Proyecto de Procesadores de Lenguajes

Grupo 17

Jorge Bodega Fernanz – v130323
Federico Pereyra López – x150382
Patricia Rivera Suarez – x150049

20-1-2019

Contenido

1. Diseño

Para la implementación de la práctica hemos optado por el lenguaje de programación Python, así como el uso de la librería PLY. Incluimos toda la librería y su funcionalidad dentro de la carpeta "***./library***".

Las características que contempla nuestro lenguaje, basándonos en las elecciones posibles y en las características comunes a todos los grupos, son:

Comentarios

Como comentario hemos obtenido el tipo de **Comentario de línea (//)**. Comienzan al escribir ambas barras y acaban con un salto de línea. Pueden ir en cualquier parte del lenguaje y no generan token.

Constantes

Tenemos tres tipos de constantes:

- **Enteras**: Representado en decimal, ocupan dos bytes. Pueden ser negativos o positivos y su rango de valores está en [-32767, 32767].
- **Cadenas**: Van encerradas entre comillas dobles y puede aparecer cualquier carácter imprimible.
- **Lógicas**: Existen dos valores para estas constantes, **true** y **false**

Operadores

Los operadores que hemos implementado en la práctica han sido:

- **Aritméticos**: Suma y resta
- **Relacionales**: Igual/No igual
- **Lógicos**: Operación AND
- **Asignación**: Asignación simple
- **Especial**: Post-autoincremento (++ como sufijo)
- **Otros**: Comas, llaves, paréntesis punto-coma.

Identificadores

Los nombres de los identificadores pueden llevar cualquier combinación de letras, número y subrayados (_), siempre que el primero sea una letra. El lenguaje es dependiente de minúsculas o mayúsculas.

Tipos de datos

Hay un tipo de dato por cada tipo de constante: constante entera (tipo **int**), constante cadena (tipo **string**) y constante lógica (tipo **bool**).

Declaraciones

La declaración de las variables debe ser de la forma "***var T id***", siendo T el tipo de dicha variable (entera, cadena o lógica).

Instrucciones de Entrada/Salida

Las instrucciones de entrada/salida implementadas son las siguientes:

- **print** (expresión): Evalúa la expresión (cadena o lógica) e imprime el resultado por pantalla.
- **prompt** (var): Lee un número o una cadena del teclado y lo almacena en la variable parámetro

Sentencias

Las sentencias que se han debido implementar son:

- **Asignación**: Se realiza mediante el (=) de la siguiente manera **[identificador = expresión]**. Como no hay conversión entre tipos, identificador y expresión deben ser del mismo tipo.
- **Llamada a una función**: **[nombreFunción (argumento1, …)]**. El número de argumentos tiene que coincidir con los de la declaración de la función.
- **Retorno de una función**: **[return (expresión)]**. Puede o no haber expresión, en el caso de no haberla, la función debe haber sido declarada sin tipo.
- **Condición simple**: **[if (condición) sentencia]**. Si la condición se evalúa como cierta, de ejecuta la sentencia, en caso contrario, se finaliza la ejecución. La sentencia solo puede ser de tipo asignación, instrucción de entrada/salida o retorno.
- **Sentencia repetitiva**: [**for (inicialización; condición; actualización) {sentencias}]**

Funciones

Se deben definir las funciones antes de poder utilizarlas, la definición se realiza mediante la palabra reservada ***function***. El tipo solo se pone si la función retorna algún valor.

```
[ function [Tipo] nombre (argumento1, …) {
        sentencias
} ]
```

2. Analizador Léxico

Los tokens son los elementos más pequeños del texto y el formato que deben seguir será:

del* < código del*, del* [atributo] del* > del* RE

donde **del*** será cualquier cantidad de espacios en blanco, tabuladores o vacío.

2.1 Tokens

Los tokens que reconoce nuestro analizador léxico son:

Operador suma <op_suma,->
Operador incremento <op_posinc,->
Operador resta <op_resta,->
Operador igual <op_igual,->
Operador noigual <op_noigual,->
Operador and <op_and,->
Operador coma <op_coma,->
Operador asignacion <op_asignacion,->
Operador punto coma <op_ptocoma,->
Operador llaveab <op_llaveab,->
Operador llavecer <op_llavecer,->
Operador parenab <op_parenab,->
Operador parencer <op_parencer,->
Palabra Reservada <PR,pos_TPR>
Identificador <ID,pos_TS>
Cadena <cadena,texto>
Entero <entero,valor>
Final fichero <fin_fich,eof>

Los tokens relacionados con los operadores son autodefinidos, esto nos facilita la identificación directamente por el tipo de token sin necesidad de acceder a valor o a una posición en una tabla.

Las palabras reservadas están incluidas en un array de Strings definido previamente, que nos permite identificar más rápidamente si el texto es una palabra reservada o será un nuevo identificador. Este array predefinido es:

```
palabras_reservadas = (
        # Palabras reservadas de nuestro lenguaje
        'true', 'false', 'var', 'function', 'int', 'bool', 'String',
        'print', 'prompt', 'return', 'for', 'if'
)
```

Los tokens de identificadores contienen su posición en tabla de símbolos, y los tokens de cadena y entero contienen directamente su valor al ser detectados como constantes.

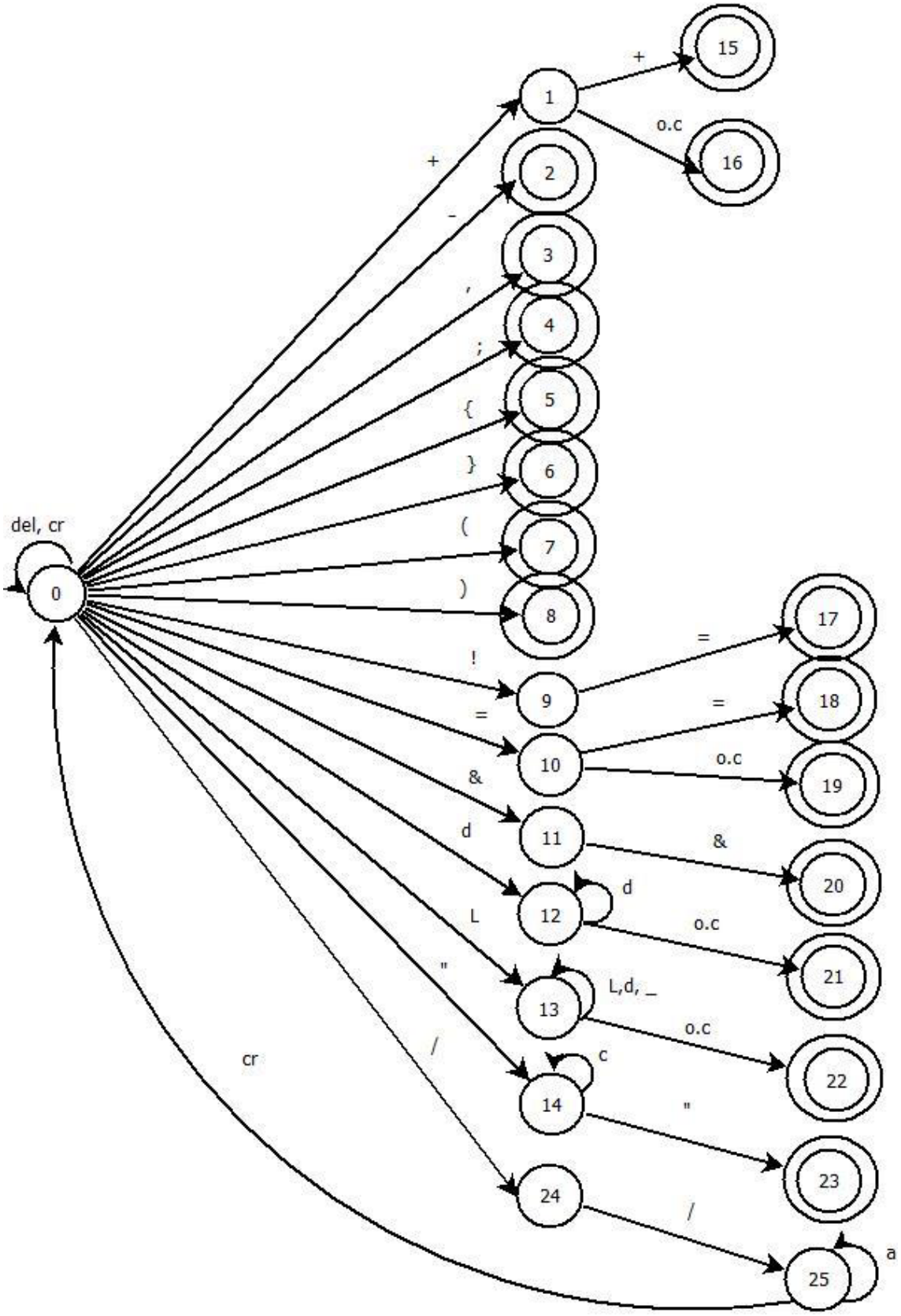
2.2 Gramática Regular

Nuestra gramática regular se compone de:

