2022

PRÁCTICA PROCESADORES DEL LENGUAJE

GRUPO 13

INTEGRANTES:

- Mario López Estaire
- Andrés Bravo Francos
- Grettell Umpierrez Sardiñas

Sentencias: Sentencia de Selección Múltiple (switch-case)
Técnicas de Análisis Sintáctico: Descendente con tablas
Operadores especiales: Asignación con multiplicación (*=)
Comentarios: Comentario de bloque (/**/)
Cadenas: Con comillas dobles (“ ”)

- Comentarios → /*...*/
- Constantes →
  - Enteras → d^+ → 16b(1 palabra) → máx (32767)
    - Cadena …. → “c*”→ máx. 64 caracteres
  - Lógicas → pal_reservada(true, false)

- Operadores →
  - Relación → ==
  - Aritméticos → *
  - Lógico → &&
  - Asignación → =
  - Asignación con multiplicación → *=
- Identificadores → (l + _) + (d + _ + l) * → l (a-z, A-Z) → d(0-9)
- Declaraciones → pal_reservada(let)
- Datos→
  - Entero → pal_reservada (int)
  - Lógico→ pal_reservada(boolean)
  - Cadena → pal_reservada (string)

- Entrada/Salida → pal_reservada (print, input).
- Sentencias → pal_reservada(return, function, if, switch, case, default, break, eof)
- Otros → ( ) { } ; : ,

ANALIZADOR LÉXICO

TOKENS

Generales
<abrirParentesis, >
<cerrarParentesis, >
<abrirCorchete, >
<cerrarCorchete, >
<coma, >
<ptoComa, >
<dosPuntos, >
<eof, >

Operadores
<opAritmetico,1 >
<opAritmetico, 2 >
<opRelacional,1 >
<opLogico,1 >
<asigMultiplicacion, - >
<asignacion, - >

Valores de Datos

<cteEntera, valor>
<cadena, “lexema”>
< true, >
< false, >

Declaraciones
<id, posTs>
<let, >

Tipos de Datos *(Palabras reservadas)*
<int, >
<boolean, >
<string, >

Funciones *(Palabras reservadas)*
<function, >
<return, >

Sentencias *(Palabras reservadas)*
<if, >
<switch, >
<case, >
<default, >
<break, >

E/S *(Palabras reservadas)*
<input, >
<print, >

Op_aritmético
1: *
2: +

Op_relacional
1: ==

Op_lógico
1: &&

Cada Token generado será volcado en un fichero “tokens.txt”.

GRAMÁTICA

$S \rightarrow delS \mid lA \mid \_A \mid dB \mid = C \mid “D \mid *E \mid \&F \mid /G \mid (\mid) \mid \{ \mid \} \mid ; \mid , \mid + \mid :$
$A \rightarrow lA \mid dA \mid \_A \mid o.c$
$B \rightarrow dB \mid o.c$
$C \rightarrow = \mid o.c$
$D \rightarrow c3D \mid “$
$E \rightarrow = \mid o.c$
$F \rightarrow \&$
$G \rightarrow *H$
$H \rightarrow c1H \mid *H’$
$H’ \rightarrow c2H \mid *H’ \mid /S$

LEYENDA:

l: {a-z,A_Z}
d: {0 - 9}
$o.c$: cualquier carácter no contemplado en el nodo
$c1$: todos los caracteres – {*}
$c2$: todos los caracteres – {*,/}
$c3$: todos los caracteres – {“}
del: tab, espacio, eol

AUTÓMATA

ACCIONES SEMÁNTICAS

1. **S→S**: leer();

2. **S→A**: lexema = c; leer();

3. **A→A**: lexema = lexema + c; leer();

4. **A→B**:
   if(lexema == palabraReservada) then generarToken(lexema, -);
   else{
   pos=BuscarLugarTS(lexema);

```
if(pos ¡= null) then generarToken(lexema, pos);
else{
        pos=insertarIdTS(lexema);
        generarToken(lexema,pos);
}
```

