Práctica Traductores de Lenguajes

G-Compilador

Andrés Ollero Morales
Víctor Alejandro Sanz Ararat
Gabriel de Oliveira Trindade

Departamento de Lenguajes y Sistemas Informáticos e Ingeniería de Software
Escuela Técnica Superior De Ingenieros Informáticos
Universidad Politécnica de Madrid
Junio 2023

Índice

1. Diseño final del Procesador

A modo de introducción, hemos querido recordar lo que es capaz de procesar el compilador. La implementación fue realizada en el lenguaje de programación Java. Nuestro grupo contaba con las siguientes opciones de práctica:

- Técnicas de Análisis Sintáctico: Descendente con tabla
- Sentencias: Sentencia de selección múltiple (switch-case)
- Operadores especiales: Asignación con resto (%=)
- Comentarios: Comentario de bloque (/* */)
- Cadenas: Con comillas dobles (¨¨)

1.1. Diseño de Tokens

```
<palabraReservada, boolean>         <palabraReservada, input>
<palabraReservada, break>           <palabraReservada, int>
<palabraReservada, case>            <palabraReservada, let>
<palabraReservada, function>        <palabraReservada, switch>
<palabraReservada, print>           <palabraReservada, return>
<palabraReservada, string>          <palabraReservada, if>
<palabraReservada, default>

<suma, >            <puntoComa, >       <asignacion, >
<negacion, >        <coma, >            <dosPuntos, >
<abrePar, >         <cierraPar, >       <comparacion, >
<abreLlave, >       <cierraLlave, >     <asignacionResto, >
<id, nºTS>          <constEnt, nº>      <cadena, "lexema">
```

1.2. Gramática de Contexto Libre (Tipo II)

Esta gramática es sobre la que hacemos las acciones semánticas mediante Traducción Dirigida por la Sintáxis.

```
Axioma = PP

NoTerminales = { PP P S SS E R RR U UU V VV L Q X B T A K C F H O D }

Terminales = { ! == + id ( ) constEnt cadena %= print input return , if break
               switch case int boolean string let function ; : = { } default }

Producciones = {
    E -> R RR
    RR -> == R RR
    RR -> lambda
    R -> U UU
    UU -> + U UU
    UU -> lambda
```

```
    U -> ! V
    U -> V
    V -> id VV
    V -> ( E )
    V -> constEnt
    V -> cadena
    VV -> ( L )
    VV -> lambda
    S -> id SS
    SS -> %= E ;
    SS -> = E ;
    SS -> ( L ) ;
    S -> print R ;
    S -> input id ;
    S -> return X ;
    L -> E Q
    L -> lambda
    Q -> , E Q
    Q -> lambda
    X -> E
    X -> lambda
    B -> switch ( E ) { O }
    B -> if ( E ) S
    O -> case constEnt : C D O
    O -> default : C D O
    O -> lambda
    D -> break ;
    D -> lambda
    B -> let id T ;
    T -> int
    T -> boolean
    T -> string
    B -> S
    F -> function id H ( A ) { C }
    H -> T
    H -> lambda
    A -> T id K
    A -> lambda
    K -> , T id K
    K -> lambda
    C -> B C
    C -> lambda
    P -> B P
    P -> F P
    P -> lambda
    PP -> P
}
```

2. Diseño del Generador de Código Intermedio

2.1. Traducción Dirigida Por la Sintáxis con Acciones Semánticas

Aprovechamos el diseño del Esquema de Traducción del procesador para añadir las nuevas acciones semánticas del Generador de Código Intermedio. A continuación se muestra una breve descripción de todas los métodos y funciones que aparecen en las acciones.

Métodos:

emite(operador, op1, op2, res): Escribe en el fichero de código intermedio la operación descrita por el cuarteto (algunos operandos pueden valer nulo).

buscaEtTS(idPos): Busca la etiqueta por la posición de la Tabla de Símbolos correspondiente del id que la representa.

nuevaEt(): Crea una nueva etiqueta en la Tabla de Símbolos.

tamRAcalculator(): Calcula cuánto ocupa el tamaño del Registro de Activación de una función en pila.

buscaTipoTS(idPos): Busca el tipo de un id según su posición en la Tabla de Símbolos correspondiente.

buscaLugarTS(idPos): Busca la posición en la Tabla de Símbolos correspondiente el id determinado.

Atributos de los No Terminales:

.evaluado: Determina si el case de un switch ya ha sido evaluado, para así no hacer las comprobaciones de los siguientes cases y los ejecute directamente si no hay break en dicho case.

