
let ass int = 0;
let pss int = 0;
function a int(int p,string q,boolean r,string s)
{
   do{
      ++p;
      print p;
   }while(100 > p && r && p%2 > 20);

}
function b int(int x,int y,int z)
{
   return x % y % z;
}
function c boolean(boolean x,boolean y)
{
   let tmp int = 8;
   ++tmp;
   let entry string;
   input entry;
   if (x && y) ++tmp;
   return tmp > ass;
}
function d (boolean bllsit)
{
   if (bllsit) bllsit = false;
   return;
```

```
}
 if(ass > pss) d(true);
```

---

- **TOKENS**

```
<ResLet,>
<ID,0>
<TypeInt,>
<AsValue,>
<CteInt,0>
<SemCol,>
<ResLet,>
<ID,1>
<TypeInt,>
<AsValue,>
<CteInt,0>
<SemCol,>
<FunID,>
<ID,2>
<TypeInt,>
<ParOpen,>
<TypeInt,>
<ID,3>
<Com,>
<TypeString,>
<ID,4>
<Com,>
<TypeBool,>
<ID,5>
<Com,>
<TypeString,>
<ID,6>
<ParClose,>
<KeyOpen,>
<LoopDo,>
<KeyOpen,>
<ResAutoSum,>
<ID,3>
<SemCol,>
<ResPrint,>
<ID,3>
<SemCol,>
```

```
<KeyClose,>
<LoopWhile,>
<ParOpen,>
<CteInt,100>
<GT,>
<ID,3>
<AND,>
<ID,5>
<AND,>
<ID,3>
<MOD,>
<CteInt,2>
<GT,>
<CteInt,20>
<ParClose,>
<SemCol,>
<KeyClose,>
<FunID,>
<ID,7>
<TypeInt,>
<ParOpen,>
<TypeInt,>
<ID,8>
<Com,>
<TypeInt,>
<ID,9>
<Com,>
<TypeInt,>
<ID,10>
<ParClose,>
<KeyOpen,>
<Return,>
<ID,8>
<MOD,>
<ID,9>
<MOD,>
<ID,10>
<SemCol,>
<KeyClose,>
<FunID,>
<ID,11>
<TypeBool,>
<ParOpen,>
```

<TypeBool,>
<ID,12>
<Com,>
<TypeBool,>
<ID,13>
<ParClose,>
<KeyOpen,>
<ResLet,>
<ID,14>
<TypeInt,>
<AsValue,>
<CteInt,8>
<SemCol,>
<ResAutoSum,>
<ID,14>
<SemCol,>
<ResLet,>
<ID,15>
<TypeString,>
<SemCol,>
<ResIn,>
<ID,15>
<SemCol,>
<CondIf,>
<ParOpen,>
<ID,12>
<AND,>
<ID,13>
<ParClose,>
<ResAutoSum,>
<ID,14>
<SemCol,>
<Return,>
<ID,14>
<GT,>
<ID,0>
<SemCol,>
<KeyClose,>
<FunID,>
<ID,16>
<ParOpen,>
<TypeBool,>
<ID,17>

```
<ParClose,>
<KeyOpen,>
<CondIf,>
<ParOpen,>
<ID,17>
<ParClose,>
<ID,17>
<AsValue,>
<TokF,>
<SemCol,>
<Return,>
<SemCol,>
<KeyClose,>
<CondIf,>
<ParOpen,>
<ID,0>
<GT,>
<ID,1>
<ParClose,>
<ID,16>
<ParOpen,>
<TokT,>
<ParClose,>
<SemCol,>
<Teof,>
```

---

- **TABLA DE SIMBOLOS**

```
TABLA PRINCIPAL #1:
--------- ----------
* Lexema:'ass'
   Atributos:
   + type:'TypeInt'
   + offset:0