5. **S→C**: valor = char_int(d); leer();

6. **C→C**: valor = valor * 10 + char_int(d); leer();

7. **C→D**: if (valor > 32767) then error (60);
   else generarToken(cte_entera, valor);

8. **S→E**: leer();
9. **E→F**: leer();
10. **F→F**: leer();
11. **F→G**: leer();
12. **G→F**: leer();
13. **G→G**: leer();
14. **G→S**: leer();

15. **S→K**: lexema = “ ”; leer();

16. **K→K**: lexema = lexema + c; leer();

17. **K→L**: if(lexema.length > 64) then error (61);
    else generarToken(cadena, lexema);

18. **S→H**: leer();

19. **H→I**: generarToken(opRelacional, 1);

20. **H→J**: generarToken(asignación, -);

21. **S→M**: leer();

22. **M→N**: generarToken(asigMultiplicacion, -);

23. **M→O**: generarToken(opAritmetico, 1);

24. **S→P**: leer();
25. **P→Q**: generarToken(opLogico, 1);

26. **S→R**: generarToken(abrirParentesis, -);
27. **S→T**: generarToken(cerrarParentesis, -);

28. **S→U**: generarToken(abrirCorchete, -);
29. **S→V**: generarToken(cerrarCorchete, -);
30. **S→W**: generarToken(ptoComa, -);
31. **S→X**: generarToken(coma, -);
32. **S→Y**: generarToken(dosPuntos, -);
33. **S→Z**: generarToken(opAritmetico, 2);

MATRIZ AFD

LEYENDA:

l: {a-z,A_Z}
d: {0 - 9}
$o.c$: cualquier carácter no contemplado en el nodo
$c1$: todos los caracteres – {*}
$c2$: todos los caracteres – {*,/}
$c3$: todos los caracteres – {“}
del: tab, espacio, eol

***Los estados encerrados EN VERDE representan estados finales (y por tanto no reciben nada).**
***Los números EN ROJO representan ERRORES.**

	l	d	_	c1	c2	c3	/	*	=	“	&	(	)	{	}	;	,	:	+	del	o.c
S	A 2	C 5	A 2				E 8	M 21	H 18	K 15	P 24	R 26	T 27	U 28	V 29	W 30	X 31	Y 32	Z 33	S 1	50
A	A 3	A 3	A 3																		B 4
B																					
C		C 6																			D 7
D																					
E								F 9													51
F				F 10				G 11													52
G					F 12		S 14	G 13													53
H									I 19												J 20
I																					
J																					
K						K 16				L 17											54
L																					
M								N 22													O 23
N																					
O																					
P											Q 25										55
Q																					
R																					
T																					
U																					
V																					
W																					
X																					
Y																					
Z																					

TABLA DE SÍMBOLOS

La Tabla de Símbolos diseñada tiene el siguiente formato:

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

> "CONTENIDOS DE LA TABLA # " + *número_de_la_TS* + " :" + **\n\n**

- **\n** : Salto de línea
- *número_de_la_TS* : número correspondiente a la TS actual

Línea del lexema

> "**\t*** LEXEMA :**\t**" + *lexema_del_id* + **\n**

- **\n** : Salto de línea
- **\t** : Tabulador
- *lexema_del_id* : nombre del identificador

Línea del atributo

> "**\t**ATRIBUTOS : **\n**" + **\t** + **"**+ " + *nombre_atributo* + " :" + **\t** + *valor_atributo* + **\n** + **\t** + "---------- ----------"

- **\n** : Salto de línea
- **\t** : Tabulador
- *nombre_atributo* : puede tomar los siguiente valores:
  - Despl
  - Tipo
  - numParam
  - TipoPararmXX
  - ModoParamXX
  - TipoRetorno
  - EtiqFuncion
  - Param

- *valor_atributo* : valor que tiene el atributo *nombre_atributo*

En esta primera entrega solamente habría una tabla (*número_de_la_TS* = 1) y un nombre de atributo (*nombre_atributo* = Despl) posible dado que no está implementado el Analizador Semántico.