.siguiente:

.pos: Determina cuál es la posicion dentro de una tabla de símbolos

.lugar: Determina cuál es el lugar de la variable dentro de una tabla de símbolos

.break: Determina la etiqueta a la que se debe saltar dentro de un switch para saltar fuera del switch cuando se encuentra un break

.tipo: Determina el tipo del no terminal correspondiente

.etiq: Determina cual es la etiqueta que tiene una función o una string

.params: Es un Array que contiene el lugar de los parámetros que se van a pasar a la llamada a función siguiente

Bloque Axioma:

PP → P $\{\emptyset\}_1$

P → B P $\{\emptyset\}_2$

P → F P $\{\emptyset\}_3$

P → λ $\{\emptyset\}_4$

Bloque declaración de funciones:

F → function id H (A) {emite(':', buscaEtTS(id.pos), NULL, NULL)}$_{5,2}$
{ C {tamRAcalculator()}$_{5,3}$ } {emite('return', NULL, NULL, NULL)}$_{5,4}$

H → T $\{\emptyset\}_{6}$

H → λ $\{\emptyset\}_{7}$

A → T id K $\{\emptyset\}_{8}$

A → λ $\{\emptyset\}_{9}$

K → , T id K $\{\emptyset\}_{10}$

K → λ $\{\emptyset\}_{11}$

C → B C $\{\emptyset\}_{12}$

C → λ $\{\emptyset\}_{13}$

Bloque sentencias compuestas y declaración de variables:

B → if (E) {B.siguiente := nuevaEt(), emite('if', E.lugar, NULL, B.siguiente)}$_{14,1}$
S {emite(':', B.siguiente, NULL, NULL)}$_{14}$

B → switch (E) { O } {emite(':', B.break, NULL, NULL)}$_{15,3}$

O → case constEnt { O.break := B.break, O.evaluado := B.evaluado
O.siguiente := nuevaEtiq(), O.lugar := E.lugar
emite('if==', B.evaluado, 1, sig_inst + 1)
emite('if!=', O.lugar, constEnt, O.siguiente) }$_{16,1}$
: C D {emite(':', 1, NULL, O.evaluado) emite(':', O.siguiente, NULL, NULL)}$_{16,3}$
O $\{\emptyset\}_{16,2}$

O → λ $\{\emptyset\}_{18}$

D → break ; {emite('goto', D.break, NULL, NULL)}$_{19}$

D → λ $\{\emptyset\}_{20}$

B → let id T ; $\{\emptyset\}_{21}$

T → int $\{\emptyset\}_{22}$

T → boolean $\{\emptyset\}_{23}$

T → string $\{\emptyset\}_{24}$

B → S $\{\emptyset\}_{25}$

Sentencias simples:

S → id SS {SS.tipo := BuscaTipoTS(id.pos)
SS.lugar := BuscaLugarTS(id.pos)
if SS.tipo == funcion then SS.etiq := BuscaEtiqTS(id.pos)}$_{26}$

SS → %= E ; {emite(‘ %’, E.lugar, SS.lugar, SS.lugar)}$_{27}$

SS → = E ; {emite(‘:=’, E.lugar, NULL, SS.lugar)}$_{28}$

SS → (L) ; {for (param in L.params)
emite(‘param’, param, NULL, NULL)
emite(‘call’,SS.etiq, NULL, NULL)}$_{29}$

S → print R ; {emite(“print”, R.lugar, NULL, NULL)}$_{30}$

S → input id ; {emite(“input”, BuscaLugarTS(id), NULL, NULL)}$_{31}$

S → return X ; {emite(‘return’, X.lugar, NULL, NULL)}$_{32}$

L → E Q {L.param := E.lugar $\bigoplus Q.param$}$_{33}$

L → λ {$\emptyset$}$_{34}$

Q → , E Q1 {Q.param := E.lugar $\bigoplus Q1.param$}$_{35}$

Q → λ {$\emptyset$}$_{36}$

X → E {X.lugar := E.lugar}$_{37}$

X → λ {$\emptyset$}$_{38}$

Expresiones:

E → R RR { if RR.tipo != vacio
then E.lugar := nuevaTemp(logico)
emite(‘if==’, R.lugar, RR.lugar, goto 2)
emite(‘:=’, 0, null, E.lugar)
emite(‘goto’, sigInst + 1)
emite(‘:=’, 1, null, E.lugar))
else then
E.lugar := R.lugar
emite(‘:=’, R.lugar, null, E.lugar)}$_{39}$

