INTEGRANTES:

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

PRÁCTICA PROCESADORES DEL LENGUAJE

Grupo 13

2022



**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

| :

LEYENDA:

: {a-z,A\_Z}

: {0 - 9}

: cualquier carácter no contemplado en el nodo

: todos los caracteres – {\*}

: todos los caracteres – {\*,/}

: todos los caracteres – {“}

: tab, espacio, eol

# AUTÓMATA

# 

# ACCIONES SEMÁNTICAS

# S→S: leer();

# S→A: lexema = c; leer();

# A→A: lexema = lexema + c; leer();

# 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);

# }

# S→C: valor = char\_int(d); leer();

# C→C: valor = valor \* 10 + char\_int(d); leer();

# C→D: if (valor > 32767) then error (60);

# else generarToken(cte\_entera, valor);

# S→E: leer();

# E→F: leer();

# F→F: leer();

# F→G: leer();

# G→F: leer();

# G→G: leer();

# G→S: leer();

# S→K: lexema = “ ”; leer();

# K→K: lexema = lexema + c; leer();

# K→L: if(lexema.length > 64 ) then error (61);

# else generarToken(cadena, lexema);

# S→H: leer();

# H→I: generarToken(opRelacional, 1);

# H→J: generarToken(asignación, -);

# S→M: leer();

# M→N: generarToken(asigMultiplicacion, - );

# M→O: generarToken(opAritmetico, 1 );

# S→P: leer();

# P→Q: generarToken(opLogico, 1 );

# S→R: generarToken(abrirParentesis, - );

# S→T: generarToken(cerrarParentesis, - );

# S→U: generarToken(abrirCorchete, - );

# S→V: generarToken(cerrarCorchete, - );

# S→W: generarToken(ptoComa, - );

# S→X: generarToken(coma, - );

# S→Y: generarToken(dosPuntos, - );

# S→Z: generarToken(opAritmetico, 2 );

# MATRIZ AFD

LEYENDA:

: {a-z,A\_Z}

: {0 - 9}

: cualquier carácter no contemplado en el nodo

: todos los caracteres – {\*}

: todos los caracteres – {\*,/}

: todos los caracteres – {“}

: tab, espacio, eol

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

**\*Los números EN ROJO representan ERRORES.**

# 

# 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

# “\tATRIBUTOS : \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.