Cada Tabla de Símbolo generada será volcada en un fichero "tabla.txt".
Un ejemplo del contenido de una tabla generada por nuestro Analizador Léxico:

CONTENIDOS DE LA TABLA # 1 :

* LEXEMA : 'a'
ATRIBUTOS :
+ despl : 0
---------- ----------

ERRORES

Léxicos

Código 50: Fue introducido un carácter no esperado.
Código 51: Fue introducido un carácter no válido. Se esperaba *.
Código 52: Fue introducido un carácter no esperado.
Código 53: Fue introducido un carácter no esperado.
Código 54: Fue introducido un carácter no esperado en la cadena.
Código 55: Fue introducido un carácter no válido. Se esperaba &.
Código 60: El número entero introducido está fuera de rango (es mayor que 32767 o menor que -32768).
Código 61: La cadena supera los 64 caracteres.

Los errores son volcados en un fichero “errores.txt”
Cada error en nuestro procesador tendrá la siguiente estructura en el fichero “errores.txt” :

“Línea ” + *num_línea* + “ – Código de error ” + *cod_error* + “ :” + **\n\t** + *mensaje_error*

- **\n** : Salto de línea
- **\t** : Tabulador
- *num_línea* : el número de línea en el fichero fuente donde se encuentra el error.
- *cod_error* : código de error según lo especificado anteriormente (50 || 51 || 52 || 53 || 54 || 55 || 60 || 61)
- *mensaje_error* : mensaje específico del error en el formato siguente

  “ERROR “ + *tipo_error* + “ – “ + *mensaje_descriptivo*

  - *tipo_error : LÉXICO* || *SINTÁCTICO* || *SEMÁNTICO*
  - *mensaje_descriptivo:* mensaje descriptivo del error que incluye el carácter, cadena o valor incorrecto, incluyendo el carácter esperado en determinados casos.

Un ejemplo del contenido de un error generado por nuestro Analizador Léxico:

Línea 1 - Código de error 51:
ERROR LÉXICO - carácter CAR: 'l' no válido. Se esperaba *.

ANEXOS

CORRECTOS

Caso de Prueba # 1

Código fuente

```
/*
    CASO DE PRUEBA #1
*/

print "Introduce el resultado de 6+6*2: ";
let a_1 int = 0;

input a_1;

let num int = 5;

if (a_1 == 18) {
    print "Bien!!";
    print "Introduce un número para sumarle a 18: ";
    input num
}

switch (a_1) {

    case 24:
        print ":( es 6*2 = 12 y luego 12 + 6 = 18 no 24";
        print "Introduce un número para sumarle a 24: ";
        input num;
        break;
    default:
        print "Vaya " + a_1 + " no es correcto.";
}


function Suma int (int a, int b) {

    j= a + b;
    return j;
    /* La función finaliza y devuelve el valor entero de la expresión */
}
```

```
print a_1 + " + " + num + " = ";
print  Suma(a_1,num);
```

Volcado

Fichero Tabla de Símbolos

CONTENIDOS DE LA TABLA # 1 :

* LEXEMA : 'a_1'
ATRIBUTOS :
+ despl : 0
---------- ----------
* LEXEMA : 'num'
ATRIBUTOS :
+ despl : 1
---------- ----------
* LEXEMA : 'Suma'
ATRIBUTOS :
+ despl : 2
---------- ----------
* LEXEMA : 'a'
ATRIBUTOS :
+ despl : 3
---------- ----------
* LEXEMA : 'b'
ATRIBUTOS :
+ despl : 4
---------- ----------
* LEXEMA : 'j'
ATRIBUTOS :
+ despl : 5
---------- ----------