0: S -> delS| crS | !A | =B |&C| dD| lE |“F | /G | +I | - | , | ; | { | } | (|)
1: A -> =
1: B -> = | λ
2: C -> &
3: D -> dD | λ
4: E -> lE | dE| _E | λ
5: F -> cF | ”
6: G -> /H
7: H -> aH | cr
8: I -> + | λ

d: {0…9}
l: {a-z, A…Z}
c: todo menos “
a: Todo menos “\n”
del: b, TAB
cr: salto de línea

2.3 AFD


del, cr
0
+
-
*
;
{
}
(
)
!
=
&
d
L
"
/
cr
1
2
3
4
5
6
7
8
9
10
11
12
13
14
24
25
+
o.c
=
=
o.c
&
d
o.c
L,d, _
o.c
c
"
/
a
15
16
17
18
19
20
21
22
23

2.4 Acciones Semánticas

Dentro de nuestras acciones semánticas, definimos varias funciones que se crearán posteriormente en el código:

- ***Leer():*** Coge el siguiente carácter.
- ***Generar_Token(tipo, valor):*** Genera un token. El valor es opcional y con operadores no se usa.
- ***Error():*** Genera un error.
- ***BuscarTPR(lex):*** Busca lex en la tabla de palabras reservadas. Devuelve la posición o -1 si no lo encuentra.
- ***BuscarTS(lex):*** Busca lex en la tabla de símbolos. Devuelve la posición o -1 si no lo encuentra.
- ***InsertarTS(lex):*** Inserta el lexema como un nuevo identificador en la tabla de símbolos.

0-0: Leer()
0-1: Leer(), lex := +
1-15: Leer(), lex := lex (+) +
Generar_Token(op_posinc,)
1-16: Generar_Token(op_suma,);
0-2: Leer(), Generar_Token(op_resta,);
0-3: Leer(), Generar_Token(op_coma,);
0-4: Leer(), Generar_Token(op_ptocoma,);
0-5: Leer(), Generar_Token(op_llaveab,);
0-6: Leer(), Generar_Token(op_llavecer,)
0-7: Leer(), Generar_Token(op_parenab,)
0-8: Leer(), Generar_Token(op_parencer,)
0-9: Leer()
9-17: Leer(), Generar_Token(op_noigual,)
0-10: Leer()
10-18: Leer(), Generar_Token(op_igual,)
10-19: Leer(), Generar_Token(op_asignación,)
0-11: Leer()
11-20: Leer(), Generar_Token(op_and,)
0-12: Leer(), dig := d;
12-12: Leer(), dig:= dig*10 (+) d
12-21: if (dig>32767 || dig < -32767)
then Error()
Generar_Token(entero, dig)
0-13: Leer(), lex:=car_leido;
13-13: Leer(), lex:=lex (+) car_leido;
13-22: pos := BuscarTPR(lex)
If(pos != -1) Generar_Token(PR, pos)
pos := BuscarTS(lex)
If(pos != -1) Generar_Token(ID, pos)
Else
pos := InsertarTS(lex)
Generar_Token(ID, pos)
0-14: Leer(), lex :=
14-14: Leer(), lex := lex (+) c
14-23: Leer(), Generar_Token(cadena, lex)
0-24: Leer()
24-25: Leer()
25-25: Leer()
25-0: Leer()

2.5 Errores

El único error contemplado por el sistema es el rango de los números enteros como se definía anteriormente. Todos los demás errores son incumplimientos de las reglas de la gramática.

3. Analizador Sintáctico

3.1 Gramática

La gramática que hemos desarrollado para esta entrega, con ligeros cambios con respecto a la anterior, es:

S0 -> S
S -> A ; S | D ; S | C S | F S | λ
A -> id = E
A1 -> A | λ
A2 -> id A3 | λ
A3 -> = E | ++
A4 -> C A4 | D ; A4 | A ; A4 | λ
D -> var D1 id
D1 -> int | bool | String
D2 -> D1 | λ
C -> if (E) S1 ; | for (A1 ; E ; A2) { A4 } | S2 ;
F -> function D2 id (F1) { A4 }
F1 -> F2 | λ
F2 -> D1 id F3
F3 -> , F2 | λ
S1 -> S2 | A
S2 -> print (E) | prompt (id) | return E1
E -> G E2
E1 -> E | λ
E2 -> && E | λ
G -> H G1
G1 -> == G | != G | λ
H -> I H1
H1 -> + H | - H | λ
I -> id J | (E) | entero | cadena | True | False
J -> ++ | (Z) | λ
Z -> id Z1 | λ
Z1 -> , id Z1 | λ

3.2 First

El conjunto First de cada uno de los elementos No Terminales es:

First(S0) = {λ, id, var, if, for, function, print, prompt, return}
First(S) = {λ, id, var, if, for, function, print, prompt, return}
First(A) = {id}
First(A1) = {id, λ}
First(A2) = {id, λ}
First(A3) = {=, ++}
First(A4) = {id, λ, if, for, var, print, prompt, return}
First(D) = {var}
First(D1) = {int, bool, String}
First(D2) = {int, bool, String, λ}
First(C) = {if, for, print, prompt, return}
First(F) = {function}
First(F1) = {λ, int, bool, String}
First(F2) = {int, bool, String}
First(F3) = {,, λ}
First(S1) = {id, print, prompt, return}
First(S2) = {print, prompt, return}
First(E) = {id, (, entero, cadena, True, False}
First(E1) = {λ, id, (, entero, cadena, True, False}
First(E2) = {&&, λ}
First(G) = {id, (, entero, cadena, True, False}
First(G1) = {==, !=, λ}
First(H) = {id, (, entero, cadena, True, False}
First(H1) = {+, -, λ}
First(I) = {id, (, entero, cadena, True, False}
First(J) = {++, (, λ}
First(Z) = {id, λ}
First(Z1) = {,, λ}

3.3 Follow

El conjunto Follow de cada uno de los elementos No Terminales es:

Follow(S0) = {$}
Follow(S) = {$}
Follow(A) = {;}
Follow(A1) = {;}
Follow(A2) = {)}
Follow(A3) = {)}
Follow(A4) = {}}
Follow(D) = {;}
Follow(D1) = {id}
Follow(D2) = {id}
Follow(C) = {id, var, if, for, function, print, prompt, return, $, }}
Follow(F) = {id, var, if, for, function, print, prompt, return, $}
Follow(F1) = {)}
Follow(F2) = {)}
Follow(F3) = {)}
Follow(S1) = {;}
Follow(S2) = {;}
Follow(E) = {;,)}
Follow(E1) = {;}
Follow(E2) = {;,)}
Follow(G) = {&&, ;,)}
Follow(G1) = {&&, ;,)}
Follow(H) = {==, !=, &&, ;,)}
Follow(H1) = {==, !=, &&, ;,)}
Follow(I) = {+, -, ==, !=, &&, ;,)}
Follow(J) = {+, -, ==, !=, &&, ;,)}
Follow(Z) = {)}
Follow(Z1) = {)}

