MEMORIA SEGUNDA ENTREGA

GRUPO 27:

- Ángel Prieto Méndez
- Adrián Calvo Moyano
- Manuel Conejo Bautista

DISEÑO DEL ANALIZADOR SINTÁCTICO **3**
Gramática 3
Tablas First/Follow 5
Tabla LL(1) 6
Gestor de Errores 6
Casos de prueba 7
Casos de prueba correctos 7
Caso 1 7
Caso 2 9
Caso 3 10
Casos de prueba incorrectos 12
Caso 1 12
Caso 2 12
Caso 3 12

DISEÑO DEL ANALIZADOR SINTÁCTICO

Gramática

Terminales = {
 if while let function return read write
 int float boolean string void false true

 id ent real cad
 + - /
 == !=
 && || !
 = --
 ( ) { } , ;
 eof
}
NoTerminales = {
 Axioma SentenciaComp Funcion Tipo TipoV Sentencia SentenciaAsig
 ValueRet Condicion FactorOp Factor OperadorAritmetico OperadorLogico
 OrAnd Valores Mas ValoresFuncion ValoresFAux Cont LetAsig ValoresFuncionLlamada
 TipoFuncion SentenciaWrite SentenciaRet LetAsigAux FactorId SentenciaAsigAux
 ValoresFuncionLlamadaAux
}
Axioma = Axioma
Producciones = {
 Axioma -> SentenciaComp Axioma
 Axioma -> Funcion Axioma
 Axioma -> eof

 SentenciaComp -> let Tipo id LetAsig ;
 SentenciaComp -> if (Condicion) Sentencia ;
 SentenciaComp -> while (Condicion) { Cont }
 SentenciaComp -> Sentencia ;

 Funcion -> function TipoFuncion id (Valores) { Cont }

 TipoFuncion -> Tipo
 TipoFuncion -> TipoV

 Tipo -> int
 Tipo -> float
 Tipo -> boolean
 Tipo -> string

 TipoV -> void

 Sentencia -> id SentenciaAsig
 Sentencia -> return SentenciaRet
 Sentencia -> read id
 Sentencia -> write SentenciaWrite

 SentenciaRet -> ValueRet
 SentenciaRet -> (ValueRet)

 SentenciaWrite -> Condicion
 SentenciaWrite -> (Condicion)

 LetAsig -> = LetAsigAux

LetAsig -> lambda

LetAsigAux -> Condicion
LetAsigAux -> (Condicion)
LetAsigAux -> id (ValoresFuncion)
SentenciaAsig -> = SentenciaAsigAux
SentenciaAsig -> (ValoresFuncion)

SentenciaAsigAux -> Condicion
SentenciaAsigAux -> (Condicion)

ValueRet -> Condicion
ValueRet -> lambda

4.1

Condicion -> FactorOp OperadorLogico OrAnd

FactorOp -> Factor OperadorAritmetico
FactorOp -> (Factor OperadorAritmetico) OperadorAritmetico

Factor -> id FactorId
Factor -> ent
Factor -> real
Factor -> cad
Factor -> false
Factor -> true
Factor -> -- id
Factor -> ! id

FactorId -> (ValoresFuncion)
FactorId -> lambda

OperadorAritmetico -> + FactorOp
OperadorAritmetico -> - FactorOp
OperadorAritmetico -> / FactorOp
OperadorAritmetico -> lambda

OperadorLogico -> == Condicion
OperadorLogico -> != Condicion
OperadorLogico -> lambda

OrAnd -> || Condicion
OrAnd -> && Condicion
OrAnd -> lambda

Valores -> Tipo id Mas
Valores -> TipoV

Mas -> , Tipo id Mas
Mas -> lambda

ValoresFuncion -> ValoresFuncionLlamada ValoresFAux
ValoresFuncion -> lambda

ValoresFAux -> , ValoresFuncionLlamada ValoresFAux
ValoresFAux -> lambda

ValoresFuncionLlamada -> id ValoresFuncionLlamadaAux
ValoresFuncionLlamada -> cad
ValoresFuncionLlamada -> ent
ValoresFuncionLlamada -> real
ValoresFuncionLlamada -> true
ValoresFuncionLlamada -> false
ValoresFuncionLlamada -> -- id
ValoresFuncionLlamada -> (Condicion)