Fichero Tokens

< print , >
< cadena , "Introduce el resultado de 6+6*2: " >
< ptoComa , >
< let , >
< id , 0 >
< int , >
< asignacion , >

< cteEntera , 0 >
< ptoComa , >
< input , >
< id , 0 >
< ptoComa , >
< let , >
< id , 1 >
< int , >
< asignacion , >
< cteEntera , 5 >
< ptoComa , >
< if , >
< abrirParentesis , >
< id , 0 >
< opRelacional , 1 >
< cteEntera , 18 >
< cerrarParentesis , >
< abrirCorchete , >
< print , >
< cadena , "Bien!!" >
< ptoComa , >
< print , >
< cadena , "Introduce un número para sumarle a 18: " >
< ptoComa , >
< input , >
< id , 1 >
< cerrarCorchete , >
< switch , >
< abrirParentesis , >
< id , 0 >
< cerrarParentesis , >
< abrirCorchete , >
< case , >
< cteEntera , 24 >
< dosPuntos , >
< print , >
< cadena , ":(es 6*2 = 12 y luego 12 + 6 = 18 no 24" >
< ptoComa , >
< print , >
< cadena , "Introduce un número para sumarle a 24: " >
< ptoComa , >
< input , >
< id , 1 >
< ptoComa , >
< break , >

< ptoComa , >
< default , >
< dosPuntos , >
< print , >
< cadena , "Vaya " >
< opAritmetico , 2 >
< id , 0 >
< opAritmetico , 2 >
< cadena , " no es correcto." >
< ptoComa , >
< cerrarCorchete , >
< function , >
< id , 2 >
< int , >
< abrirParentesis , >
< int , >
< id , 3 >
< coma , >
< int , >
< id , 4 >
< cerrarParentesis , >
< abrirCorchete , >
< id , 5 >
< asignacion , >
< id , 3 >
< opAritmetico , 2 >
< id , 4 >
< ptoComa , >
< return , >
< id , 5 >
< ptoComa , >
< cerrarCorchete , >
< print , >
< id , 0 >
< opAritmetico , 2 >
< cadena , " + " >
< opAritmetico , 2 >
< id , 1 >
< opAritmetico , 2 >
< cadena , " = " >
< ptoComa , >
< print , >
< id , 2 >
< abrirParentesis , >
< id , 0 >

< coma , >
< id , 1 >
< cerrarParentesis , >
< ptoComa , >
< eof , >

Caso de Prueba # 2

Código fuente

```
/*
    CASO DE PRUEBA #2: Probando todos los posibles tokens
*/

let unaCadena64 string = "Si la vida te da limones haz limonada. Quieres limones
?????????"

let _un_num_1 int = 32767;  /* a una unidad del error */

let a int = _un_num_1;

let bool boolean = true;

if (a == _un_num_1 &&bool) {
    bool = false;
}

function PrintRecursivo boolean (string cadena) {

    print cadena;

    if(bool == false) {
        bool = true;
        PrintRecursivo(cadena);
    }

    return true;
}

a *= a;
_un_num_1 = a * a;

switch(a){
    case _un_num_1:
```

```
        PrintRecursivo(unaCadena64);
        break;
    default:
        PrintRecursivo(unaCadena64);
}
```

Volcado

Fichero Tabla de Símbolos

CONTENIDOS DE LA TABLA # 1 :

* LEXEMA : 'unaCadena64'
ATRIBUTOS :
+ despl : 0
---------- ----------
* LEXEMA : '_un_num_1'
ATRIBUTOS :
+ despl : 1
---------- ----------
* LEXEMA : 'a'
ATRIBUTOS :
+ despl : 2
---------- ----------
* LEXEMA : 'bool'
ATRIBUTOS :
+ despl : 3
---------- ----------
* LEXEMA : 'PrintRecursivo'
ATRIBUTOS :
+ despl : 4
---------- ----------
* LEXEMA : 'cadena'
ATRIBUTOS :
+ despl : 5
---------- ----------