--------- ----------
* Lexema:'d'
   Atributos:
   + type:'Function'
   + NumParams:1
   + TypesParams:'[TypeBool]'
   + ReturnType:'Void'
```

```
   + offset:205

--------- ----------
* Lexema:'pss'
   Atributos:
   + type:'TypeInt'
   + offset:1

--------- ----------
* Lexema:'a'
   Atributos:
   + type:'Function'
   + NumParams:4
   + TypesParams:'[TypeInt, TypeString, TypeBool, TypeString]'
   + ReturnType:'TypeInt'
   + offset:2

--------- ----------
* Lexema:'b'
   Atributos:
   + type:'Function'
   + NumParams:3
   + TypesParams:'[TypeInt, TypeInt, TypeInt]'
   + ReturnType:'TypeInt'
   + offset:133

--------- ----------
* Lexema:'c'
   Atributos:
   + type:'Function'
   + NumParams:2
   + TypesParams:'[TypeBool, TypeBool]'
   + ReturnType:'TypeBool'
   + offset:137

--------- ----------

TABLA DE LA FUNCION a #2:
--------- ----------
* Lexema:'p'
   Atributos:
   + type:'TypeInt'
   + offset:0
```

```
--------- ----------
* Lexema:'q'
   Atributos:
   + type:'TypeString'
   + offset:1

--------- ----------
* Lexema:'r'
   Atributos:
   + type:'TypeBool'
   + offset:65

--------- ----------
* Lexema:'s'
   Atributos:
   + type:'TypeString'
   + offset:66

--------- ----------
--------- ----------

TABLA DE LA FUNCION b #3:
--------- ----------
* Lexema:'x'
   Atributos:
   + type:'TypeInt'
   + offset:0

--------- ----------
* Lexema:'y'
   Atributos:
   + type:'TypeInt'
   + offset:1

--------- ----------
* Lexema:'z'
   Atributos:
   + type:'TypeInt'
   + offset:2

--------- ----------
--------- ----------
```

```
TABLA DE LA FUNCION c #4:
--------- ----------
* Lexema:'x'
   Atributos:
   + type:'TypeBool'
   + offset:0

--------- ----------
* Lexema:'y'
   Atributos:
   + type:'TypeBool'
   + offset:1

--------- ----------
* Lexema:'tmp'
   Atributos:
   + type:'TypeInt'
   + offset:2

--------- ----------
* Lexema:'entry'
   Atributos:
   + type:'TypeString'
   + offset:3

--------- ----------
--------- ----------

TABLA DE LA FUNCION d #5:
--------- ----------
* Lexema:'bllsit'
   Atributos:
   + type:'TypeBool'
   + offset:0

--------- ----------
```

---

- **TRAZA PARSER**

```
D    1 32 21 24 22 20 9 16 5 14 1 32 21 24 22 20 9 16 5 14 2 53 27 24 54 24 56
25 56 26 56 25 57 40 33 40 35 39 8 40 35 37 29 9 15 19 14 41 34 9 16 5 11 9 15
```