3.4 Condición LL(1)

S -> {id} ∩ {var} ∩ {if, for, print, prompt, return} ∩ {function} ∩ {$} = Ø
D1 -> {int} ∩ {bool} ∩ {String} = Ø
D2 -> {int, bool, String} ∩ {id} = Ø
A1 -> {id} ∩ {;} = Ø
A2 -> {id} ∩ {)} = Ø
A3 -> {=} ∩ {++} = Ø
A4 -> {if, for, print, prompt, return} ∩ {var} ∩ {id} ∩ {}} = Ø
C -> {if} ∩ {for} ∩ {print, prompt, return} = Ø
F1 -> {int, bool, String} ∩ {)} = Ø
F3 -> {,} ∩ {)} = Ø
S1 -> {print, prompt, return} ∩ {id} = Ø
S2 -> {print} ∩ {prompt} ∩ {return} = Ø
E1 -> {id, (, entero, cadena, True, False} ∩ {;} = Ø
E2 -> {&&} ∩ {;,)} = Ø
G1 -> {==} ∩ {!=} ∩ {&&, =, ++,), ;} = Ø
H1 -> {+} ∩ {-} ∩ {==, !=, &&, =, ++,), ;} = Ø
I -> {id} ∩ {(} ∩ {entero} ∩ {cadena} ∩ {True} ∩ {False} = Ø
J -> {++} ∩ {(} ∩ {+, -, ==, !=, &&,), ;} = Ø
Z -> {id} ∩ {)} = Ø
Z1 -> {,} ∩ {)} = Ø

3.5 Procedures

```
def s0(self):
    print('S0 -> S' if self.flag_imprimir else '')
    self.parse += '1 '
    self.s()

def s(self):
    print('S -> A ; S | D ; S | C S | F S | λ' if self.flag_imprimir else '')
    tipo_token = self.get_tipo_token()
    if self.puntero_tokens == len(self.tokens):
        self.parse += '6 '
    elif tipo_token == 'ID':
        self.parse += '2 '
        self.a()
        self.check_token('op_ptocoma')
        self.s()
    elif tipo_token == 'PR':
        palabra = self.get_palabra_reservada()
        if palabra == 'var':
            self.parse += '3 '
            self.d()
            self.check_token('op_ptocoma')
            self.s()
        elif palabra in ['if', 'for', 'print', 'prompt', 'return']:
            self.parse += '4 '
            self.c()
            self.s()
        elif palabra == 'function':
            self.parse += '5 '
            self.f()
            self.s()

def a(self):
    print('A -> id = E' if self.flag_imprimir else '')
    self.parse += '7 '
    self.check_token('ID')
    self.check_token('op_asignacion')
    self.e()

def a1(self):
    print('A1 -> A | λ' if self.flag_imprimir else '')
    tipo_token = self.get_tipo_token()
    if tipo_token == 'ID':
        self.parse += '8 '
        self.a()
    else:
        self.parse += '9 '
```

```
def a2(self):
    print('A2 -> id A3 | λ' if self.flag_imprimir else '')
    tipo_token = self.get_tipo_token()
    if tipo_token == 'ID':
        self.parse += '10 '
        self.check_token('ID')
        self.a3()
    else:
        self.parse += '11 '

def a3(self):
    print('A3 -> = E | ++' if self.flag_imprimir else '')
    tipo_token = self.get_tipo_token()
    if tipo_token == 'op_asignacion':
        self.parse += '12 '
        self.check_token('op_asignacion')
        self.e()
    elif tipo_token == 'op_posinc':
        self.parse += '13 '
        self.check_token('op_posinc')

def a4(self):
    print('A4 -> C A4 | D ; A4 | A ; A4 | λ' if self.flag_imprimir else '')
    tipo_token = self.get_tipo_token()
    if tipo_token == 'PR':
        palabra = self.get_palabra_reservada()
        if palabra in ['if', 'for', 'print', 'prompt', 'return']:
            self.parse += '14 '
            self.c()
            self.a4()
        elif palabra == 'var':
            self.parse += '15 '
            self.d()
            self.check_token('op_ptocoma')
            self.a4()
    elif tipo_token == 'ID':
        self.parse += '16 '
        self.a()
        self.check_token('op_ptocoma')
        self.a4()
    else:
        self.parse += '17 '

def d(self):
    print('D -> var D1 id' if self.flag_imprimir else '')
    self.parse += '18 '
    self.check_token('PR', 'var')
    self.d1()
    self.check_token('ID')
```

```
def d1(self):
    print('D1 -> int | bool | String' if self.flag_imprimir else '')
    if self.get_tipo_token() == 'PR':
        palabra = self.get_palabra_reservada()
        if palabra == 'int':
            self.parse += '19 '
            self.check_token('PR', 'int')
        elif palabra == 'bool':
            self.parse += '20 '
            self.check_token('PR', 'bool')
        elif palabra == 'String':
            self.parse += '21 '
            self.check_token('PR', 'String')
    else:
        self.error_sintactico()

def d2(self):
    print('D2 -> D1 | λ' if self.flag_imprimir else '')
    tipo_token = self.get_tipo_token()
    if tipo_token == 'PR':
        palabra = self.get_palabra_reservada()
        if palabra in ['int', 'bool', 'String']:
            self.parse += '22 '
            self.d1()
    else:
        self.parse += '23 '

def c(self):
    print('C -> if ( E ) S1 ; | for ( A1 ; E ; A2 ) { A4 } | S2 ;' if self.flag_imprimir else '')
    tipo_token = self.get_tipo_token()
    if tipo_token == 'PR':
        palabra = self.get_palabra_reservada()
        if palabra == 'if':
            self.parse += '24 '
            self.check_token('PR', 'if')
            self.check_token('op_parenab')
            self.e()
            self.check_token('op_parencer')
            self.s1(
            self.check_token('op_ptocoma')
        elif palabra == 'for':
            self.parse += '25 '
            self.check_token('PR', 'for')
            self.check_token('op_parenab')
            self.a1()
            self.check_token('op_ptocoma')
            self.e()
            self.check_token('op_ptocoma')
            self.a2()
            self.check_token('op_parencer')
            self.check_token('op_llaveab')
            self.a4()
            self.check_token('op_llavecer')
```

```
        elif palabra in ['print', 'prompt', 'return']:
            self.parse += '26 '
            self.s2()
            self.check_token('op_ptocoma')

def f(self):
    print('F -> function D2 id ( F1 ) { A4 }' if self.flag_imprimir else '')
    self.parse += '27 '
    self.check_token('PR', 'function')
    self.d2()
    self.check_token('ID')
    self.check_token('op_parenab')
    self.f1()
    self.check_token('op_parencer')
    self.check_token('op_llaveab')
    self.a4()
    self.check_token('op_llavecer')

def f1(self):
    print('F1 -> F2 | λ' if self.flag_imprimir else '')
    tipo_token = self.get_tipo_token()
    if tipo_token == 'PR':
        palabra = self.get_palabra_reservada()
        if palabra in ['int', 'bool', 'String']:
            self.parse += '28 '
            self.f2()
    else:
        self.parse += '29 '

def f2(self):
    print('F2 -> D1 id F3' if self.flag_imprimir else '')
    self.parse += '30 '
    self.d1()
    self.check_token('ID')
    self.f3()

def f3(self):
    print('F3 -> , F2 | λ' if self.flag_imprimir else '')
    tipo_token = self.get_tipo_token()
    if tipo_token == 'op_coma':
        self.parse += '31 '
        self.check_token('op_coma')
        self.f2()
    else:
        self.parse += '32 '
```

```
def s1(self):
    print('S1 -> S2 | A' if self.flag_imprimir else '')
    tipo_tokens = self.get_tipo_token()
    if tipo_tokens == 'PR':
        palabra = self.get_palabra_reservada()
        if palabra in ['print', 'prompt', 'return']:
            self.parse += '33 '
            self.s2()
    else:
        self.parse += '34 '
        self.a()

def s2(self):
    print('S2 -> print ( E ) | prompt ( id ) | return E1' if self.flag_imprimir else '')
    tipo_tokens = self.get_tipo_token()
    if tipo_tokens == 'PR':
        palabra = self.get_palabra_reservada()
        if palabra == 'print':
            self.parse += '35 '
            self.check_token('PR', 'print')
            self.check_token('op_parenab')
            self.e()
            self.check_token('op_parencer')
        elif palabra == 'prompt':
            self.parse += '36 '
            self.check_token('PR', 'prompt')
            self.check_token('op_parenab')
            self.check_token('ID')
            self.check_token('op_parencer')
        elif palabra == 'return':
            self.parse += '37 '
            self.check_token('PR', 'return')
            self.e1()

def e(self):
    print('E -> G E2' if self.flag_imprimir else '')
    self.parse += '38 '
    self.g()
    self.e2()
```

```
def e1(self):
    print('E1 -> E | λ' if self.flag_imprimir else '')
    tipo_token = self.get_tipo_token()
    if tipo_token in ['op_parenab', 'entero', 'cadena'] or tipo_token == 'ID':
        self.parse += '39 '
        self.e()
    elif tipo_token == 'PR':
        palabra = self.get_palabra_reservada()
        if palabra in ['True', 'False']:
            self.parse += '39 '
            self.e()
    else:
        self.parse += '40 '

def e2(self):
    print('E2 -> && E | λ' if self.flag_imprimir else '')
    tipo_token = self.get_tipo_token()
    if tipo_token == 'op_and':
        self.parse += '41 '
        self.check_token('op_and')
        self.e()
    else:
        self.parse += '42 '

def g(self):
    print('G -> H G1' if self.flag_imprimir else '')
    self.parse += '43 '
    self.h()
    self.g1()

def g1(self):
    print('G1 -> == G | != G | λ' if self.flag_imprimir else '')
    tipo_token = self.get_tipo_token()
    if tipo_token == 'op_igual':
        self.parse += '44 '
        self.check_token('op_igual')
        self.g()
    elif tipo_token == 'op_noigual':
        self.parse += '45 '
        self.check_token('op_noigual')
        self.g()
    else:
        self.parse += '46 '

def h(self):
    print('H -> I H1' if self.flag_imprimir else '')
    self.parse += '47 '
    self.i()
    self.h1()
```

```
def h1(self):
    print('H1 -> + H | - H | λ' if self.flag_imprimir else '')
    tipo_token = self.get_tipo_token()
    if tipo_token == 'op_suma':
        self.parse += '48 '
        self.check_token('op_suma')
        self.h()
    elif tipo_token == 'op_resta':
        self.parse += '49 '
        self.check_token('op_resta')
        self.h()
    else:
        self.parse += '50 '

def i(self):
    print('I -> id J | ( E ) | entero | cadena | True | False' if self.flag_imprimir else '')
    tipo_token = self.get_tipo_token()
    if tipo_token == 'ID':
        self.parse += '51 '
        self.check_token('ID')
        self.j()
    elif tipo_token == 'op_parenab':
        self.parse += '52 '
        self.check_token('op_parenab')
        self.e()
        self.check_token('op_parencer')
    elif tipo_token == 'entero':
        self.parse += '53 '
        self.check_token('entero')
    elif tipo_token == 'cadena':
        self.parse += '54 '
        self.check_token('cadena')
    elif tipo_token == 'PR':
        palabra = self.get_palabra_reservada()
        if palabra == 'True':
            self.parse += '55 '
            self.check_token('PR', 'True')
        elif palabra == 'False':
            self.parse += '56 '
            self.check_token('PR', 'False')
```

```
def j(self):
    print('J -> ++ | ( Z ) | λ' if self.flag_imprimir else '')
    tipo_token = self.get_tipo_token()
    if tipo_token == 'op_posinc':
        self.parse += '57 '
        self.check_token('op_posinc')
    elif tipo_token == 'op_parenab':
        self.parse += '58 '
        self.check_token('op_parenab')
        self.z()
        self.check_token('op_parencer')
    else:
        self.parse += '59 '

def z(self):
    print('Z -> id Z1 | λ' if self.flag_imprimir else '')
    tipo_token = self.get_tipo_token()
    if tipo_token == 'ID':
        self.parse += '60 '
        self.check_token('ID')
        self.z1()
    else:
        self.parse += '61 '

def z1(self):
    print('Z1 -> , id Z1 | λ' if self.flag_imprimir else '')
    tipo_token = self.get_tipo_token()
    if tipo_token == 'op_coma':
        self.parse += '62 '
        self.check_token('op_coma')
        self.check_token('ID')
        self.z1()
    else:
        self.parse += '63 '
```

