Práctica Procesadores de Lenguajes

Grupo 107

Pablo Castillo Martínez – y160268

Sumario

INTRODUCCIÓN

Como grupo, tenemos asignadas las siguientes opciones para la realización de la práctica para la asignatura de Procesadores de Lenguajes:

-Sentencias: **Sentencia de selección múltiple (switch-case)**

-Operadores especiales: **Post-auto-decremento (-- como sufijo)**
-Técnicas de Análisis Sintáctico: **Descendente recursivo**
-Comentarios: **Comentario de bloque (/* */)**
-Cadenas: **Con comillas dobles (" ")**

Para el resto de operadores, se han elegido:

-Aritmético: +
-Relacional: <
-Lógico: &&

Con respecto a la parte de programación, se decidió hacerlo todo con Java debido a la familiaridad con el lenguaje de programación.

Esta implementación de un procesador de lenguajes tiene varios bugs conocidos; no pretende ser perfecto en cada aspecto, sino cumplir su función principal de manera sencilla. Algunos de los cuales son:

- Al meter comentarios de la forma “/* COMENTARIO */”, debe haber uno o más espacios después de ‘/*’ y uno o varios antes de ‘*/’, de lo contrario, el procesador no reconoce la secuencia como un comentario. Funciona entre varias líneas, ese es el único bug de los comentarios.
- Se cuentan las líneas bien, excepto en el caso de que se metan comentarios, que son ignorados. Por tanto, sus lineas no se cuentan. Es decir, si las 3 primeras líneas están comentadas, estas no saldrán en el análisis posterior. Este es un bug “tolerable”, un pequeño precio a pagar en el resto de la práctica, y solo con los comentarios, por lo que es de poco importancia.
- El procesador **PARA** al encontrar el primer error. En caso de salir varios errores en el archivo de errores, hacer caso al primero. Los demás son de un bug en el sistema de errores al no parar la ejecución por completo.

TODAS las pruebas y **TODO** el desarrollado se han llevado a cabo en un sistema Linux, con la última versión de Java instalada y mediante el IDE Eclipse. Si hay algún fallo al iniciar el procesador en otro sistema operativo que no sea Linux, probar en uno que lo sea, pues puede haber ciertos problemas a la hora de leer archivos, a pesar de haber usado “File.separator” de Java, que permite poner “/” o “\” en el path del fichero dependiendo de si estamos en un SO basado en UNIX o un Windows. Han habido errores al intentar testear el mismo código en un sistema Windows. Sólo se garantiza que funciona en Linux, al menos.

El sistema por el cual se introduce el nombre del archivo a analizar es muy sencillo; basta con poner el absoluto del fichero, no el relativo. A partir de este absoluto, en el directorio en el que se encuentre el fichero se crearán el resto de ficheros para los distintos análisis que se piden en la práctica.

ANALIZADOR LÉXICO

TOKENS

<T_BOOLEAN, ->
<T_ID, lexema>
<T_ENTERO, valor>
<T_COMA,->
<T_PARENTESISABRE,->
<T_PARENTESISCIERRA,->
<T_CADENA, cadena>
<T_IGUAL,->
<T_STRING,->
<T_VAR,->
<T_SWITCH,->
<T_CASE,->
<T_BREAK,->
<T_VACIO,->
<T_SUMA,->
<T_IF,->
<T_MENOR,->
<T_ERROR,->

<T_DOSPUNTOS,-> <T_PRINT,->
<T_COMENTABRE,-> <T_INPUT,->
<T_COMENTCIERRA,-> <T_RETURN,->
<T_TRUE,-> <T_FUNC,->
<T_FALSE,-> <EOL,->
<T_LLAVEABRE,-> <EOF,->
<T_LLAVECIERRA,-> <T_AND,->
<T_PUNTOCOMA,-> <T_INT,->
<T_POSTDECREMENTO,->