```
19 12 9 15 19 12 9 15 19 13 9 16 5 11 9 16 5 14 41 2 53 27 24 54 24 56 24 56 24
57 40 35 38 49 9 15 19 13 9 15 19 13 9 15 19 14 41 2 53 27 26 54 26 56 26 57 40
32 21 24 22 20 9 16 5 14 40 35 39 8 40 32 21 25 23 40 35 37 30 40 31 42 9 15 19
12 9 15 19 14 44 39 8 40 35 38 49 9 15 19 11 9 15 19 14 41 2 53 28 54 26 57 40
31 42 9 15 19 14 44 36 51 20 9 16 7 14 40 35 38 50 41 1 31 42 9 15 19 11 9 15 19
14 44 36 52 45 9 16 6 14 48 3
```

---

- ## ARBOL DEL A.S

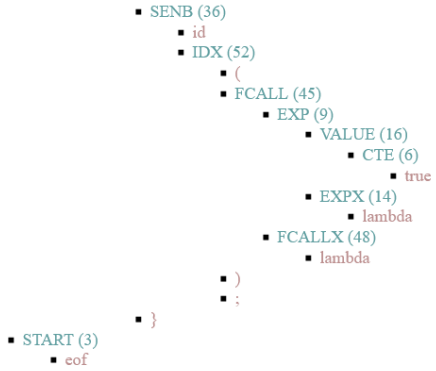
- SENB (37)
  - INOUT (29)
    - print
    - EXP (9)
      - VALUE (15)
        - id
        - XPX (19)
          - lambda
      - EXPX (14)
        - lambda
    - ;
- BODY (41)
  - lambda
- }
- WILE (34)
  - while
  - (
  - EXP (9)
    - VALUE (16)
      - CTE (5)
        - num
    - EXPX (11)
      - >
      - EXP (9)
        - VALUE (15)
          - id
          - XPX (19)
            - lambda
        - EXPX (12)
          - &&
          - EXP (9)
            - VALUE (15)
              - id
              - XPX (19)
                - lambda
            - EXPX (12)
              - &&
              - EXP (9)
                - VALUE (15)
                  - id
                  - XPX (19)
                    - lambda
                - EXPX (13)
                  - %
                  - EXP (9)
                    - VALUE (16)
                      - CTE (5)
                        - num
                    - EXPX (11)
                      - >
                      - EXP (9)
                        - VALUE (16)
                          - CTE (5)
                            - num
                        - EXPX (14)
                          - lambda
  - )
  - ;
- BODY (41)
  - lambda
- }

- START (2)
  - FUN (53)
    - function
    - id
    - TDX (27)
      - TD (24)
        - int
    - (
    - PARM (54)
      - TD (24)
        - int
      - id
      - PARMX (56)
        - ,
        - TD (24)
          - int
        - id
        - PARMX (56)
          - ,
          - TD (24)
            - int
          - id
          - PARMX (57)
            - lambda
    - )
    - {
    - BODY (40)
      - SENA (35)
        - SENB (38)
          - return
          - RX (49)
            - EXP (9)
              - VALUE (15)
                - id
                - XPX (19)
                  - lambda
              - EXPX (13)
                - %
                - EXP (9)
                  - VALUE (15)
                    - id

- XPX (19)
  - lambda
- EXPX (13)
  - %
  - EXP (9)
    - VALUE (15)
      - id
      - XPX (19)
        - lambda
    - EXPX (14)
      - lambda
- ;
- BODY (41)
  - lambda
- }

- START (2)
  - FUN (53)
    - function
    - id
    - TDX (27)
      - TD (26)
        - boolean
    - (
    - PARM (54)
      - TD (26)
        - boolean
      - id
      - PARMX (56)
        - ,
        - TD (26)
          - boolean
        - id
        - PARMX (57)
          - lambda
    - )
    - {
    - BODY (40)
      - SENA (32)
        - DECL (21)
          - let
          - id
          - TD (24)
            - int
          - DECLX (22)
            - ASIGN (20)
              - =
              - EXP (9)
                - VALUE (16)
                  - CTE (5)
                    - num
                - EXPX (14)
                  - lambda
              - ;
          - ;
      - BODY (40)
        - SENA (35)
          - SENB (39)
            - INC (8)
              - ++
              - id
            - ;
        - BODY (40)
          - SENA (32)
            - DECL (21)
              - let
              - id
              - TD (25)
                - string
              - DECLX (23)
                - lambda
              - ;
          - BODY (40)
            - SENA (35)
              - SENB (37)
                - INOUT (30)
                  - input
                  - id
                  - ;
              - BODY (40)
                - SENA (31)
                  - IFX (42)
                    - if
                    - (
                    - EXP (9)
                      - VALUE (15)
                        - id
                        - XPX (19)
                          - lambda
                      - EXPX (12)
                        - &&
                        - EXP (9)
                          - VALUE (15)
                            - id
                            - XPX (19)
                              - lambda
                          - EXPX (14)
                            - lambda
                    - )
                    - IFAX (44)
                      - {
                      - SENB (39)
                        - INC (8)

- ++
- id
- ;
- }
- BODY (40)
- SENA (35)
- SENB (38)
- return
- RX (49)
- EXP (9)
- VALUE (15)
- id
- XPX (19)
- lambda
- EXPX (11)
- >
- EXP (9)
- VALUE (15)
- id
- XPX (19)
- lambda
- EXPX (14)
- lambda
- ;
- BODY (41)
- lambda
- }
- START (2)
- FUN (53)
- function
- id
- TDX (28)
- lambda
- (
- PARM (54)
- TD (26)
- boolean
- id
- PARMX (57)
- lambda
- )
- {
- BODY (40)
- SENA (31)
- IFX (42)
- if
- (
- EXP (9)
- VALUE (15)
- id
- XPX (19)
- lambda
- EXPX (14)
- lambda
- )
- IFAX (44)
- {
- SENB (36)
- id
- IDX (51)
- ASIGN (20)
- =
- EXP (9)
- VALUE (16)
- CTE (7)
- false
- EXPX (14)
- lambda
- ;
- }
- BODY (40)
- SENA (35)
- SENB (38)
- return
- RX (50)
- lambda
- ;
- BODY (41)
- lambda
- }
- START (1)
- SENA (31)
- IFX (42)
- if
- (
- EXP (9)
- VALUE (15)
- id
- XPX (19)
- lambda
- EXPX (11)
- >
- EXP (9)
- VALUE (15)
- id
- XPX (19)
- lambda
- EXPX (14)
- lambda
- )
- IFAX (44)
- {




---

## Prueba 5