4. Analizador Semántico

Esta ha sido nuestra implementación del analizador semántico basándonos en el analizador sintáctico previamente descrito.

S0 -> { TSG.creaTSG(); despG=0; TS.actual := ^TSG, despl := ^desplG } **S** { LiberaTS(TSG) }
S -> A ; S_1 { S.tipo := if (A.tipo = tipo_ok)
then S_1.tipo
else tipo_error
}
S -> D ; S_1 { S.tipo := if (D.tipo = tipo_ok)
then S_1.tipo
else tipo_error
}
S -> C S_1 { S.tipo := if (C.tipoRet = vacio)
then if (C.tipo = tipo_ok)
then S_1.tipo
else tipo_error
else tipo_error("No puede utilizarse el return fuera de una función")
}
S -> F S_1 { S.tipo := if (F.tipo = tipo_ok)
then S_1.tipo
else tipo_error
}
S -> λ {}
A -> id = E { A.tipo := if (!BuscarTS(id.ent))
then tipo_error("Se esperaba un id válido")
else if (E.tipo != BuscarTipoTS(id.ent))
then tipo_error("No se puede asignar un valor de distinto tipo")
else tipo_ok
}
A1 -> A { A1.tipo := A.tipo }
A1 -> λ {}
A2 -> id A3 { A2.tipo := if (A3.tipo != BuscarTipoTS(id.ent))
then tipo_error
else tipo_ok
}
A2 -> λ {}
A3 -> = E { A3.tipo := E.tipo }
A3 -> ++ { A3.tipo := ent }
A4 -> C $A4_1$ { A4.tipoRet := if (C.tipoRet != vacio)
then C.tipoRet
else if ($A4_1$.tipoRet != vacio)
then $A4_1$.tipoRet
else vacio ;
A4.tipo := if (C.tipo != tipo_ok)
then tipo_error
else if (C.tipoRet != vacio)
then tipo_ok
else $A4_1$.tipo
}