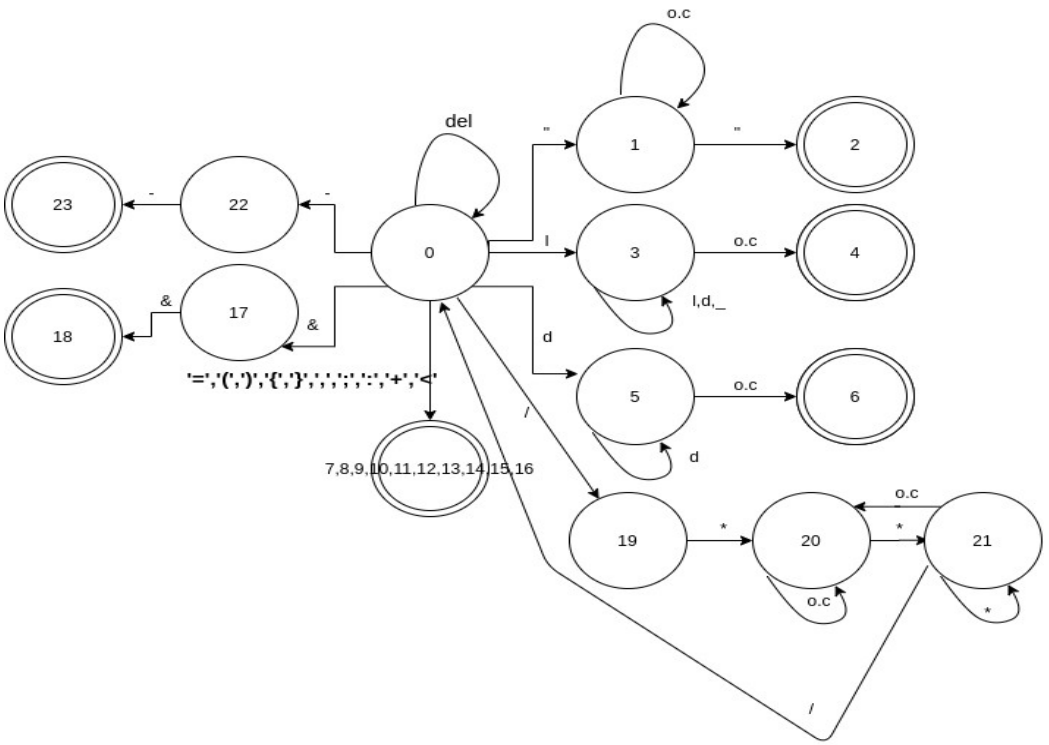
Como comentario, e puede ver a simple vista que tenemos tokens T_COMENTABRE y T_COMENTCIERRA que delimitan los comentarios y hacen que lo que esté dentro no se tenga en cuenta al pasar a las siguientes fases.

GRAMÁTICA

S → del S | “ A | l B | / C | + | - D | & E | d F | (|) | { | } | , | ; | : | = | <
A → o.c. A | “
B → l B | d B | _ B | lambda
C → * C2
C2 → * C3 | o.c C2
C3 → * C3 | o.c. C2 | / S
D → -
E → &
F → d F | lambda

Donde **l** es igual a una letra del conjunto (a-z) U (A-Z); **d** es un dígito válido del conjunto (0-9); **o.c** cualquier otro carácter distinto al esperado; por último, **del** es cualquier cantidad de espacios, tabulaciones, saltos de línea...

AUTÓMATA



ACCIONES SEMÁNTICAS

0-0: Leer
0-1: lexema=” ; Leer
1-1: lexema+=o.c ; Leer
1-2: lexema+= ”; Gtoken (T_CADENA,lexema)
0-3: lexema=l; Leer
3-3: lexema+=l,d,_; Leer
3-4: if (palabraReservada(lexema)) then Gtoken(palabraReservadaLexema,-)
else if (pos=buscarTS(lexema)) then Gtoken(T_ID,pos)
else pos=meterTS(lexema); Gtoken(T_ID,pos)
0-5: valor=d; Leer
5-5: valor=10*valor+d; Leer
5-6:if (valor>32767) then Error(“rango superior al proporcionado por el lenguaje”)
else Gtoken(T_ENTERO,valor)
0-7: Gtoken(T_IGUAL)
0-8: Gtoken(T_PARENTESISABRE)
0-9: Gtoken(T_PARENTESISCIERRA)
0-10: Gtoken(T_LLAVEABRE)
0-11: Gtoken(T_LLAVECIERRA)
0-12: Gtoken(T_COMA)
0-13: Gtoken(T_PUNTOCOMA)
0-14: Gtoken(T_DOSPUNTOS)
0-15: Gtoken(T_SUMA)
0-16: Gtoken(T_MENOR)
17-18: Gtoken(T_AND)
22-23: Gtoken(T_POSTDECREMENTO)

Cualquier otra transición no considerada en el autómata dará como resultado un error.