ValoresFuncionLlamadaAux -> (ValoresFuncion)
ValoresFuncionLlamadaAux -> lambda

Cont -> SentenciaComp Cont
Cont -> lambda

}

Tablas First/Follow

Símbolos	First	Follow
Axioma	let if while id return read write function eof	$
SentenciaComp	let if while id return read write	let if while id return read write function eof }
Funcion	function	let if while id return read write function eof
TipoFuncion	int float boolean string void	id
Tipo	int float boolean string	id
TipoV	void	id
Sentencia	id return read write	;
SentenciaRet	(id ent real cad false true -- !	;
SentenciaWrite	(id ent real cad false true -- !	;
LetAsig	= λ	;
LetAsigAux	(id ent real cad false true -- !	;
SentenciaAsig	= (	;
SentenciaAsigAux	(id ent real cad false true -- !	;
ValueRet	id ent real cad false true -- ! (λ	;)
Condicion	id ent real cad false true -- ! (	) ;
FactorOp	id ent real cad false true -- ! (	== != | | &&) ;
Factor	id ent real cad false true -- !	+ - / == != | | &&) , ;
FactorId	(λ	+ - / == != | | &&) ;
OperadorAritmetico	+ - / λ	) == != | | && , ;
OperadorLogico	== != λ	| | &&) , ;
OrAnd	| | && λ	) , ;
Valores	int float boolean string void	)
Mas	, λ	)

ValoresFuncion	(id ent real cad true false -- λ	)
ValoresFAux	, λ	)
ValoresFuncionLlamada	id cad ent real true false -- (	,)
ValoresFuncionLlamada Aux	(λ	,)
Cont	let if while id return read write λ	}

Tabla LL(1)

Para comprobar que la gramática anterior es una gramática válida hemos decidido mostrarlo con la tabla LL(1) del enlace adjunto, donde se puede ver si la gramática es válida o no. En este caso, como no se encuentra ninguna celda con dos movimientos posibles para un mismo símbolo, podemos decir que nuestra gramática es válida.

https://docs.google.com/spreadsheets/d/1NMAuFcdLwH-SmccrrHT2HS9to8rCjVQ-h6fIlUFYZ4s/edit?gid=0#gid=0

Gestor de Errores

El analizador sintáctico es capaz de detectar e identificar los siguientes errores:

-Falta de un identificador: este error salta cuando no se detecta un identificador en una posición en la que no se detecta un token del tipo ID, siendo este el requerido se manda el siguiente mensaje:

error.println("Se esperaba un identificador y no un token del tipo " + token + " en la linea " + numLinea); donde token es el token faltante y numLinea la linea donde se ha dado el error

-Falta de un identificador: este error salta cuando no se detecta un identificador en una posición en la que no se detecta un token del tipo ID, siendo este el requerido se manda el siguiente mensaje:

error.println("Se esperaba un identificador y no un token del tipo " + token + " en la linea " + numLinea); donde token es el token faltante y numLinea la linea donde se ha dado el error

-Falta de ‘;’ : este error salta cuando no se detecta un ‘;’ en una posición en la que no se detecta un token del tipo PNTCOMA, siendo este el requerido se manda el siguiente mensaje:

error.println("No se ha cerrado correctamente la sentencia, se esperaba ';'" +" en la linea " + numLinea); donde token es el token faltante y numLinea la linea donde se ha dado el error

-Falta de ‘(‘ : este error salta cuando no se detecta un ‘(‘ en una posición en la que no se detecta un token del tipo APAR, siendo este el requerido se manda el siguiente mensaje:

error.println("Se esperaba '(' en la linea " + numLinea); donde numLinea la linea donde se ha dado el error