A4 -> D ; A4$_1$ { A4.tipoRet := A4$_1$.tipoRet ;
A4.tipo := if(D1.tipo = tipo_ok)
then A4$_1$.tipo
else tipo_error
}
A4 -> A ; A4$_1$ { A4.tipoRet := A4$_1$.tipoRet ;
A4.tipo := if(A.tipo = tipo_ok)
then A4$_1$.tipo
else tipo_error
}
A4 -> λ { A4.tipoRet := vacio }
D -> var D1 id { D.tipo := tipo_ok ; insertarTipoTSG(id.ent, D1.tipo, ent.desp) }
D1 -> int { D1.tipo := ent }
D1 -> bool { D1.tipo := log }
D1 -> String { D1.tipo := cad }
D2 -> D1 { D2.tipo := D1.tipo }
D2 -> λ { D2.tipo := vacio }
C -> if (E) S1 ; { C.tipoRet := S1.tipoRet ;
C.tipo := if (E.tipo != log)
then tipo_error("Expresión en if debe ser logica")
else S1.tipo
}
C -> for (A1 ; E ; A2) { A4 } { C.tipoRet := A4.tipoRet ;
C.tipo := if (E.tipo != log)
then tipo_error("Expresión en for debe ser logica")
else if(A1.tipo = A2.tipo = A4.tipo = tipo_ok)
then tipo_ok
else tipo_error
}
C -> S2 ; { C.tipoRet := S2.tipoRet ; C.tipo := S2.tipo }
F -> function D2 id (F1) { A4 } { insertarTipoRetornoFunctionTSG(id.ent, D2.tipo) ;
F.tipo := if (F1.tipo != tipo_ok)
then tipo_error
else if (A4.tipo != tipo_ok)
then tipo_error
else if (D2.tipo = A4.tipoRet)
then tipo_ok
else tipo_error("El tipo de la función no es el mismo
que el retornado")
}
F1 -> F2 { F1.tipo := F2.tipo }
F1 -> λ {}
F2 -> D1 id F3 { F2.tipo := F3.tipo }
F3 -> , F2 { F3.tipo := tipo_ok }
F3 -> λ {}
S1 -> S2 {S1.tipoRet := S2.tipoRet ; S1.tipo := S2.tipo }
S1 -> A {S1.tipoRet := vacio ; S1.tipo := A.tipo }

```
S2 -> print ( E ) { S1.tipoRet := vacio ; S1.tipo := E.tipo }
S2 -> prompt ( id ) {       S1.tipoRet := vacio ;
                            S1.tipo := if (!BuscarTS(id.ent))
                                        then   tipo_error("Se esperaba un id válido")
                                        else   tipo_ok
                    }
S2 -> return E1 {  S1.tipoRet := E1.tipo ;
                    S1.tipo := if (E1.tipo = tipo_error)
                                then tipo_error
                                else tipo_ok
                    }
E -> G E2 {         E.tipo := if (E2.tipo = vacio)
                            then G.tipo
                            else if (G.tipo != log)
                                 then tipo_error("Se esperaba un valor de tipo lógico")
                                 else if (E2.tipo = tipo_error)
                                      then tipo_error
                                      else log
          }
E1 -> E { E1.tipo := E.tipo }
E1 -> λ {}
E2 -> && E {        E2.tipo := if (E.tipo = log)
                                then tipo_ok
                                else tipo_error("Se esperaba un valor de tipo lógico")
          }
E2 -> λ { E2.tipo := vacio }
G -> H G1 {         G.tipo := if (G1.tipo = vacio)
                            then H.tipo
                            else if (H.tipo != ent)
                                 then tipo_error("Se esperaba un valor de tipo entero")
                                 else if (G1.tipo = tipo_error)
                                      then tipo_error
                                      else log
          }
G1 -> == G {        G1.tipo := if (G.tipo = ent)
                                then tipo_ok
                                else tipo_error("Se esperaba un valor de tipo entero")
          }
G1 -> != G {         G1.tipo := if (G.tipo = ent)
                                 then tipo_ok
                                 else tipo_error("Se esperaba un valor de tipo entero")
          }
G1 -> λ { G1.tipo := vacio }
H -> I H1 {         H.tipo := if (H1.tipo = vacio)
                            then I.tipo
                            else if (I.tipo != ent)
                                 then tipo_error("Se esperaba un valor de tipo entero")
                                 else if (H1.tipo = tipo_error)
                                      then tipo_error
                                      else ent
          }
```

```
H1 -> + H {         H1.tipo := if (H.tipo != ent)
                               then tipo_error("Se esperaba un valor de tipo entero")
                               else tipo_ok
          }
H1 -> - H {         H1.tipo := if (H.tipo != ent)
                               then tipo_error("Se esperaba un valor de tipo entero")
                               else tipo_ok
          }
H1 -> λ { H1.tipo := vacio }
I -> id J {         I.tipo := if (J.tipo = tipo_error)
                              then tipo_error
                              else if (J.tipoInc = ent)
                                   then if (BuscarTipoTS(id.ent) != ent)
                                        then tipo_error("Se esperaba un tipo de dato entero para
                                        autoincrementar")
                                        else ent
                              else if (J.tipoInc = funcion)
                                   then if (BuscarTipoTS(id.ent) != function)
                                         then tipo_error("Se esperaba identificador de tipo función")
                                         else getTipoRetorno(id.ent)
                                   else BuscarTipoTS(id.ent)
          }
I -> ( E ) { I.tipo := E.tipo }
I -> entero { I.tipo := ent )
I -> cadena { I.tipo := cad }
I -> True { I.tipo := log }
I -> False { I.tipo := log }
J -> ++ { J.tipoInc := ent ; J.tipo := tipo_ok }
J -> ( Z ) {J.tipoInc := funcion ; J.tipo := Z.tipo }
J -> λ { j.tipoInc := vacio }
Z -> id Z1 {        Z.tipo := if (!BuscarTS(id.ent))
                              then tipo_error
                              else Z1.tipo
          }
Z -> λ {}
Z1 -> , id Z1₁ {    Z1.tipo := if (!BuscarTS(id.ent))
                               then tipo_error
                               else Z1₁.tipo
          }
Z1 -> λ {}
```

5. Tabla de Símbolos

Las Tablas de Símbolos (TS) son estructuras de datos que almacenan toda la información de los identificadores del lenguaje fuente. Un fichero con las TS podrá tener líneas en blanco o líneas con la información de la TS. Cada línea de información debe acabar obligatoriamente con un salto de línea. Existen tres tipos de líneas obligatorias, cada una de ellas con un formato diferente (definido en la web de la asignatura):

- **Línea con el número de la TS**: Esta línea se utiliza como encabezado para comenzar cada TS. El formato de esta línea es:

  pal* # del* num del* : del* RC

- **Línea del lexema:** Esta línea se utiliza para indicar cada entrada de la TS.
  El formato de esta línea es:

  del* * del* [LEXEMA del* :] del* 'nombre' del* RC

- **Línea de atributo:** Esta línea se utiliza para indicar cada atributo perteneciente a la entrada previa indicada en la 'línea del lexema' de la TS.
  El formato de esta línea es:

  del* + del* atributo del* : del* valor RC

En caso de que se trate de una función, se guarda los datos necesarios de esa función en una entrada en la TS, pero a su vez apunta a una tabla temporal que se ira rellenando durante la ejecución de dicha función .

La información de un elemento de nuestro lenguaje se almacenará en los atributos de dicho elemento. Estos atributos son:

- **Tipo**: representa el tipo del identificador
- **Despl**: dirección relativa de cada variable
- **numParam**: Número de parámetros formales (Solo subprogramas)
- **TipoParamXX**: Tipo del parámetro. XX representa un numero en [01, numParam] (Solo subprogramas)
- **ModoParamXX**: Modo de paso de parámetros. No lo contemplamos en nuestra implementación.
- **TipoRetorno**: Tipo devuelto por un subprograma

Ejemplo de Tabla de Símbolos con un formato correcto en un único fichero

```
TABLA PRINCIPAL #1:
*       LEXEMA : 'suma'
        ATRIBUTOS:
                + tipo: 'int'
                + numParam: 2
                + tipoparam1: 'int'
                + tipoparam2:'float'
                + Despl: 6
```

Anexo

Se componen de cinco pruebas que se ejecutan de forma correcta, y cinco pruebas que no se ejecutan de forma correcta. Para el manejo de errores, cuando se detecta uno, se produce la finalización del programa, pero mantiene la información generada de tabla de símbolos y del analizador léxico.

Debido a que el tamaño de algunos de los arboles es muy grande, incluimos cual va a ser el fichero que contiene su árbol, dentro del sistema de carpeta que contiene el CD con nuestra implementación.

Prueba 1

Funcionamiento correcto

```
// Incluimos todos los tipos de variables
// contempladas para comprobar el
// funcionamiento
var String textoPrueba; // Declaración
var int enteroPrueba; // Declaración de variable
var bool boolPrueba; // Declaración de variable
textoPrueba = "Prueba"; // Inicialización
enteroPrueba= 55; // Inicialización
boolPrueba = True; // Inicialización

patata = 17; // Inicialización de variable global
```

```
CONTENIDO DE LA TABLA # TSGeneral
*       LEXEMA: 'textoPrueba'
        ATRIBUTOS:
        +       Tipo : cadena
        +       Despl : 0
--------------------
*       LEXEMA: 'enteroPrueba'
        ATRIBUTOS:
        +       Tipo : entero
        +       Despl : 256
--------------------
*       LEXEMA: 'boolPrueba'
        ATRIBUTOS:
        +       Tipo : logico
        +       Despl : 258
--------------------
*       LEXEMA: 'patata'
        ATRIBUTOS:
        +       Tipo : entero
        +       Despl : 259
--------------------
==============================
```

Sus tokens generados serán:

```
<PR, 3>
<PR, 7>
<ID, 1>
<op_ptocoma, ->
<PR, 3>
<PR, 5>
<ID, 2>
<op_ptocoma, ->
<PR, 3>
<PR, 6>
<ID, 3>
<op_ptocoma, ->
<ID, 1>
<op_asignacion, ->
<cadena, Prueba>
<op_ptocoma, ->
<ID, 2>
<op_asignacion, ->
<entero, 55>
<op_ptocoma, ->
<ID, 3>
<op_asignacion, ->
<PR, 1>
<op_ptocoma, ->
<ID, 4>
<op_asignacion, ->
<entero, 17>
<op_ptocoma, ->
```

Su fichero con el árbol resultado es ***./Caso1/arbol1.html***
Su fichero de parse resultante es:

Descendente 1 3 18 21 3 18 19 3 18 20 2 7 38 43 47 54 50 46 42 2 7 38 43 47 53 50 46 42 2 7 38 43 47 55 50 46 42 2 7 38 43 47 53 50 46 42 6

Funcionamiento incorrecto

```
// Incluimos todos los tipos de variables
// contempladas para comprobar el
// funcionamiento
var String textoPrueba; // Declaración de variable cadena
var int enteroPrueba; // Declaración de variable entera
var bool boolPrueba: // Declaración de variable booleana
textoPrueba = "Prueba"; // Inicialización
enteroPrueba= 55; // Inicialización
boolPrueba = True; // Inicialización

patata = 17; // Inicialización de variable global
```