ANALIZADOR SINTÁCTICO

GRAMÁTICA EMPLEADA

Se decidió cambiar la gramática proporcionada originalmente por el gestor de prácticas por una descendente recursiva, sin recursividad por la izquierda. El resultado ha sido la siguiente gramática, que en la siguiente sección estudiaremos por qué es válida:

P -> B P //1
P -> F P //2
P -> eof //3

B -> var T id B2 ; //4
B -> if abreParentesis E cierraParentesis S //5
B -> S //6
B -> switch abreParentesis E cierraParentesis abreCorchete CASE cierraCorchete //7

B2 -> igual E //8
B2 -> lambda //9

T -> int //10
T -> string //11
T -> boolean //12

C -> B C //13
C -> lambda //14

S -> id S2 ; //15
S -> return X ; //16
S -> print abreParentesis E cierraParentesis ; //17
S -> input abreParentesis id cierraParentesis ; //18
S -> break ; //19

S2 -> postDecre //20
S2 -> igual E //21
S2 -> abreParentesis L cierraParentesis //22

X -> E //23
X -> lambda //24

F -> function H id abreParentesis A cierraParentesis abreCorchete C
cierraCorchete //25

A -> T id K //26
A -> lambda //27

K -> coma T id K //28
K -> lambda //29

H -> T //30
H -> lambda //31

L -> E Q //32
L -> lambda //33

Q -> coma E Q //34
Q -> lambda //35

E -> Y E2 //36

E2 -> and Y E2 //37
E2 -> lambda //38

Y -> D Y2 //39

Y2 -> menor D //40
Y2 -> lambda //41

D -> V D2 //42

D2 -> mas V D2 //43
D2 -> lambda //44

V -> id V2 //45
V -> abreParentesis E cierraParentesis //46
V -> cte_entero //47
V -> cte_cadena //48
V -> true //49
V -> false //50

V2 -> postDecre //51
V2 -> abreParentesis L cierraParentesis //52
V2 -> lambda //53

CASE -> case E : C CASE //54
CASE -> lambda //55

DEMOSTRACIÓN DE VALIDEZ

Esta gramática es LL(1) y válida, estudiemos las tablas (En verde, los no terminales que pueden acabar en lambda y para los que hay que tener en cuenta el follow):

No Terminal	First	Follow
P	eof var if switch function id return print input break	
B	var if switch id return print input break	case cierraCorchete eof function var if switch id return print input break
B2	igual	;
T	int string boolean	id
C	var if switch id return print input break	case cierraCorchete
S	id return print input break	case cierraCorchete eof function var if switch id return print input break
S2	postDecre igual abreParentesis	;
X	id abreParentesis cte_entero cte_cadena true false	;
F	function	eof var if switch function

		id return print input break
A	Int string boolean	cierraParentesis
K	coma	cierraParentesis
H	Int string boolean	id
L	id abreParentesis cte_entero cte_cadena true false	cierraParentesis
Q	coma	cierraParentesis
E	id abreParentesis cte_entero cte_cadena true false	coma cierraParentesis : ;
E2	and	coma cierraParentesis : ;
Y	id abreParentesis cte_entero cte_cadena true false	and coma cierraParentesis : ;
Y2	menor	and coma cierraParentesis : ;
D	id abreParentesis cte_entero cte_cadena true false	menor and coma cierraParentesis : ;
D2	mas	menor and coma cierraParentesis : ;
V	id abreParentesis cte_entero cte_cadena true false	mas menor and coma cierraParentesis : ;
V2	postDecre abreParentesis	mas menor and coma cierraParentesis : ;
CASE	case	cierraCorchete

Si estudiamos en detalle el movimiento de cada no terminal en la tabla LL1 con cada no terminal disponible a su alcance, se comprueba que podemos alcanzar todos los no terminales, se comprueba que para no terminal solo hay un movimiento por cada no terminal a su alcance, y nunca pueden ser dos a la vez. Además, P es el inicial y el terminal.

PROCEDURES

A continuación, una lista de todos los procedures que se han tenido que realizar para el descendente recursivo.

Procedure (P) :
if (next_token ϵ first(B P)) then

```
        B()
        P()
else if (next_token ϵ first(F P)) then
        F()
        P()
else if (next_token == eof) then
        equiparar(eof)
else error;
```

Procedure (B) :

```
if (next_token ϵ first(var T id B2 ;)) then
        equiparar(var)
        T()
        equiparar(id)
        B2()
        equiparar(;)
else if (next_token ϵ first(if abreParentesis  E  cierraParentesis S)) then
        equiparar(if)
        equiparar(abreParentesis)
        E()
        equiparar(cierraParentesis)
        S()
else if (next_token ϵ first(S)) then
        S()
else if (next_token ϵ first(switch abreParentesis  E  cierraParentesis
abreCorchete CASE cierraCorchete )) then
        equiparar(switch)
        equiparar(abreParentesis)
        E()
        equiparar(cierraParentesis)
        equiparar(abreCorchete)
        CASE()
        equiparar(cierraCorchete)
        S()

else error;
```