-Falta de un ‘)‘: este error salta cuando no se detecta un ‘)’ en una posición en la que no se detecta un token del tipo CPAR, siendo este el requerido se manda el siguiente mensaje:

error.println("Se esperaba ')' en la linea " + numLinea + " No se ha cerrado correctamente el parentesis"); donde token

es el token faltante y numLinea la linea donde se ha dado el error

-Falta de un ‘{‘: este error salta cuando no se detecta un ‘{‘ en una posición en la que no se detecta un token del tipo ALLAVE, siendo este el requerido se manda el siguiente mensaje:

error.println("Se esperaba '{' en la linea " + numLinea); donde numLinea la linea donde se ha dado el error

-Falta de un ‘}‘: este error salta cuando no se detecta un ‘}‘ en una posición en la que no se detecta un token del tipo CLLAVE, siendo este el requerido se manda el siguiente mensaje:

error.println("Se esperaba '}' en la linea " + numLinea + " No se ha cerrado correctamente la llave"); donde token es el token faltante y numLinea la linea donde se ha dado el error

-Caso Default: este error salta cuando el token que se recibe no puede ocupar la posición en la que está, pero no es factible indicar exactamente que se pretendía ahí, al haber múltiples funciones. Es este el mensaje que se recibe:

error.println("El token " + token + " en la linea " + numLinea + " no puede ocupar esa posicion");

Casos de prueba

Casos de prueba correctos

Caso 1

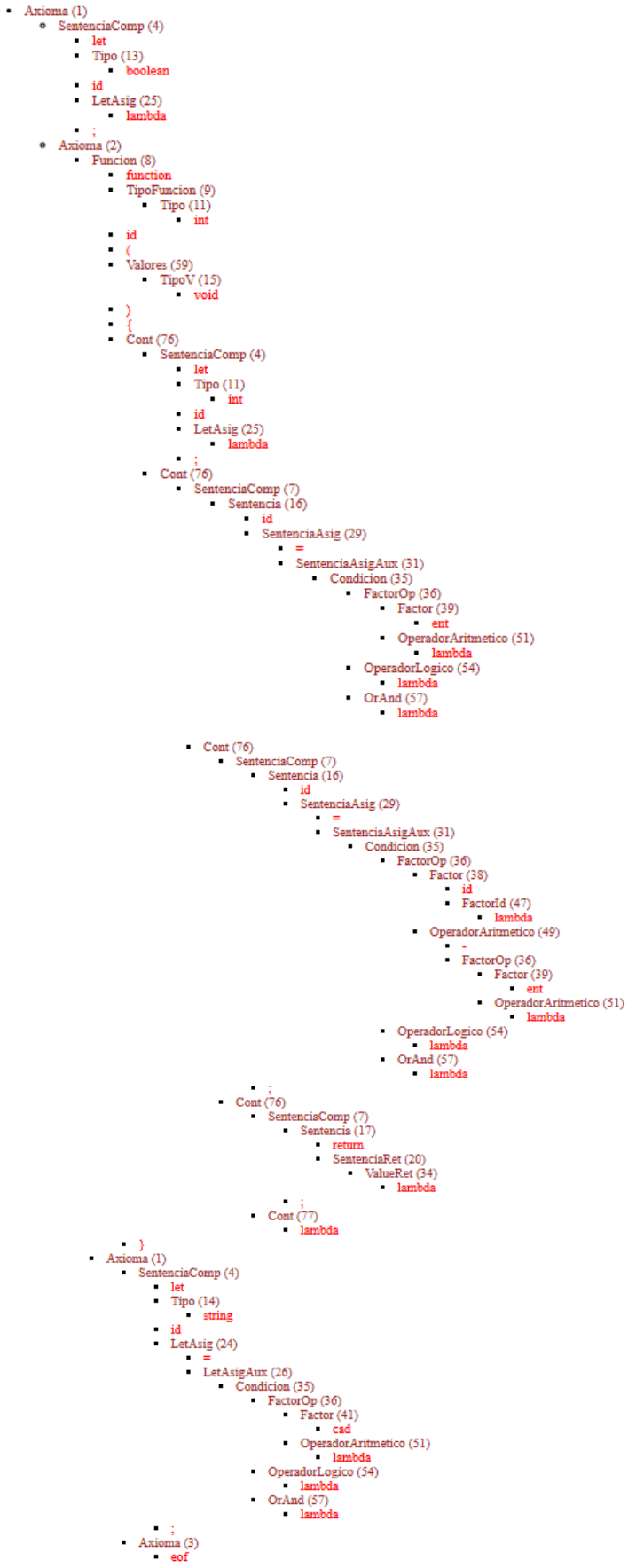
Código

```
let boolean b1;
function (void){
  let int x;
  x = 10;
  x = x - 5;
  return;
}
let String cadena = "Hola que tal";
```