Fichero Tokens

< let , >
< id , 0 >
< string , >
< asignacion , >

< cadena , "Si la vida te da limones haz limonada. Quieres limones ?????????" >
< let , >
< id , 1 >
< int , >
< asignacion , >
< cteEntera , 32767 >
< ptoComa , >
< let , >
< id , 2 >
< int , >
< asignacion , >
< id , 1 >
< ptoComa , >
< let , >
< id , 3 >
< boolean , >
< asignacion , >
< true , >
< ptoComa , >
< if , >
< abrirParentesis , >
< id , 2 >
< opRelacional , 1 >
< id , 1 >
< opLogico , 1 >
< id , 3 >
< cerrarParentesis , >
< abrirCorchete , >
< id , 3 >
< asignacion , >
< false , >
< ptoComa , >
< cerrarCorchete , >
< function , >
< id , 4 >
< boolean , >
< abrirParentesis , >
< string , >
< id , 5 >
< cerrarParentesis , >
< abrirCorchete , >
< print , >
< id , 5 >
< ptoComa , >
< if , >

< abrirParentesis , >
< id , 3 >
< opRelacional , 1 >
< false , >
< cerrarParentesis , >
< abrirCorchete , >
< id , 3 >
< asignacion , >
< true , >
< ptoComa , >
< id , 4 >
< abrirParentesis , >
< id , 5 >
< cerrarParentesis , >
< ptoComa , >
< cerrarCorchete , >
< return , >
< true , >
< ptoComa , >
< cerrarCorchete , >
< id , 2 >
< asigMultiplicacion , >
< id , 2 >
< ptoComa , >
< id , 1 >
< asignacion , >
< id , 2 >
< opAritmetico , 1 >
< id , 2 >
< ptoComa , >
< switch , >
< abrirParentesis , >
< id , 2 >
< cerrarParentesis , >
< abrirCorchete , >
< case , >
< id , 1 >
< dosPuntos , >
< id , 4 >
< abrirParentesis , >
< id , 0 >
< cerrarParentesis , >
< ptoComa , >
< break , >
< ptoComa , >

< default , >
< dosPuntos , >
< id , 4 >
< abrirParentesis , >
< id , 0 >
< cerrarParentesis , >
< ptoComa , >
< cerrarCorchete , >
< eof , >

Caso de Prueba # 3

Código fuente

```
/* CASO # 3: Probando delimitadores */

let z1                          int     ;
let f_11                        boolean ;
let _cad                        string  ;
let n2                          int     ;
let l2                          boolean ;
input z1;
f_11 = l2;

if(z1&& l2)_cad="HELLO WORLD";
n2 *= z1 + 378;

print                           44
                                  *
                                  z1
                                  *
                                  n2;

function funcionMyFun boolean(boolean var2)
{
                                l2 =  var2;
                                if (l2) z1 = funcionMyFun (var2);
                                varglobal = 1099;
                                return var2;
}
if (funcionMyFun(l2))
    print varglobal;
```

Fichero Tabla de Símbolos

CONTENIDOS DE LA TABLA # 1 :

* LEXEMA : 'z1'
ATRIBUTOS :
+ despl : 0
---------- ----------
* LEXEMA : 'f_11'
ATRIBUTOS :
+ despl : 1
---------- ----------
* LEXEMA : '_cad'
ATRIBUTOS :
+ despl : 2
---------- ----------
* LEXEMA : 'n2'
ATRIBUTOS :
+ despl : 3
---------- ----------
* LEXEMA : 'l2'
ATRIBUTOS :
+ despl : 4
---------- ----------
* LEXEMA : 'funcionMyFun'
ATRIBUTOS :
+ despl : 5
---------- ----------
* LEXEMA : 'var2'
ATRIBUTOS :
+ despl : 6
---------- ----------
* LEXEMA : 'varglobal'
ATRIBUTOS :
+ despl : 7
---------- ----------