Procedure (B2) :

```
if (next_token ϵ first(igual E)) then
        equiparar(igual)
        E()
else if (next_token ϵ follow(B2)) then {}
else error;
```

Procedure (T) :

```
if (next_token ϵ first(int)) then
        equiparar(int)
if (next_token ϵ first(string)) then
        equiparar(string)
if (next_token ϵ first(boolean)) then
```

```
        equiparar(boolean)
else error;
```

Procedure (C) :

```
if (next_token ϵ first(B C)) then
        B()
        C()
if (next_token ϵ follow(C)) then {}
else error;
```

Procedure (S) :

```
if (next_token ϵ first(id S2 ; )) then
        equiparar(id)
        S2()
        equiparar(;)
if (next_token ϵ first(return X ;)) then
        equiparar(return)
        X()
        equiparar(;)
if (next_token ϵ first(print abreParentesis  E  cierraParentesis ; )) then
        equiparar(print)
        equiparar(abreParentesis)
        E()
        equiparar(cierraParentesis)
        equiparar(;)
if (next_token ϵ first(input abreParentesis  id  cierraParentesis ;)) then
        equiparar(input)
        equiparar(abreParentesis)
        equiparar(id)
        equiparar(cierraParentesis)
        equiparar(;)
if (next_token ϵ first(break ; )) then
        equiparar(break)
        equiparar(;)
else error;
```

Procedure (S2) :

```
if (next_token ϵ first(postdecre)) then
        equiparar(postdecre)
if (next_token ϵ first(igual E)) then
        equiparar(igual)
        E()
if (next_token ϵ first( abreParentesis L cierraParentesis  )) then
        equiparar(abreParentesis)
        L()
        equiparar(cierraParentesis)
else error;
```

Procedure (X) :

```
if (next_token ϵ first(E)) then
```

E()
else if (next_token ϵ follow(X)) then {}
else error;

Procedure (F) :
if (next_token ϵ first(function H id abreParentesis A cierraParentesis abreCorchete C cierraCorchete)) then
equiparar(function)
H()
equiparar(id)
equiparar(abreParentesis)
A()
equiparar(cierraParentesis)
equiparar(abreCorchete)
C()
equiparar(cierraCorchete)
else error;

Procedure (A) :
if (next_token ϵ first(T id K)) then
T()
equiparar(id)
K()
else if (next_token ϵ follow(A)) then {}
else error;

Procedure (K) :
if (next_token ϵ first(coma T id K)) then
equiparar(coma)
T()
equiparar(id)
K()
else if (next_token ϵ follow(K)) then {}
else error;

Procedure (H) :
if (next_token ϵ first(T)) then
T()
else if (next_token ϵ follow(T)) then {}
else error;

Procedure (L) :
if (next_token ϵ first(E Q)) then
E()
Q()
else if (next_token ϵ follow(L)) then {}
else error;

Procedure (Q) :
if (next_token ϵ first(coma E Q)) then

```
        equiparar(coma)
        E()
        Q()
else if (next_token ϵ follow(Q)) then {}
else error;
```

Procedure (E) :

```
if (next_token ϵ first(Y E2)) then
        Y()
        E2()
else error;
```

Procedure (E2) :

```
if (next_token ϵ first(and Y E2 )) then
        equiparar(and)
        Y()
        E2()
else if (next_token ϵ follow(E2)) then {}
else error;
```

Procedure (Y) :

```
if (next_token ϵ first(D Y)) then
        D()
        Y()
else error;
```

Procedure (Y2) :

```
if (next_token ϵ first(menor D )) then
        equiparar(menor)
        D()
else if (next_token ϵ follow(Y2)) then {}
else error;
```

Procedure (D) :

```
if (next_token ϵ first(V D2 )) then
        V()
        D2()
else error;
```

Procedure (D2) :

```
if (next_token ϵ first(mas V D2 )) then
        equiparar(mas)
        V()
        D2()
else if (next_token ϵ follow(D2)) then {}
else error;
```

Procedure (V) :

```
if (next_token ϵ first(id V2 )) then
        equiparar(id)
        V2()
```

```
else if (next_token ϵ first(abreParentesis  E  cierraParentesis)) then
        equiparar(abreParentesis)
        E()
        equiparar(cierraParentesis)
else if (next_token ϵ first(cte_entero)) then
        equiparar(cte_entero)
else if (next_token ϵ first(cte_cadena)) then
        equiparar(cte_cadena)
else if (next_token ϵ first(true)) then
        equiparar(true)
else if (next_token ϵ first(false)) then
        equiparar(false)
else error;
```