Parse

DES 1 4 13 25 2 8 9 11 59 15 76 4 11 25 76 7 16 29 31 35 36 39 51 54 57 76 7 16 29 31 35 36 38 47 49 36 39 51 54 57 76 7 17 20
34 77 1 4 24 26 35 36 41 51 54 57 3

Árbol


Axioma (1)
SentenciaComp (4)
let
Tipo (13)
boolean
id
LetAsig (25)
lambda
;
Axioma (2)
Funcion (8)
function
TipoFuncion (9)
Tipo (11)
int
id
(
Valores (59)
TipoV (15)
void
)
{
Cont (76)
SentenciaComp (4)
let
Tipo (11)
int
id
LetAsig (25)
lambda
;
Cont (76)
SentenciaComp (7)
Sentencia (16)
id
SentenciaAsig (29)
=
SentenciaAsigAux (31)
Condicion (35)
FactorOp (36)
Factor (39)
ent
OperadorAritmetico (51)
lambda
OperadorLogico (54)
lambda
OrAnd (57)
lambda
Cont (76)
SentenciaComp (7)
Sentencia (16)
id
SentenciaAsig (29)
=
SentenciaAsigAux (31)
Condicion (35)
FactorOp (36)
Factor (38)
id
FactorId (47)
lambda
OperadorAritmetico (49)
-
FactorOp (36)
Factor (39)
ent
OperadorAritmetico (51)
lambda
OperadorLogico (54)
lambda
OrAnd (57)
lambda
;
Cont (76)
SentenciaComp (7)
Sentencia (17)
return
SentenciaRet (20)
ValueRet (34)
lambda
;
Cont (77)
lambda
}
Axioma (1)
SentenciaComp (4)
let
Tipo (14)
string
id
LetAsig (24)
=
LetAsigAux (26)
Condicion (35)
FactorOp (36)
Factor (41)
cad
OperadorAritmetico (51)
lambda
OperadorLogico (54)
lambda
OrAnd (57)
lambda
;
Axioma (3)
eof