Fichero Tokens

< let , >
< id , 0 >
< int , >
< ptoComa , >
< let , >

< id , 1 >
< boolean , >
< ptoComa ,   >
< let , >
< id , 2 >
< string , >
< ptoComa ,   >
< let , >
< id , 3 >
< int , >
< ptoComa ,   >
< let , >
< id , 4 >
< boolean , >
< ptoComa ,   >
< input , >
< id , 0 >
< ptoComa ,   >
< id , 1 >
< asignacion ,   >
< id , 4 >
< ptoComa ,   >
< if , >
< abrirParentesis ,   >
< id , 0 >
< opLogico , 1 >
< id , 4 >
< cerrarParentesis ,   >
< id , 2 >
< asignacion ,   >
< cadena , "HELLO WORLD" >
< ptoComa ,   >
< id , 3 >
< asigMultiplicacion ,   >
< id , 0 >
< opAritmetico , 2 >
< cteEntera , 378 >
< ptoComa ,   >
< print , >
< cteEntera , 44 >
< opAritmetico , 1 >
< id , 0 >
< opAritmetico , 1 >
< id , 3 >
< ptoComa ,   >

< function , >
< id , 5 >
< boolean , >
< abrirParentesis , >
< boolean , >
< id , 6 >
< cerrarParentesis , >
< abrirCorchete , >
< id , 4 >
< asignacion , >
< id , 6 >
< ptoComa , >
< if , >
< abrirParentesis , >
< id , 4 >
< cerrarParentesis , >
< id , 0 >
< asignacion , >
< id , 5 >
< abrirParentesis , >
< id , 6 >
< cerrarParentesis , >
< ptoComa , >
< id , 7 >
< asignacion , >
< cteEntera , 1099 >
< ptoComa , >
< return , >
< id , 6 >
< ptoComa , >
< cerrarCorchete , >
< if , >
< abrirParentesis , >
< id , 5 >
< abrirParentesis , >
< id , 4 >
< cerrarParentesis , >
< cerrarParentesis , >
< print , >
< id , 7 >
< ptoComa , >
< eof , >

Caso de Prueba # 4

Código fuente

```
/*
    CASO DE PRUEBA #4 C#1 con errores
*/

print "Introduce el resultado de 6+6*2: ";

let a_1 int = 32800;  /* Este num es mayoy que el max */

input a_1;

/



let num int = 5;
let b boolean = true&false;

if (a_1 == 18) {
    print "Bien!!";
    print "Introduce un número para sumarle a 18: ";
    input num
}

switch (a_1) {

    case 24:
        print "Nooo :(. La precedencia de operadores obliga a realizar 6*2 = 12
primero y luego 12 + 6 = 18";
        print "Introduce un número para sumarle a 24: ";
        input num;
        break;
    default:
        print "Vaya " + a_1 + " no es correcto.";
}



function Suma int (int a, int b) {

    j= a - b;
    return j;
    /* La función finaliza y devuelve el valor entero de la expresión */
}
```

```
print a_1 + " + " + num + " = ";
print  Suma(a_1,num);
```

Fichero error

Línea 7 - Código de error 60:

ERROR LÉXICO - El número introducido está fuera de rango.

NUM: 32800 es mayor que 32767.

Línea 11 - Código de error 51:

ERROR LÉXICO - carácter CAR: '

' no válido. Se esperaba *.

Línea 15 - Código de error 55:

ERROR LÉXICO - carácter CAR: 'f' no válido. Se esperaba &.

Línea 26 - Código de error 61:

ERROR LÉXICO - La cadena introducida tiene más del número máximo de caracteres permitidos: 64.

CAD: 'Nooo :(. La precedencia de operadores obliga a realizar 6*2 = 12 primero y luego 12 + 6 = 18'.

Línea 37 - Código de error 50:

ERROR LÉXICO - carácter CAR: '-' no esperado.