```
let a int = 0;
let b int = 50;
function sout (string cad) {

   print cad;

}
function out(){
   let entry string;
   input entry;
   do{
      sout(entry);
      ++a;
   }while(100 > a && a % b > 0);
   if( a > 300) print "a mayor que 300";
   return;}
```

---

- **TOKENS**

```
<ResLet,>
<ID,0>
<TypeInt,>
<AsValue,>
<CteInt,0>
<SemCol,>
<ResLet,>
<ID,1>
<TypeInt,>
```

```
<AsValue,>
<CteInt,50>
<SemCol,>
<FunID,>
<ID,2>
<ParOpen,>
<TypeString,>
<ID,3>
<ParClose,>
<KeyOpen,>
<ResPrint,>
<ID,3>
<SemCol,>
<KeyClose,>
<FunID,>
<ID,4>
<ParOpen,>
<ParClose,>
<KeyOpen,>
<ResLet,>
<ID,5>
<TypeString,>
<SemCol,>
<ResIn,>
<ID,5>
<SemCol,>
<LoopDo,>
<KeyOpen,>
<ID,2>
<ParOpen,>
<ID,5>
<ParClose,>
<SemCol,>
<ResAutoSum,>
<ID,0>
<SemCol,>
<KeyClose,>
<LoopWhile,>
<ParOpen,>
<CteInt,100>
<GT,>
<ID,0>
<AND,>
```

```
<ID,0>
<MOD,>
<ID,1>
<GT,>
<CteInt,0>
<ParClose,>
<SemCol,>
<CondIf,>
<ParOpen,>
<ID,0>
<GT,>
<CteInt,300>
<ParClose,>
<ResPrint,>
<Cad,"a mayor que 300">
<SemCol,>
<Return,>
<SemCol,>
<KeyClose,>
<Teof,>
```

---

- **TABLA DE SIMBOLOS**

```
TABLA PRINCIPAL #1:
--------- ----------
* Lexema:'a'
   Atributos:
   + type:'TypeInt'
   + offset:0