Procedure (V2) :

```
if (next_token ϵ first(postdecre )) then
        equiparar(postdecre)
else if (next_token ϵ firt(abreParentesis  L  cierraParentesis)) then
        equiparar(abreParentesis)
        L()
        equiparar(cierraParentesis)
else if (next_token ϵ follow(V2)) then {}
else error;
```

Procedure (CASE) :

```
if (next_token ϵ first(case E : C CASE)) then
        equiparar(case)
        E()
        equiparar(:)
        C()
        CASE()
else if (next_token ϵ follow(cierraCorchete)) then {}
else error;
```

ANALIZADOR SEMÁNTICO

P' -> *{CrearTSG()}* P *{CerrarTablas()}*

P -> B P
P -> F P
P -> eof

B -> var T id *{if (idDeclarado(id)), then error, else meterTS(id.entrada, T.tipo)}*
B2 *{if (!B2.tipo==T.tipo && !B2.tipo==vacio) then error}* ;
B -> if abreParentesis E *{if (E.tipo != logico) then error}* cierraParentesis S

B -> switch abreParentesis E *{if (E.tipo!=entero) then error}* cierraParentesis abreCorchete CASE cierraCorchete
B -> S

B2 -> igual E *{B2.tipo=E.tipo}*
B2 -> lambda *{B2.tipo=vacio}*

T -> int *{T.tipo=entero}*
T -> string *{T.tipo=cadena}*
T -> logico *{T.tipo=logico}*

C -> B C
C -> lambda

S -> id *{S2.entrada=id.entrada}{if (id.entrada==noExiste), then meterTS(id.entrada, entero)}* S2 ;*{if (S2.tipo==entero && id.entrada==entero), then S.tipo=entero; else if (S2.tipo=vacio), then S.tipo=tipo_ok; else if (id.entrada==funcion && S2.tipo==tipo_ok) then S.tipo=tipoFuncion(id); else error}*
S -> *{if !(funcion) then: error}* return X ; *{if (funcion.tipo!=X.tipo) then error; else devuelto=true;funcion=false}*
S -> print abreParentesis E cierraParentesis ;
S -> input abreParentesis id cierraParentesis ; *{if (id.tipo==bool || IDdeclarado) then error}*
S -> break; *{if (zonaBreak==false) then error;}*

S2 -> postDecre *{S2.tipo=entero}*
S2 -> igual E *{S2.tipo=E.tipo}*
S2 ->*{if !(tipoID(S2.entrada!=funcion)) then error}* abreParentesis L cierraParentesis *{if !(compararParametros(S2.lista,L.lista)) then error, else S2.tipo=tipo_ok}*

X -> E *{X.tipo=E.tipo}*
X -> lambda *{X.tipo=vacio}*

F -> *{if (funcion) then error}* function *{funcion=true, TSL=crearTS(),TablaActual=TSL}* H *{F.tipo=H.tipo}* id abreParentesis A cierraParentesis abreCorchete C cierraCorchete *{meterTS(id.entrada,funcion),meterParametros(A.lista),metertipos(A.lista),meterdesplazamientos(A.lista)}{TablaActual=TSG}*

A -> T id K *{Flista.meter(id.entrada,T.tipo),Fparametros=1}*
A -> lambda *{Flista=null, Fparametros=0}*

K -> coma T id K *{Flista.meter(id.entrada,T.tipo),Fparametros++}*
K -> lambda

H -> T *{H.tipo=T.tipo}*
H -> lambda {H.tipo=vacio}

L -> E Q *{LLista.meter(E.tipo)}*
L -> lambda *{L.lista=vacio}*

Q -> coma E Q *{LLista.meter(E.tipo)}*
Q -> lambda

E -> Y E2 *{if (E2.tipo==tipo_ok || E2.tipo==Y.tipo) then E.tipo=Y.tipo, else error}*

E2 -> and Y E2 *{if (Y.tipo==logico && (E2.tipo==tipo_ok || E2.tipo==logico)) then E2.tipo=logico, else error}*
E2 -> lambda *{E2.tipo=tipo_ok}*

Y -> D Y2 *{if (D.tipo==entero && Y2.tipo==logico) then Y.tipo=logico, else if (Y2.tipo==tipo_ok) then Y.tipo=D.tipo, else error}*

Y2 -> menor D *{if (D.tipo==entero) then Y2.tipo=logico, else error}*
Y2 -> lambda *{Y2.tipo=tipo_ok}*

D -> V D2 *{if (V.tipo==D2.tipo || D2.tipo==tipo_ok)) then D.tipo=V.tipo, else error}*

D2 -> mas V D2 *{if (V.tipo==entero && (D2.tipo==entero || D2.tipo==tipo_ok)) then D2.tipo=V.tipo, else error}*
D2 -> lambda *{D2.tipo=tipo_ok}*