Caso de Prueba # 5

Código fuente

```
/*
    CASO DE PRUEBA #5:
*/

let unaCadena64 string = "Si la vida te da limones haz limonada. Quieres limones
??????????"

let _un_num_1 int = 50000;

let a int = _un_num_1;

let bool boolean = true;

if (a <= _un_num_1 || bool != false & ) {
    bool = false;
```

```
}

/* Cod 52 😀 * /

/* Cod 53 *😀 * /

function PrintRecursivo boolean (string cadena) {

    print cadena;

    if(bool == false) {
        bool = true;
        PrintRecursivo(cadena);
    }

    return true;
}

a = a/a;


_un_num_1 = a * a;

switch(a){
    case _un_num_1:
        PrintRecursivo("Una cadena de más de 64 caracteres. Una cadena de más de 64
caracteres.");
        break;
    default:
        PrintRecursivo(unaCadena64);
}

unaCadena64 = "Emoji 😀";
```

Fichero error

Línea 7 - Código de error 60:
ERROR LÉXICO - El número introducido está fuera de rango.
NUM: 50000 es mayor que 32767.

Línea 13 - Código de error 50:
ERROR LÉXICO - carácter CAR: '<' no esperado.

Línea 13 - Código de error 50:
ERROR LÉXICO - carácter CAR: '|' no esperado.

Línea 13 - Código de error 50:
ERROR LÉXICO - carácter CAR: '|' no esperado.

Línea 13 - Código de error 50:
ERROR LÉXICO - carácter CAR: '!' no esperado.

Línea 13 - Código de error 55:

ERROR LÉXICO - carácter CAR: ' ' no válido. Se esperaba &.

Línea 17 - Código de error 52:
ERROR LÉXICO - carácter CAR: 'ð' no esperado.

Línea 19 - Código de error 53:
ERROR LÉXICO - carácter CAR: 'ð' no esperado.

Línea 33 - Código de error 51:
ERROR LÉXICO - carácter CAR: 'a' no válido. Se esperaba *.

Línea 40 - Código de error 61:
ERROR LÉXICO - La cadena introducida tiene más del número máximo de caracteres permitidos: 64.
CAD: 'Una cadena de mÃ¡s de 64 caracteres. Una cadena de mÃ¡s de 64 caracteres.'.

Línea 47 - Código de error 54:
ERROR LÉXICO - carácter CAR: 'Ÿ' no esperado dentro de la cadena

Caso de Prueba # 6

Código fuente

```
/* CASO # 6: */

let z1      int     ;
let f_11    boolean ;
let _cad    string  ;
let n2      int     ;
let l2      boolean ;
input z1;
f_11 = l2;


if(z1>n2 && l2)_cad='HELLO WORLD';

// Comentario incorrecto //

n2 *= z1 % 378;

print 44
            *
            z1
            *
            n2;

function funcionMyFun boolean(boolean var2)
{
      l2 =  var2;
      if (l2) z1 = funcionMyFun (var2);
      varglobal = 1099;
      return var2;
}
if (funcionMyFun(l2))
```

```
    print varglobal;
```

Fichero error

Línea 12 - Código de error 50:
ERROR LÉXICO - carácter CAR: '>' no esperado.

Línea 12 - Código de error 50:
ERROR LÉXICO - carácter CAR: "' no esperado.

Línea 12 - Código de error 50:
ERROR LÉXICO - carácter CAR: "' no esperado.

Línea 14 - Código de error 51:
ERROR LÉXICO - carácter CAR: '/' no válido. Se esperaba *.

Línea 14 - Código de error 51:
ERROR LÉXICO - carácter CAR: ' ' no válido. Se esperaba *.

Línea 14 - Código de error 51:
ERROR LÉXICO - carácter CAR: '/' no válido. Se esperaba *.

Línea 14 - Código de error 51:
ERROR LÉXICO - carácter CAR: '
' no válido. Se esperaba *.

Línea 16 - Código de error 50:
ERROR LÉXICO - carácter CAR: '%' no esperado.