El mensaje de error resultado es:

Error Léxico (Línea 6): Caracter no permitido ":"

Prueba 2

Funcionamiento correcto

```
// Definimos una función con parámetros

var int a;

for (a = 0; True; a++) {
  a = a + 5;
}

if (True && False) print(10 + 7);
if (True && False) prompt(a);

function int valor0 () {
  var int j;
  return 0;
}

var String prueba;
```

```
CONTENIDO DE LA TABLA # TSGeneral

*       LEXEMA: 'a'
        ATRIBUTOS:
        +       Tipo : entero
        +       Despl : 0
--------------------
*       LEXEMA: 'valor0'
        ATRIBUTOS:
        +       Tipo : funcion
        +       Despl : 2
        +       numParam : 0
        +       TipoRetorno : entero
--------------------
*       LEXEMA: 'prueba'
        ATRIBUTOS:
        +       Tipo : cadena
        +       Despl : 4
--------------------
==============================

CONTENIDO DE LA TABLA # TS_valor0

*       LEXEMA: 'j'
        ATRIBUTOS:
        +       Tipo : entero
        +       Despl : 0
--------------------
==============================
```

Sus tokens generados serán:

<PR, 3>
<PR, 5>
<ID, 1>
<op_ptocoma, ->
<PR, 11>
<op_parenab, ->
<ID, 1>
<op_asignacion, ->
<entero, 0>
<op_ptocoma, ->
<PR, 1>
<op_ptocoma, ->
<ID, 1>
<op_posinc, ->
<op_parencer, ->
<op_llaveab, ->
<ID, 1>
<op_asignacion, ->
<ID, 1>
<op_suma, ->
<entero, 5>
<op_ptocoma, ->
<op_llavecer, ->
<PR, 12>
<op_parenab, ->
<PR, 1>
<op_and, ->
<PR, 2>
<op_parencer, ->
<PR, 8>
<op_parenab, ->
<entero, 10>
<op_suma, ->
<entero, 7>
<op_parencer, ->
<op_ptocoma, ->
<PR, 12>
<op_parenab, ->
<PR, 1>
<op_and, ->
<PR, 2>
<op_parencer, ->
<PR, 9>
<op_parenab, ->
<ID, 1>
<op_parencer, ->
<op_ptocoma, ->
<PR, 4>
<PR, 5>
<ID, 2>
<op_parenab, ->
<op_parencer, ->
<op_llaveab, ->
<PR, 3>
<PR, 5>
<ID, 3>
<op_ptocoma, ->
<PR, 10>
<entero, 0>
<op_ptocoma, ->
<op_llavecer, ->
<PR, 3>
<PR, 7>
<ID, 4>
<op_ptocoma, ->

Su fichero con el árbol resultado es **./Caso2/arbol2.html**
Su parse resultante es:

Descendente 1 3 18 19 4 25 8 7 38 43 47 53 50 46 42 38 43 47 55 50 46 42 10 13 16 7 38 43 47 51 59 48 47 53 50 46 42 17 4 24 38 43 47 55 50 46 41 38 43 47 56 50 46 42 33 35 38 43 47 53 48 47 53 50 46 42 4 24 38 43 47 55 50 46 41 38 43 47 56 50 46 42 33 36 5 27 22 19 29 15 18 19 14 26 37 39 38 43 47 53 50 46 42 17 3 18 21 6

Funcionamiento incorrecto

```
// Definimos una funcion con parámetros

var int a;

for (True; a++) {
  a = a + 5;
}

if (True && False) print(10 + 7); // Probamos la declaración de if con prints y prompts
if (True && False) prompt(a); // Probamos la declaración de if con prints y prompts

function int valor0 () {
  var int j; // Probamos la declaración de variables
  return 0;
}

var String prueba;
```

El mensaje de error resultado es:

Error Sintáctico (Línea 5): Token <PR, 1> erróneo

Prueba 3

Funcionamiento correcto

```
// Probamos algunas palabras reservadas
function int suma (int a, int b) {
   return a + b;
}

function bool logica () {
   return True;
   return 0;
}

function sumaSinRetorno (int a, int b) {
   var int c;
   c = a + b;
}

function int bucle (int limite) {
   var int i;
   for (i = 0; True; i++){
      print(i);
      if (i == 10) return 10;
   }
}
```

CONTENIDO DE LA TABLA # TSGeneral

```
*       LEXEMA: 'suma'
        ATRIBUTOS:
        +       Tipo : funcion
        +       Despl : 0
        +       numParam : 2
        +       TipoRetorno : entero
        +       TipoParam1 : entero
        +       TipoParam2 : entero
--------------------
*       LEXEMA: 'logica'
        ATRIBUTOS:
        +       Tipo : funcion
        +       Despl : 2
        +       numParam : 0
        +       TipoRetorno : logico
--------------------
*       LEXEMA: 'sumaSinRetorno'
        ATRIBUTOS:
        +       Tipo : funcion
        +       Despl : 4
        +       numParam : 2
        +       TipoRetorno : vacio
        +       TipoParam1 : entero
        +       TipoParam2 : entero
--------------------
*       LEXEMA: 'bucle'
        ATRIBUTOS:
        +       Tipo : funcion
        +       Despl : 6
        +       numParam : 1
        +       TipoRetorno : entero
        +       TipoParam1 : entero
--------------------
==============================
```

CONTENIDO DE LA TABLA # TS_suma

```
*       LEXEMA: 'a'
        ATRIBUTOS:
        +       Tipo : entero
        +       Despl : 0
--------------------
*       LEXEMA: 'b'
        ATRIBUTOS:
        +       Tipo : entero
        +       Despl : 2
--------------------
==============================
```

CONTENIDO DE LA TABLA # TS_logica

```
==============================
```

CONTENIDO DE LA TABLA # TS_sumaSinRetorno

```
*       LEXEMA: 'a'
        ATRIBUTOS:
        +       Tipo : entero
        +       Despl : 0
--------------------
*       LEXEMA: 'b'
        ATRIBUTOS:
        +       Tipo : entero
        +       Despl : 2
--------------------
*       LEXEMA: 'c'
        ATRIBUTOS:
        +       Tipo : entero
        +       Despl : 4
--------------------
==============================
```

CONTENIDO DE LA TABLA # TS_bucle

```
*       LEXEMA: 'limite'
        ATRIBUTOS:
        +       Tipo : entero
        +       Despl : 0
--------------------
*       LEXEMA: 'i'
        ATRIBUTOS:
        +       Tipo : entero
        +       Despl : 2
--------------------
==============================
```

Sus tokens generados serán:

<PR, 4>
<PR, 5>
<ID, 1>
<op_parenab, ->
<PR, 5>
<ID, 2>
<op_coma, ->
<PR, 5>
<ID, 3>
<op_parencer, ->
<op_llaveab, ->
<PR, 10>
<ID, 2>
<op_suma, ->
<ID, 3>
<op_ptocoma, ->
<op_llavecer, ->
<PR, 4>
<PR, 6>
<ID, 4>
<op_parenab, ->
<op_parencer, ->
<op_llaveab, ->
<PR, 10>
<PR, 1>
<op_ptocoma, ->
<PR, 10>
<entero, 0>
<op_ptocoma, ->
<op_llavecer, ->
<PR, 4>
<ID, 5>
<op_parenab, ->
<PR, 5>
<ID, 2>
<op_coma, ->
<PR, 5>
<ID, 3>
<op_parencer, ->
<op_llaveab, ->
<PR, 3>
<PR, 5>
<ID, 6>
<op_ptocoma, ->
<ID, 6>
<op_asignacion, ->
<ID, 2>
<op_suma, ->
<ID, 3>
<op_ptocoma, ->
<op_llavecer, ->
<PR, 4>
<PR, 5>
<ID, 7>
<op_parenab, ->
<PR, 5>
<ID, 8>
<op_parencer, ->
<op_llaveab, ->
<PR, 3>
<PR, 5>
<ID, 9>
<op_ptocoma, ->
<PR, 11>
<op_parenab, ->
<ID, 9>
<op_asignacion, ->
<entero, 0>
<op_ptocoma, ->
<PR, 1>
<op_ptocoma, ->
<ID, 9>
<op_posinc, ->
<op_parencer, ->
<op_llaveab, ->
<PR, 8>
<op_parenab, ->
<ID, 9>
<op_parencer, ->
<op_ptocoma, ->
<PR, 12>
<op_parenab, ->
<ID, 9>
<op_igual, ->
<entero, 10>
<op_parencer, ->
<PR, 10>
<entero, 10>
<op_ptocoma, ->
<op_llavecer, ->
<op_llavecer, ->