V -> id *{V2.entrada=id.entrada}{if (id.entrada==noExiste), then meterTS(id.entrada, entero)}* V2 *{if (V2.tipo==entero && id.entrada==entero), then V.tipo=entero; else if (V2.tipo=vacio), then V.tipo=tipo_ok; else if (id.entrada==funcion && V2.tipo==tipo_ok) then V.tipo=tipoFuncion(id); else error}*
V -> abreParentesis E cierraParentesis *{V.tipo=E.tipo}*
V -> cte_entero *{V.tipo=entero}*
V -> cte_cadena *{V.tipo=cadena}*
V -> true *{V.tipo=logico}*
V -> false *{V.tipo=logico}*

V2 -> postDecre *{V2.tipo=entero}*
V2 -> *{if !(tipoID(V2.entrada!=funcion)) then error}* abreParentesis L cierraParentesis *{if !(compararParametros(V2.lista,L.lista)) then error, else V2.tipo=tipo_ok}*
V2 -> lambda *{V2.tipo=tipo_ok}*

CASE -> case E *{if (E.tipo!=entero) then error}* : C CASE
CASE -> lambda

ERRORES

El procesador es capaz de reconocer errores en cada una de las tres partes. En el léxico destaca sobrepasar el número máximo para un entero o no reconocer un token introducido porque no existe en la implementación (por ejemplo, un ‘>’ en lugar de un ‘<’).
Los errores del sintáctico son capaces de reconocer cuando falta un token, como un punto y coma, dos puntos, id… En el lugar que corresponda.
El semántico se encarga de informar cuando un tipo no es igual a otro, cuando los tipos devueltos son distintos al expresado al principio de una función, parámetros de llamada incorrectos… Y todo esto con un número mínimo de errores en la implementación.

TABLA DE SÍMBOLOS

Para la implementación de la tabla de símbolos, se ha decidido crear dos tablas de símbolos, una de ellas la global, y otra la local. Se intercambia una por otra cuando entramos y salimos de una función (De global a local al entrar a función y viceversa al salir). Las tablas son idénticas entre ellas, y tienen los siguientes campos:

Posición	Lexema	Tipo	Desplazamiento	Parámetros	Tipo Devuelto

La tabla global no devuelve ningún tipo, solo devuelven las locales que son funciones. Además, no se pueden anidar más funciones dentro de otras a menos que sea la misma que la local y se quiera llamar de manera recursiva. Esto se cumple.

COMENTARIOS

Finalmente, para terminar, quería dedicar un apartado a lo aprendido y a otros comentarios.

Con un proyecto de esta envergadura he aprendido a seguir un desarrollo constante que espero que siga presente en el resto de mi vida desarrollando software. Es cierto que podría ser más limpio y legible, pero también es cierto que éste es un código en producción particular y no un gran proyecto en una empresa, por lo que es normal desarrollar de manera más rápida y sin demasiado testing.

Dar las gracias también a los profesores de la asignatura por su tiempo y dedicación a explicar el temario. Sinceramente, al principio no creía estar a la altura de un proyecto así de forma individual, pero gracias al esfuerzo continuado y las lecciones aprendidas ha sido más sencillo de lo que pudiera

parecer en un primer momento. Esta ha sido una de las prácticas más entretenidas, interesantes y amenas que he tenido en los 4 años de grado.

Finalmente, agradecer al libro de la bibliografía "Compilers. Principles, Techniques and Tools", de los autores Aho, A. V.; Lam, M. S.; Sethi, R.; Ullman, J. D. , que en este tiempo de cuarentena ha sido de gran ayuda ante dudas de diseño en el procesador de lenguajes y errores presentes.

ANEXO

CASOS CORRECTOS

CASO PRIMERO

```
/*
/home/pablo/eclipse-workspace/PDL/docs/CASOS_OK/caso1/fuente.txt
*/

var int n=50;
function int recursivo(int min, int max){
	devolver=recursivo(min+5,max);
	return devolver;
}

var int g1=10;
var int resultado=recursivo(g1+4,50);
```

LISTADO DE TOKENS

```
<EOL, -1>
<T_VAR, -1>
<T_INT, -1>
<T_ID, n>
<T_IGUAL, -1>
<T_ENTERO, 50>
<T_PUNTOCOMA, -1>
<EOL, -1>
<T_FUNC, -1>
<T_INT, -1>
<T_ID, recursivo>
<T_PARENTESISABRE, -1>
<T_INT, -1>
<T_ID, min>
<T_COMA, -1>
<T_INT, -1>
<T_ID, max>
<T_PARENTESISCIERRA, -1>
<T_LLAVEABRE, -1>
<EOL, -1>
<T_ID, devolver>
<T_IGUAL, -1>
<T_ID, recursivo>
<T_PARENTESISABRE, -1>
<T_ID, min>
<T_SUMA, -1>
<T_ENTERO, 5>
<T_COMA, -1>
<T_ID, max>
<T_PARENTESISCIERRA, -1>
```