--------- ----------
* Lexema:'b'
   Atributos:
   + type:'TypeInt'
   + offset:1

--------- ----------
* Lexema:'sout'
   Atributos:
   + type:'Function'
   + NumParams:1
   + TypesParams:'[TypeString]'
```

```
   + ReturnType:'Void'
   + offset:2

--------- ----------
* Lexema:'out'
   Atributos:
   + type:'Function'
   + NumParams:0
   + TypesParams:'[]'
   + ReturnType:'Void'
   + offset:66

--------- ----------

TABLA DE LA FUNCION sout #2:
--------- ----------
* Lexema:'cad'
   Atributos:
   + type:'TypeString'
   + offset:0

--------- ----------
--------- ----------

TABLA DE LA FUNCION out #3:
--------- ----------
* Lexema:'entry'
   Atributos:
   + type:'TypeString'
   + offset:0

--------- ----------
```

---

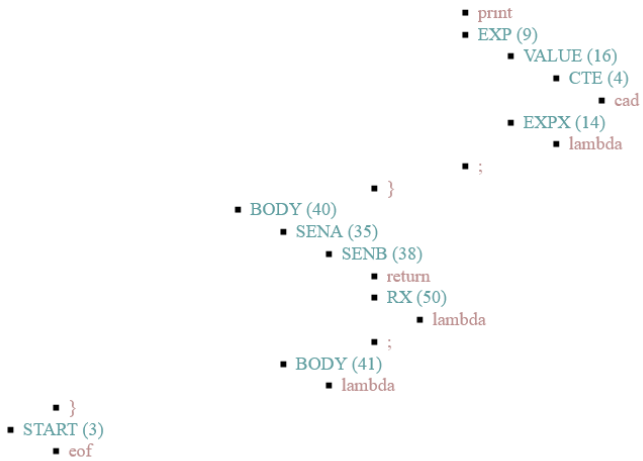
- **TRAZA PARSER**

```
D    1 32 21 24 22 20 9 16 5 14 1 32 21 24 22 20 9 16 5 14 2 53 28 54 25 57 40
35 37 29 9 15 19 14 41 2 53 28 55 40 32 21 25 23 40 35 37 30 40 33 40 35 36 52
45 9 15 19 14 48 40 35 39 8 41 34 9 16 5 11 9 15 19 12 9 15 19 13 9 15 19 11 9
16 5 14 40 31 42 9 15 19 11 9 16 5 14 44 37 29 9 16 4 14 40 35 38 50 41 3
```

---

- ARBOL DEL A.S
  - START (1)
    - SENA (32)
      - DECL (21)
        - let
        - id
        - TD (24)
          - int
        - DECLX (22)
          - ASIGN (20)
            - =
            - EXP (9)
              - VALUE (16)
                - CTE (5)
                  - num
              - EXPX (14)
                - lambda
            - ;
        - ;
    - START (1)
      - SENA (32)
        - DECL (21)
          - let
          - id
          - TD (24)
            - int
          - DECLX (22)
            - ASIGN (20)
              - =
              - EXP (9)
                - VALUE (16)
                  - CTE (5)
                    - num
                - EXPX (14)
                  - lambda
              - ;
          - ;
        - START (2)
          - FUN (53)
            - function
            - id
            - TDX (28)
              - lambda
            - (
            - PARM (54)
              - TD (25)
                - string
              - id
              - PARMX (57)
                - lambda
            - )
            - {
            - BODY (40)
              - SENA (35)
                - SENB (37)
                  - INOUT (29)
                    - print
                    - EXP (9)
                      - VALUE (15)
                        - id
                        - XPX (19)
                          - lambda
                      - EXPX (14)
                        - lambda
                    - ;
              - BODY (41)
                - lambda
            - }
          - START (2)
            - FUN (53)
              - function
              - id
              - TDX (28)
                - lambda
              - (
              - PARM (55)
                - lambda
              - )
              - {
              - BODY (40)
                - SENA (32)
                  - DECL (21)
                    - let
                    - id
                    - TD (25)
                      - string
                    - DECLX (23)
                      - lambda
                    - ;
                - BODY (40)
                  - SENA (35)
                    - SENB (37)
                      - INOUT (30)
                        - input


id
;
BODY (40)
SENA (33)
do
{
BODY (40)
SENA (35)
SENB (36)
id
IDX (52)
(
FCALL (45)
EXP (9)
VALUE (15)
id
XPX (19)
lambda
EXPX (14)
lambda
FCALLX (48)
lambda
)
;
BODY (40)
SENA (35)
SENB (39)
INC (8)
++
id
;
BODY (41)
lambda
}
WILE (34)
while
(
EXP (9)
VALUE (16)
CTE (5)
num
EXPX (11)
>
EXP (9)
VALUE (15)
id
XPX (19)
lambda
EXPX (12)
&&
EXP (9)
VALUE (15)
id
XPX (19)
lambda
EXPX (13)
%
EXP (9)
VALUE (15)
id
XPX (19)
lambda
EXPX (11)
>
EXP (9)
VALUE (16)
CTE (5)
num
EXPX (14)
lambda
)
;
BODY (40)
SENA (31)
IFX (42)
if
(
EXP (9)
VALUE (15)
id
XPX (19)
lambda
EXPX (11)
>
EXP (9)
VALUE (16)
CTE (5)
num
EXPX (14)
lambda
)
IFAX (44)
{
SENB (37)
INOUT (29)




---

# Pruebas con errores

## Prueba 6