Su fichero con el árbol resultado es **./Caso3/arbol3.html**
Su parse resultante es:

Descendente 1 5 27 22 19 28 30 19 31 30 19 32 14 26 37 39 38 43 47 51 59 48 47 51 59 50 46 42 17 5 27 22 20 29 14 26 37 39 38 43 47 55 50 46 42 14 26 37 39 38 43 47 53 50 46 42 17 5 27 23 28 30 19 31 30 19 32 15 18 19 16 7 38 43 47 51 59 48 47 51 59 50 46 42 17 5 27 22 19 28 30 19 32 15 18 19 14 25 8 7 38 43 47 53 50 46 42 38 43 47 55 50 46 42 10 13 14 26 35 38 43 47 51 59 50 46 42 14 24 38 43 47 51 59 50 44 43 47 53 50 46 42 33 37 39 38 43 47 53 50 46 42 17 17 6

Funcionamiento incorrecto

```
var int a;
var String b;
var bool c;

a = 2;
b = "Prueba";
c = 1;
```

El mensaje de error resultado es:

Error Semántico (Línea 7): El tipo de dato es diferente al tipo de identificador.

Prueba 4

Funcionamiento correcto

```
// Probamos algunas palabras reservadas

var String a;
var bool b;
var int c;

function String funcion1 () {
   global1 = 1;
   var String retorno;
   retorno = "Valor de prueba para el retorno";
   return retorno;
}

function bool funcion2 () {
   global2 = 2;
   var bool retorno;
   retorno = True;
   return retorno;
}

function int funcion3 () {
   global3 = 3;
   var int retorno;
   retorno = 1337;
   return retorno;
}

a = funcion1();
b = funcion2();
c = funcion3();

if (b && (c == c)) print("True");
```

```
CONTENIDO DE LA TABLA # TSGeneral

*       LEXEMA: 'a'
        ATRIBUTOS:
        +       Tipo : cadena
        +       Despl : 0
--------------------
*       LEXEMA: 'b'
        ATRIBUTOS:
        +       Tipo : logico
        +       Despl : 256
--------------------
*       LEXEMA: 'c'
        ATRIBUTOS:
        +       Tipo : entero
        +       Despl : 257
--------------------
*       LEXEMA: 'funcion1'
        ATRIBUTOS:
        +       Tipo : funcion
        +       Despl : 259
        +       numParam : 0
        +       TipoRetorno : cadena
--------------------
*       LEXEMA: 'global1'
        ATRIBUTOS:
        +       Tipo : entero
        +       Despl : 261
--------------------
*       LEXEMA: 'funcion2'
        ATRIBUTOS:
        +       Tipo : funcion
        +       Despl : 263
        +       numParam : 0
        +       TipoRetorno : logico
--------------------
*       LEXEMA: 'global2'
        ATRIBUTOS:
        +       Tipo : entero
        +       Despl : 265
--------------------
*       LEXEMA: 'funcion3'
        ATRIBUTOS:
        +       Tipo : funcion
        +       Despl : 267
        +       numParam : 0
        +       TipoRetorno : entero
--------------------
*       LEXEMA: 'global3'
        ATRIBUTOS:
        +       Tipo : entero
        +       Despl : 269
--------------------
==============================

CONTENIDO DE LA TABLA # TS_funcion1

*       LEXEMA: 'retorno'
        ATRIBUTOS:
        +       Tipo : cadena
        +       Despl : 0
--------------------
==============================

CONTENIDO DE LA TABLA # TS_funcion2

*       LEXEMA: 'retorno'
        ATRIBUTOS:
        +       Tipo : logico
        +       Despl : 0
--------------------
==============================

CONTENIDO DE LA TABLA # TS_funcion3

*       LEXEMA: 'retorno'
        ATRIBUTOS:
        +       Tipo : entero
        +       Despl : 0
--------------------
==============================
```

Sus tokens generados serán:

<PR, 3>
<PR, 7>
<ID, 1>
<op_ptocoma, ->
<PR, 3>
<PR, 6>
<ID, 2>
<op_ptocoma, ->
<PR, 3>
<PR, 5>
<ID, 3>
<op_ptocoma, ->
<PR, 4>
<PR, 7>
<ID, 4>
<op_parenab, ->
<op_parencer, ->
<op_llaveab, ->
<ID, 5>
<op_asignacion, ->
<entero, 1>
<op_ptocoma, ->
<PR, 3>
<PR, 7>
<ID, 6>
<op_ptocoma, ->
<ID, 6>
<op_asignacion, ->
<cadena, Valor de prueba
para el retorno>
<op_ptocoma, ->
<PR, 10>
<ID, 6>
<op_ptocoma, ->
<op_llavecer, ->
<PR, 4>
<PR, 6>
<ID, 7>
<op_parenab, ->
<op_parencer, ->
<op_llaveab, ->
<ID, 8>
<op_asignacion, ->
<entero, 2>
<op_ptocoma, ->
<PR, 3>
<PR, 6>
<ID, 6>
<op_ptocoma, ->
<ID, 6>
<op_asignacion, ->
<PR, 1>
<op_ptocoma, ->
<PR, 10>
<ID, 6>
<op_ptocoma, ->
<op_llavecer, ->
<PR, 4>
<PR, 5>
<ID, 9>
<op_parenab, ->
<op_parencer, ->
<op_llaveab, ->
<ID, 10>
<op_asignacion, ->
<entero, 3>
<op_ptocoma, ->
<PR, 3>
<PR, 5>
<ID, 6>
<op_ptocoma, ->
<ID, 6>
<op_asignacion, ->
<entero, 1337>
<op_ptocoma, ->
<PR, 10>
<ID, 6>
<op_ptocoma, ->
<op_llavecer, ->
<ID, 1>
<op_asignacion, ->
<ID, 4>
<op_parenab, ->
<op_parencer, ->
<op_ptocoma, ->
<ID, 2>
<op_asignacion, ->
<ID, 7>
<op_parenab, ->
<op_parencer, ->
<op_ptocoma, ->
<ID, 3>
<op_asignacion, ->
<ID, 9>
<op_parenab, ->
<op_parencer, ->
<op_ptocoma, ->
<PR, 12>
<op_parenab, ->
<ID, 2>
<op_and, ->
<op_parenab, ->
<ID, 3>
<op_igual, ->
<ID, 3>
<op_parencer, ->
<op_parencer, ->
<PR, 8>
<op_parenab, ->
<cadena, True>
<op_parencer, ->
<op_ptocoma, ->

Su fichero con el árbol resultado es **./Caso4/arbol4.html**
Su parse resultante es:

Descendente 1 3 18 21 3 18 20 3 18 19 5 27 22 21 29 16 7 38 43 47 53 50 46 42 15 18 21 16 7 38 43 47 54 50 46 42 14 26 37 39 38 43 47 51 59 50 46 42 17 5 27 22 20 29 16 7 38 43 47 53 50 46 42 15 18 20 16 7 38 43 47 55 50 46 42 14 26 37 39 38 43 47 51 59 50 46 42 17 5 27 22 19 29 16 7 38 43 47 53 50 46 42 15 18 19 16 7 38 43 47 53 50 46 42 14 26 37 39 38 43 47 51 59 50 46 42 17 2 7 38 43 47 51 58 61 50 46 42 2 7 38 43 47 51 58 61 50 46 42 2 7 38 43 47 51 58 61 50 46 42 4 24 38 43 47 51 59 50 46 41 38 43 47 52 38 43 47 51 59 50 44 43 47 51 59 50 46 42 50 46 42 33 35 38 43 47 54 50 46 42 6

Funcionamiento incorrecto

```
// Probamos algunas palabras reservadas

var String a;
var bool b;
var int c;

function String funcion1 () {
   global1 = 1;
   var String retorno;
   retorno = "Valor de prueba para el retorno";
   return retorno;
}

function bool funcion2 () {
   global2 = 2;
   var bool retorno;
   retorno = True;
   return retorno;
}

function int funcion3 () {
   global3 = 3;
   var int retorno;
   retorno = 1337;
}

a = funcion1();
b = funcion2();
c = funcion3();

if (b && (c == c)) print("True");
```

El mensaje de error resultado es:

Error Semántico (Línea 25): El tipo de retorno no coincide con el especificado en la declaración.