<T_PUNTOCOMA, -1>
<EOL, -1>
<T_RETURN, -1>
<T_ID, devolver>
<T_PUNTOCOMA, -1>
<EOL, -1>
<T_LLAVECIERRA, -1>
<EOL, -1>
<EOL, -1>
<T_VAR, -1>
<T_INT, -1>
<T_ID, g1>
<T_IGUAL, -1>
<T_ENTERO, 10>
<T_PUNTOCOMA, -1>
<EOL, -1>
<T_VAR, -1>
<T_INT, -1>
<T_ID, resultado>
<T_IGUAL, -1>
<T_ID, recursivo>
<T_PARENTESISABRE, -1>
<T_ID, g1>
<T_SUMA, -1>
<T_ENTERO, 4>
<T_COMA, -1>
<T_ENTERO, 50>
<T_PARENTESISCIERRA, -1>
<T_PUNTOCOMA, -1>
<EOF, -1>

TABLA DE SÍMBOLOS

CONTENIDO DE LA TABLA # 0 (de funcion GLOBAL) :
* LEXEMA: 'resultado'
ATRIBUTOS:
+ tipo: 'T_INT'
+ despl : 6

* LEXEMA: 'g1'
ATRIBUTOS:
+ tipo: 'T_INT'
+ despl : 4

* LEXEMA: 'devolver'
ATRIBUTOS:
+ tipo: 'T_INT'
+ despl : 2

* LEXEMA: 'recursivo'
ATRIBUTOS:
+ tipo: 'T_FUNC'
 +NumParam: 2
  +TipoParam1: T_INT
  +ModoParam1: 1 (es por valor)
   +TipoParam2: T_INT
   +ModoParam2: 1 (es por valor)
    +TipoRetorno: T_INT
     +EtiqFuncion: Etrecursivo

* LEXEMA: 'n'
ATRIBUTOS:
+ tipo: 'T_INT'
+ despl : 0

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

CONTENIDO DE LA TABLA # 1 (de funcion recursivo) :
* LEXEMA: 'max'
ATRIBUTOS:
+ tipo: 'T_INT'
+ despl : 2

* LEXEMA: 'min'
ATRIBUTOS:
+ tipo: 'T_INT'
+ despl : 0

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

ÁRBOL RESULTANTE

(Árbol del caso primero incluido en la entrega para su mejor visualización, ya que aquí no se ve de la mejor manera posible. Leer de izquierda a derecha)

Árbol resultado de:

Gramática: C:\Users\Pablo\Downloads\ ResultadoGramatica.txt

Parse: C:\Users\Pablo\Downloads\ ResultadoSintactico.txt

- P (1)
  - B (4)
    - var
  - T (10)
    - int
- id
- B2 (8)
- igual
- E (36)
  - Y (39)
    - D (42)
      - V (47)
        - cte_entero
  - D2 (44)
    - lambda
  - Y2 (41)
    - lambda
  - E2 (38)
    - lambda
- ;
- P (2)
  - F (25)
    - function
  - H (30)
    - T (10)
      - int
- id
- abreParentesis
- A (26)
  - T (10)
    - int
- id
- K (28)
  - coma
  - T (10)
    - int
- id
- K (29)
  - lambda
- cierraParentesis
- abreCorchete
- C (13)
  - B (6)
    - S (15)
  - id
  - S2 (21)
  - igual
  - E (36)
    - Y (39)
      - D (42)
        - V (45)
          - id
  - V2 (52)
    - abreParentesis
    - L (32)
      - E (36)
        - Y (39)
          - D (42)
            - V (45)
              - id
        - V2 (53)
          - lambda
        - D2 (43)
          - mas
          - V (47)
            - cte_entero
          - D2 (44)
            - lambda
          - Y2 (41)
            - lambda
          - E2 (38)
            - lambda
          - Q (34)
            - coma
            - E (36)
              - Y (39)
                - D (42)
                  - V (45)
                    - id
              - V2 (53)
                - lambda
              - D2 (44)
                - lambda
              - Y2 (41)
                - lambda
              - E2 (38)
                - lambda
              - Q (35)
                - lambda
    - cierraParentesis
    - D2 (44)
      - lambda
    - Y2 (41)
      - lambda
    - E2 (38)
      - lambda
- ;
- C (13)
  - B (6)
    - S (16)
  - return
  - X (23)
    - E (36)
      - Y (39)
        - D (42)
          - V (45)
            - id
      - V2 (53)
        - lambda
      - D2 (44)
        - lambda
      - Y2 (41)
        - lambda
      - E2 (38)
        - lambda
- ;
- C (14)
  - lambda
- cierraCorchete