Caso 2

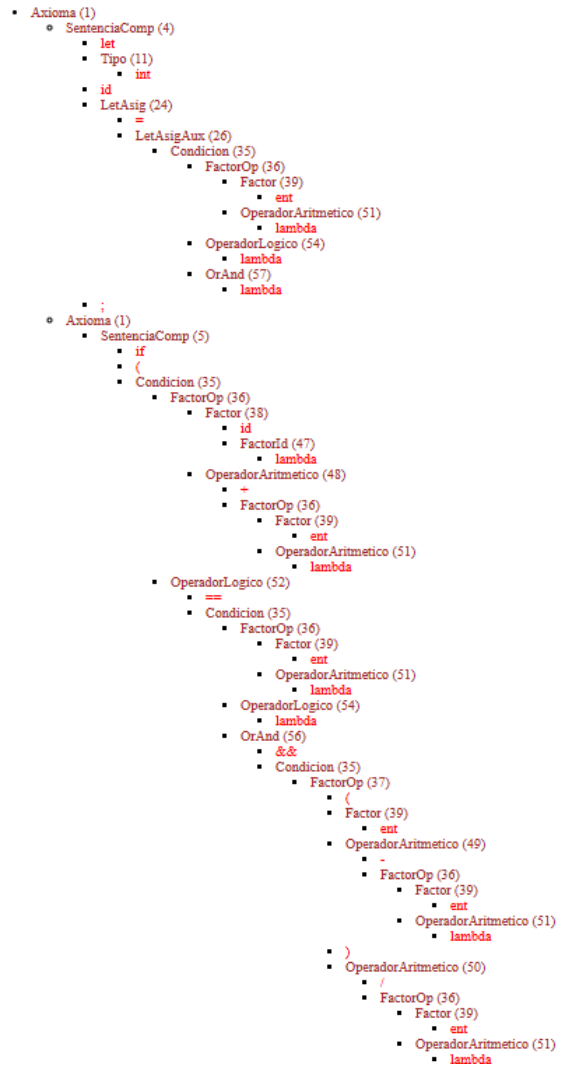
Código

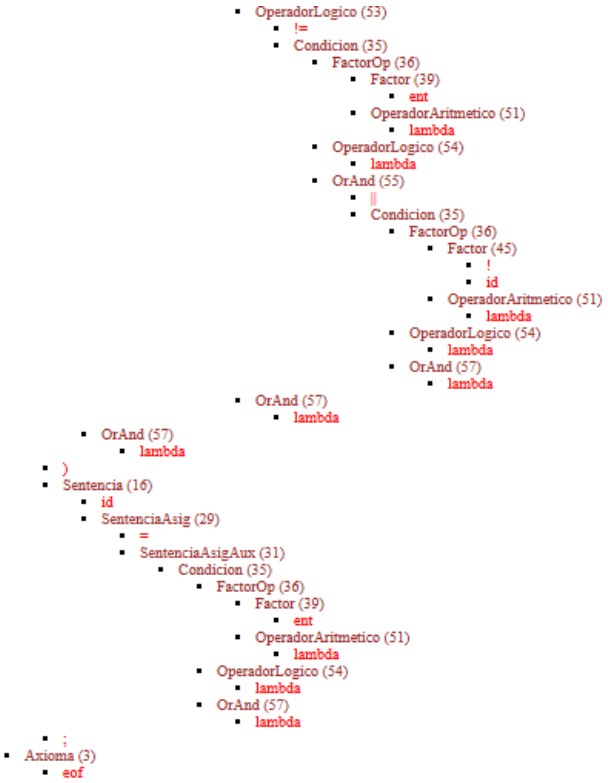
```
let int x = 5;
if ( x + 7 == 54 && (77 -76) / 5 != 10 || !x) x= 57;
```

Parse

DES 1 4 11 24 26 35 36 39 51 54 57 1 5 35 36 38 47 48 36 39 51 52 35 36 39 51 54 56 35 37 39 49 36 39 51 50 36 39 51 53 35 36 39 51 54 55 35 36 45 51 54 57 57 57 16 29 31 35 36 39 51 54 57 3