Prueba 5

Funcionamiento correcto

```
// Probamos algunas palabras reservadas
var int a;
var int resultado;

function int multibucle (int a, int n) {
   resultado = 0;
   var int i;
   var int j;
   var int z;
   for (i = 0; True && False; i++) {
      for (j = i; True; j = i + 100){
         for (z = i + j; False; z = i + j + j){
            resultado = i + j + z;
         }
      }
   }
   print(resultado);
   prompt(i);
   return resultado;
}

function int recursividad (int p) {
   var int b;
   var int suma;
   suma = p + 1;
   b = recursividad(suma);
   if (b == 10) return b;
}
```

CONTENIDO DE LA TABLA # TSGeneral

```
*  LEXEMA: 'a'
   ATRIBUTOS:
   +  Tipo : entero
   +  Despl : 0
--------------------
*  LEXEMA: 'resultado'
   ATRIBUTOS:
   +  Tipo : entero
   +  Despl : 2
--------------------
*  LEXEMA: 'multibucle'
   ATRIBUTOS:
   +  Tipo : funcion
   +  Despl : 4
   +  numParam : 2
   +  TipoRetorno : entero
   +  TipoParam1 : entero
   +  TipoParam2 : entero
--------------------
*  LEXEMA: 'recursividad'
   ATRIBUTOS:
   +  Tipo : funcion
   +  Despl : 6
   +  numParam : 1
   +  TipoRetorno : entero
   +  TipoParam1 : entero
--------------------
==============================
```

CONTENIDO DE LA TABLA # TS_multibucle

```
*  LEXEMA: 'a'
   ATRIBUTOS:
   +  Tipo : entero
   +  Despl : 0
--------------------
*  LEXEMA: 'n'
   ATRIBUTOS:
   +  Tipo : entero
   +  Despl : 2
--------------------
*  LEXEMA: 'i'
   ATRIBUTOS:
   +  Tipo : entero
   +  Despl : 4
--------------------
*  LEXEMA: 'j'
   ATRIBUTOS:
   +  Tipo : entero
   +  Despl : 6
--------------------
*  LEXEMA: 'z'
   ATRIBUTOS:
   +  Tipo : entero
   +  Despl : 8
--------------------
==============================
```

CONTENIDO DE LA TABLA # TS_recursividad

```
*  LEXEMA: 'p'
   ATRIBUTOS:
   +  Tipo : entero
   +  Despl : 0
--------------------
*  LEXEMA: 'b'
   ATRIBUTOS:
   +  Tipo : entero
   +  Despl : 2
--------------------
*  LEXEMA: 'suma'
   ATRIBUTOS:
   +  Tipo : entero
   +  Despl : 4
--------------------
==============================
```

Sus tokens generados serán:

<PR, 3>
<PR, 5>
<ID, 1>
<op_ptocoma, ->
<PR, 3>
<PR, 5>
<ID, 2>
<op_ptocoma, ->
<PR, 4>
<PR, 5>
<ID, 3>
<op_parenab, ->
<PR, 5>
<ID, 1>
<op_coma, ->
<PR, 5>
<ID, 4>
<op_parencer, ->
<op_llaveab, ->
<ID, 2>
<op_asignacion, ->
<entero, 0>
<op_ptocoma, ->
<PR, 3>
<PR, 5>
<ID, 5>
<op_ptocoma, ->
<PR, 3>
<PR, 5>
<ID, 6>
<op_ptocoma, ->
<PR, 3>
<PR, 5>
<ID, 7>
<op_ptocoma, ->
<PR, 11>
<op_parenab, ->
<ID, 5>
<op_asignacion, ->
<entero, 0>
<op_ptocoma, ->
<PR, 1>
<op_and, ->
<PR, 2>
<op_ptocoma, ->
<ID, 5>
<op_posinc, ->
<op_parencer, ->
<op_llaveab, ->
<PR, 11>
<op_parenab, ->
<ID, 6>
<op_asignacion, ->
<ID, 5>
<op_ptocoma, ->
<PR, 1>
<op_ptocoma, ->
<ID, 6>
<op_asignacion, ->
<ID, 5>
<op_suma, ->
<entero, 100>
<op_parencer, ->
<op_llaveab, ->
<PR, 11>
<op_parenab, ->
<ID, 7>
<op_asignacion, ->
<ID, 5>
<op_suma, ->
<ID, 6>
<op_ptocoma, ->
<PR, 2>
<op_ptocoma, ->
<ID, 7>
<op_asignacion, ->
<ID, 5>
<op_suma, ->
<ID, 6>
<op_suma, ->
<ID, 6>
<op_parencer, ->
<op_llaveab, ->
<ID, 2>
<op_asignacion, ->
<ID, 5>
<op_suma, ->
<ID, 6>
<op_suma, ->
<ID, 7>
<op_ptocoma, ->
<op_llavecer, ->
<op_llavecer, ->
<op_llavecer, ->
<PR, 8>
<op_parenab, ->
<ID, 2>
<op_parencer, ->
<op_ptocoma, ->
<PR, 9>
<op_parenab, ->
<ID, 5>
<op_parencer, ->
<op_ptocoma, ->
<PR, 10>
<ID, 2>
<op_ptocoma, ->
<op_llavecer, ->
<PR, 4>
<PR, 5>
<ID, 8>
<op_parenab, ->
<PR, 5>
<ID, 9>
<op_parencer, ->
<op_llaveab, ->
<PR, 3>
<PR, 5>
<ID, 10>
<op_ptocoma, ->
<PR, 3>
<PR, 5>
<ID, 11>
<op_ptocoma, ->
<ID, 11>
<op_asignacion, ->
<ID, 9>
<op_suma, ->
<entero, 1>
<op_ptocoma, ->
<ID, 10>
<op_asignacion, ->
<ID, 8>
<op_parenab, ->
<ID, 11>
<op_parencer, ->
<op_ptocoma, ->
<PR, 12>
<op_parenab, ->
<ID, 10>
<op_igual, ->
<entero, 10>
<op_parencer, ->
<PR, 10>
<ID, 10>
<op_ptocoma, ->
<op_llavecer, ->

Su fichero con el árbol resultado es **./Caso5/arbol5.html**
Su parse resultante es:

Descendente 1 3 18 19 3 18 19 5 27 22 19 28 30 19 31 30 19 32 16 7 38 43 47 53 50 46 42 15 18 19 15 18 19 15 18 19 14 25 8 7 38 43 47 53 50 46 42 38 43 47 55 50 46 41 38 43 47 56 50 46 42 10 13 14 25 8 7 38 43 47 51 59 50 46 42 38 43 47 55 50 46 42 10 12 38 43 47 51 59 48 47 53 50 46 42 14 25 8 7 38 43 47 51 59 48 47 51 59 50 46 42 38 43 47 56 50 46 42 10 12 38 43 47 51 59 48 47 51 59 48 47 51 59 50 46 42 16 7 38 43 47 51 59 48 47 51 59 48 47 51 59 50 46 42 17 17 17 14 26 35 38 43 47 51 59 50 46 42 14 26 36 14 26 37 39 38 43 47 51 59 50 46 42 17 5 27 22 19 28 30 19 32 15 18 19 15 18 19 16 7 38 43 47 51 59 48 47 53 50 46 42 16 7 38 43 47 51 58 60 63 50 46 42 14 24 38 43 47 51 59 50 44 43 47 53 50 46 42 33 37 39 38 43 47 51 59 50 46 42 17 6

Funcionamiento incorrecto

```
// Probamos algunas palabras reservadas
var int a;
var int resultado;

function int multibucle (int a, int n) {
   resultado = 0;
   var int i;
   var int j;
   var int z;
   for (i = 0; True && False; i++) {
      for (j = i; True; j = i + 100){
         for (z = i + j; False; z = i + j + j){
            resultado = i + j + z;
         }
      }
   }
   print(resultado);
   prompt(i);
   return resultado;
}

function int recursividad (int p) {
   var int b;
   var int suma;
   suma = p + 1;
   b = recursividad();
   if (b == 10) return b;
}
```

El mensaje de error resultado es:

Error Semántico (Línea 26): Los parámetros de la función son incorrectos.