```
/* String e Int muy largos, faltan algunos puntos y comas */
let sexto string;
let a int;
function alert (string msg_)
{
   print "mnzimswrj0szl1i46uoxz1geg87h5xp4bnga70pktl9tgodjakcur4lpawhoghkfb"
   print msg_;

}
function pideTexto ()
{
   print "Introduce un texto";
   a = 99999;
   input texto;
}
pideTexto();
  alert
```

---

- **Errores**

```
ERROR FOUND USING THE LEXICAL ANALYZER
####################
Error #12 @ line 7: String demasiado largo

ERROR FOUND USING THE LEXICAL ANALYZER
```

```
###################
Error #12 @ line 7: String demasiado largo

ERROR FOUND USING THE SINTACTIC ANALYZER
###################
Error #107 @ line 8: Se esperaba ';'
- TRAZA -> D    1 32 21 25 23 1 32 21 24 23 2 53 28 54 25 57 40 35 37 29 9 16 4
14
- ÚLTIMO TK LEIDO -> <ResPrint,>


ERROR FOUND USING THE SINTACTIC ANALYZER
###################
Error #103 @ line 8: Se esperaba el inicio de la expresión: '(
- TRAZA -> D    1 32 21 25 23 1 32 21 24 23 2 53 28 54 25 57 40 35 37 29 9 16 4
14 40 35 36
- ÚLTIMO TK LEIDO -> <SemCol,>


ERROR FOUND USING THE LEXICAL ANALYZER
###################
Error #11 @ line 14: SE HA SUPERADO EL MÁXIMO INT

ERROR FOUND USING THE SINTACTIC ANALYZER
###################
Error #103 @ line 18: Se esperaba el inicio de la expresión: '(
- TRAZA -> D    1 32 21 25 23 1 32 21 24 23 2 53 28 54 25 57 40 35 37 29 9 16 4
14 40 35 36 41 2 53 28 55 40 35 37 29 9 16 4 14 40 35 36 51 20 9 16 5 14 40 35
37 30 41 1 35 36 52 46 1 35 36
- ÚLTIMO TK LEIDO -> <Teof,>
```

---

---

---

---

## Prueba 7