Árbol





Caso 3

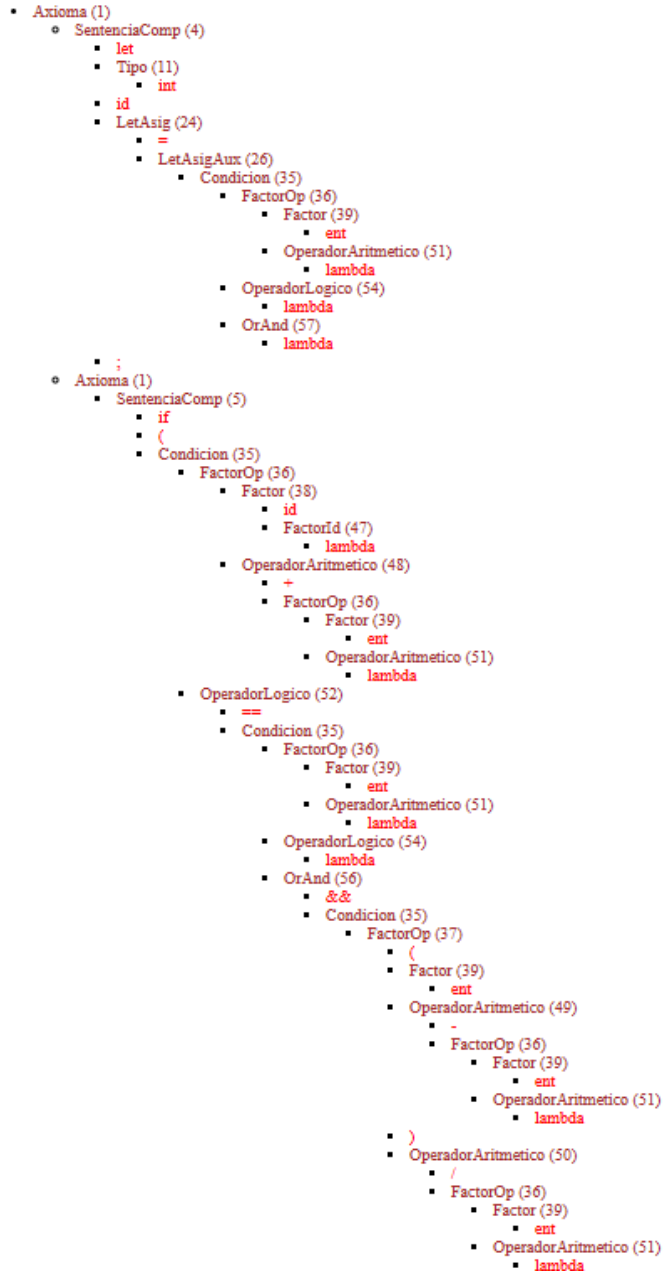
Código

```
let String hola = "Hola me llamo Manuel";
write hola ;
hola = "adios";
write (adios);
```

Parse

DES 1 4 14 24 26 35 36 41 51 54 57 1 7 19
22
35 36 38 47 51 54 57 1 7 16 29 31 35 36 41 51 54 57 1 7 19
23
35 36 38 47 51 54 57 3

Arbol


Axioma (1)
SentenciaComp (4)
let
Tipo (11)
int
id
LetAsig (24)
=
LetAsigAux (26)
Condicion (35)
FactorOp (36)
Factor (39)
ent
OperadorAritmetico (51)
lambda
OperadorLogico (54)
lambda
OrAnd (57)
lambda
;
Axioma (1)
SentenciaComp (5)
if
(
Condicion (35)
FactorOp (36)
Factor (38)
id
FactorId (47)
lambda
OperadorAritmetico (48)
+
FactorOp (36)
Factor (39)
ent
OperadorAritmetico (51)
lambda
OperadorLogico (52)
==
Condicion (35)
FactorOp (36)
Factor (39)
ent
OperadorAritmetico (51)
lambda
OperadorLogico (54)
lambda
OrAnd (56)
&&
Condicion (35)
FactorOp (37)
(
Factor (39)
ent
OperadorAritmetico (49)
-
FactorOp (36)
Factor (39)
ent
OperadorAritmetico (51)
lambda
)
OperadorAritmetico (50)
/
FactorOp (36)
Factor (39)
ent
OperadorAritmetico (51)
lambda

Casos de prueba incorrectos

Caso 1

Código

```
let String = "Hola";
```

Traza de errores

Se esperaba un identificador y no un token del tipo ASIG en la linea 1

Caso 2

Código

```
let String cadena = "Hola"
```

Traza de errores

No se ha cerrado correctamente la sentencia, se esperaba ';' en la linea 1

Caso 3

Código

```
let String cadena = ("Hola"
```

Traza de errores

Se esperaba ')' en la linea 1 No se ha cerrado correctamente el parentesis
No se ha cerrado correctamente la sentencia, se esperaba ';' en la linea 1

Índice de comentarios

4.1 La gramática de las expresiones no está correcta. No tiene en cuenta adecuadamente la precedencia y asociatividad de los operadores. Re-escribe la gramática basándote en las indicaciones dadas en clase