- P (1)
  - B (4)
  - var
- T (10)
- int

- id
- B2 (8)
- igual
- E (36)
  - Y (39)
  - D (42)
  - V (47)
  - cte_entero
- D2 (44)
- lambda
- Y2 (41)
- lambda
- E2 (38)
- lambda

- ;
- P (1)
  - B (4)
  - var
- T (10)
- int

- id
- B2 (8)
- igual
- E (36)
  - Y (39)
  - D (42)
  - V (45)
  - id
- V2 (52)
- abreParentesis
- L (32)
  - E (36)
  - Y (39)
  - D (42)
  - V (45)

- id
- V2 (53)
- lambda
- D2 (43)
- mas
- V (47)
- cte_entero
- D2 (44)
- lambda
- Y2 (41)
- lambda
- E2 (38)
- lambda
- Q (34)
- coma
- E (36)
  - Y (39)
  - D (42)
  - V (47)
  - cte_entero
- D2 (44)
- lambda
- Y2 (41)
- lambda
- E2 (38)
- lambda
- Q (35)
- lambda

- cierraParentesis
- D2 (44)
- lambda
- Y2 (41)
- lambda
- E2 (38)
- lambda

- ;
- P (3)
- eof

CASO SEGUNDO

```
function int test(int n,int m){
        if (n < m && n+2< m+1 && true) return n;
}

var int devolver=test(5,6);
print(devolver);
```

CASO TERCERO

```
var boolean flag=true;
var int entrada=5;
input(entrada);
if (flag) print(entrada);
```

```
entrada=0;
```

CASO CUARTO

```
function int testSwitch(int a, int b){
	switch(a){
	case 1: a=a+5; break;
	case 2: a=10;
	case 3: a=b;
	}
	return a;
}

var int devuelto=testSwitch(20,30);
```

CASO QUINTO

```
var boolean test=false;

function string rama(int n, int m){
	if (n < m && test) return "false";
	return "true";
}

var int entrada1;
var int entrada2;
var string nombre;

input(entrada1);
input(entrada2);
input(nombre);

print(entrada1);
print(entrada2);
print(nombre);

rama(entrada1,entrada2);
```

CASOS INCORRECTOS

CASO PRIMERO

```
function int test(int n,int m){
	if (n < m && n+2< m+1 && true) return true;
}
```

```
var int devolver=test(5,6);
print(devolver);
```

ERRORES

A partir de aquí se puede comprobar el bug del que se hablaba al principio. A pesar de detectar correctamente el primer error y la linea, también reconoce el siguiente “error fantasma” al no parar completamente el programa y a pesar de no existir. Simplemente, prestar atención al primero que salte en la ejecución.

SEMANTICO: ERROR EN LINEA 2 , TIPO DE FUNCION Y TIPO DEVUELTO SON DISTINTOS
SINTACTICO: ERROR EN LINEA 2 , SE ESPERABA UN CIERRE DE LLAVE

CASO SEGUNDO

```
var int t=5;

if (t && true)print ("STRING");
```

ERRORES

SEMANTICO: ERROR, EN LINEA 3 , LA EXPRESION DEL IF NO ES DE TIPO BOOLEANO

CASO TERCERO

```
var string n="string";
switch(n){
        case 1:
        case 2:
                return 5;
                break;
}
```

ERRORES

SEMANTICO: ERROR EN LINEA 2 , LA EXPRESION DEL SWITCH NO ES UN ENTERO

CASO CUARTO

```
var int n=5;
switch(n){
```

```
	case 1:
	case 2:
		return 5;
		break;
}
```

SEMANTICO: ERROR EN LINEA 5 , USADO RETURN SIN ESTAR DENTRO DE UNA FUNCION
SINTACTICO: ERROR EN LINEA 5 , CASE INCORRECTO
SINTACTICO: ERROR EN LINEA 5 , SE ESPERABA UN DOS PUNTOS O LLAVE CIERRA
SINTACTICO: ERROR EN LINEA 5 , SE ESPERABA CERRAR LLAVE EN EL SWITCH

CASO QUINTO

```
function int test(int n){
	return n;
}

function string test(int n){
	return "n";
}
```

ERRORES

SEMANTICO: ERROR EN LINEA 4 , FUNCION O ID test YA DECLARADO EN ZONA GLOBAL