```
/* Tests INOUT mal */
function badPrint(){
    print true
    print ;
}
```

```
function badInput(){
    input (something);
}

badPrint();
badInput();
```

---

- **ERRORES**

```
ERROR FOUND USING THE SINTACTIC ANALYZER
###################
Error #107 @ line 5: Se esperaba ';'
- TRAZA -> D    2 53 28 55 40 35 37 29 9 16 6 14
- ÚLTIMO TK LEIDO -> <ResPrint,>


ERROR FOUND USING THE SINTACTIC ANALYZER
###################
Error #102 @ line 5: Se esperaba una declaración o un condicional
- TRAZA -> D    2 53 28 55 40 35 37 29 9 16 6 14 40
- ÚLTIMO TK LEIDO -> <SemCol,>


ERROR FOUND USING THE SINTACTIC ANALYZER
###################
Error #105 @ line 9: Se esperaba un identificador
- TRAZA -> D    2 53 28 55 40 35 37 29 9 16 6 14 40 41 2 53 28 55 40 35 37 30
- ÚLTIMO TK LEIDO -> <ParOpen,>


ERROR FOUND USING THE SINTACTIC ANALYZER
###################
Error #103 @ line 9: Se esperaba el inicio de la expresión: '(
- TRAZA -> D    2 53 28 55 40 35 37 29 9 16 6 14 40 41 2 53 28 55 40 35 37 30 40
35 36
- ÚLTIMO TK LEIDO -> <ParClose,>


ERROR FOUND USING THE SINTACTIC ANALYZER
###################
Error #102 @ line 9: Se esperaba una declaración o un condicional
```

```
- TRAZA -> D    2 53 28 55 40 35 37 29 9 16 6 14 40 41 2 53 28 55 40 35 37 30 40
35 36 40
- ÚLTIMO TK LEIDO -> <SemCol,>
```

---

---

---

---

## Prueba 8

```
/* Mal formados function y bucle while */
let texto string;

print x__x;
function alert (string msg_)
{
   print msg_;
}
pideTexto ()
{
   while(5>n);
}
pideTexto();
  alert
   (texto);
```

---

- **ERRORES**

```
ERROR FOUND USING THE SINTACTIC ANALYZER
####################
Error #107 @ line 11: Se esperaba ';'
- TRAZA -> D    1 32 21 25 23 1 35 37 29 9 15 19 14 2 53 28 54 25 57 40 35 37 29
9 15 19 14 41 1 35 36 52 46
- ÚLTIMO TK LEIDO -> <KeyOpen,>


ERROR FOUND USING THE SINTACTIC ANALYZER
####################
Error #100 @ line 12: Error sintáctico genérico
- TRAZA -> D    1 32 21 25 23 1 35 37 29 9 15 19 14 2 53 28 54 25 57 40 35 37 29
```

```
9 15 19 14 41 1 35 36 52 46
- ÚLTIMO TK LEIDO -> <LoopWhile,>
```

---

---

---

---

## Prueba 9

```
let n1 int = fun();
let mentira boolean = false;
function badExp(int n2){
    do{
        if(input)
            print n1 && 5;
        n2++;
    }while(n2>9);
    return n1;
}
badExp(12,a, a > 3);
```

---

- **ERRORS**

```
ERROR FOUND USING THE SINTACTIC ANALYZER
####################
Error #117 @ line 1: Necesaria previa declaracion de las funciones
- TRAZA -> D    1 32 21 24 22 20 9 15
- ÚLTIMO TK LEIDO -> <ParOpen,>


ERROR FOUND USING THE SEMANTIC ANALYZER
####################
Error #204 @ line 1: Numero de argumentos invalidos para la funcion: fun

ERROR FOUND USING THE SEMANTIC ANALYZER
####################
Error #221 @ line 5: Intentando declarar una expresión vacía/inválida
- TRAZA -> D    1 32 21 24 22 20 9 15 18 46 14 1 32 21 26 22 20 9 16 7 14 2 53
28 54 24 57 40 33 40 31 42
- ÚLTIMO TK LEIDO -> <ResIn,>
```

```
ERROR FOUND USING THE SINTACTIC ANALYZER
###################
Error #113 @ line 5: Se esperaba el cierre de la expresión: ')'
- TRAZA -> D    1 32 21 24 22 20 9 15 18 46 14 1 32 21 26 22 20 9 16 7 14 2 53
28 54 24 57 40 33 40 31 42
- ÚLTIMO TK LEIDO -> <ResIn,>


ERROR FOUND USING THE SINTACTIC ANALYZER
###################
Error #108 @ line 5: Se esperaba un identificador, ++, return o operacion E-S
- TRAZA -> D    1 32 21 24 22 20 9 15 18 46 14 1 32 21 26 22 20 9 16 7 14 2 53
28 54 24 57 40 33 40 31 42 44
- ÚLTIMO TK LEIDO -> <ParClose,>


ERROR FOUND USING THE SINTACTIC ANALYZER
###################
Error #103 @ line 7: Se esperaba el inicio de la expresión: '('
- TRAZA -> D    1 32 21 24 22 20 9 15 18 46 14 1 32 21 26 22 20 9 16 7 14 2 53
28 54 24 57 40 33 40 31 42 44 40 35 37 29 9 15 19 12 9 16 5 14 40 35 36
- ÚLTIMO TK LEIDO -> <ResAutoSum,>


ERROR FOUND USING THE SINTACTIC ANALYZER
###################
Error #102 @ line 7: Se esperaba una declaración o un condicional
- TRAZA -> D    1 32 21 24 22 20 9 15 18 46 14 1 32 21 26 22 20 9 16 7 14 2 53
28 54 24 57 40 33 40 31 42 44 40 35 37 29 9 15 19 12 9 16 5 14 40 35 36 40
- ÚLTIMO TK LEIDO -> <SemCol,>
```

---

---

---

---

## Prueba 10

```
/*Varibales declaradas */
let texto string;

print ;
```

```
input
esto_es_un_nombre_de_variable_global_de_tipo_entero_que_tiene_que_aparecer_en_la
_TS_global;
print x__x;
function alert (string msg_)
{
    let texto string ="hola";
   print msg_;
}
function pideTexto ()
{
   print "Introduce un texto";
    alert(texto)
   input texto;
}
pideTexto();
  alert
   (nuevaVar);
```

---

- **ERRORES**

```
ERROR FOUND USING THE SEMANTIC ANALYZER
####################
Error #221 @ line 5: Intentando declarar una expresión vacía/inválida
- TRAZA -> D    1 32 21 25 23 1 35 37 29
- ÚLTIMO TK LEIDO -> <SemCol,>


ERROR FOUND USING THE SINTACTIC ANALYZER
####################
Error #107 @ line 17: Se esperaba ';'
- TRAZA -> D    1 32 21 25 23 1 35 37 29 1 35 37 30 1 35 37 29 9 15 19 14 2 53
28 54 25 57 40 32 21 25 22 20 9 16 4 14 40 35 37 29 9 15 19 14 41 2 53 28 55 40
35 37 29 9 16 4 14 40 35 36 52 45 9 15 19 14 48
- ÚLTIMO TK LEIDO -> <ResIn,>


ERROR FOUND USING THE SINTACTIC ANALYZER
####################
Error #103 @ line 17: Se esperaba el inicio de la expresión: '('
- TRAZA -> D    1 32 21 25 23 1 35 37 29 1 35 37 30 1 35 37 29 9 15 19 14 2 53
28 54 25 57 40 32 21 25 22 20 9 16 4 14 40 35 37 29 9 15 19 14 41 2 53 28 55 40
```

```
35 37 29 9 16 4 14 40 35 36 52 45 9 15 19 14 48 40 35 36
- ÚLTIMO TK LEIDO -> <SemCol,>
```

---

---

---

---