RR → == R RR {RR.lugar := R.lugar}$_{40}$

RR → λ {$\emptyset$}$_{41}$

R → U UU {UU.lugar := nuevaTemp(constEnt)
if UU1.tipo == vacio
then emite(':=', U.lugar, null, UU.lugar)
else
then emite('+', U.lugar, UU1.lugar, UU.lugar)}$_{43}$

UU → + U UU {if (U.tipo = constEnt && UU2.tipo != tipo.error)
then UU.tipo := tipo.ok
else then UU.tipo := tipo.error
error("error semantico: operando solo admite valores enteros");
POP(3)}$_{44}$

UU → λ {Ø}$_{45}$

U → ! V {U.lugar := nuevaTemp(boolean)
emite('not', V.lugar, NULL, U.lugar}$_{46}$

U → V {U.lugar := V.lugar}$_{47}$

V → id VV {if buscaTipoTS(id.pos) != function
then V.lugar := buscaLugarTS(id.pos)
else then
VV.etiq := BuscaEtiqTS(id.pos)
V.lugar = VV.lugar}$_{48}$

V → (E) {V.lugar := E.lugar}$_{49}$

V → constEnt {V.lugar := nuevaTemp(tipo.constEnt);
emite(':=', constEnt.valor, NULL, V.lugar)}$_{50}$

V → cadena {V.lugar := nuevaTemp(tipo.cadena);
emite(':=', cadena.etiqueta, NULL, V.lugar)}$_{51}$

VV → (L) {VV = nuevaTemp()
for (param in L.params)
emite('param', param, NULL, NULL)
emite('call', VV.etiq, NULL, VV.lugar)}$_{52}$

VV → λ {Ø}$_{53}$

3. Diseño del Generador de Código Final

3.1. Plantilla de Traducción de Cuartetos

A continuación se muestran todos los posibles cuartetos generados por el Generador de Código Intermedio, y su correspondiente traducción a Código Objeto. Nótese que para cada operando que aparece en el cuarteto, se recogerá la dirección del mismo de una forma u otra (puede ser una constante, o estar en alguan tabla de símbolos...). También algunas traducciones pueden variar dependiendo del caso en el que estemos ejecutando (dentro de una función o desde main).

Emitiremos la traducción a ensamblador en el momento en el que se haga un emite del código intermedio.

- (+, op1, op2, res)

```
ADD op1, op2
MOVE .A, res
```

- (:=, op1, NULL, res)

```
MOVE op1, res
```

- (:, et01, null, null)

```
et01: NOP
```

- (%, op1, op2, res)

```
MOD op1, op2
MOVE .A, res
```

- (print, op1, null, null)
  En el caso de que sea una string:

```
ADD #0, op1
WRSTR [.A]
```

  Accedemos accedemos a lo que apunta el acumulador, ya que es una etiqueta a una string.

  En el caso de que sea un entero:

```
WRINT op1
```

- (input, op1, null, null)
  En el caso de que sea una string:

```
INSTR op1
```

  En el caso de que sea un entero:

```
ININT op1
```

- (return, op1, null, null)

```
SUB #Tam_RA_X, #1
ADD .A, .IX
MOVE op1, [.A]
BR [.IX]
```

- (return, op1, null, null)

```
BR [.IX]
```

- (not, op1, null, res)

```
XOR op1, #1
MOVE .A, res
```

- (goto, null, null, sig_int + 1)

```
BR $3
```

- (goto, et1, null, null)

```
BR /et1
```

- (if, op1, null, et01)

```
CMP op1, #1
BNZ /et01
```

- (if==, op1, op2, 2)

```
CMP op1, op2
BZ $5
```

- (if!=, op1, constEnt, et01)

```
CMP op1, constEnt
BNZ /et01
```

- (call, et1, null, res)

```
ADD #Tam_RA_X, .IX
MOVE #dir_ret_X, [.A]
MOVE .A, .IX
BR /etX
dir_ret_X: NOP
SUB #Tam_RA_X, #1
ADD .A, .IX
MOVE [.A], .R9
SUB .IX, #Tam_RA_X
MOVE .A, .IX
MOVE .R9, res
```

- (call, et1, null, null)

```
ADD #Tam_RA_X, .IX
MOVE #dir_ret_X, [.A]
MOVE .A, .IX
BR /etX
dir_ret_X: NOP
SUB .IX, #Tam_RA_X
MOVE .A, .IX
```

- (param, op1, null, null)

```
ADD #0, .IX
ADD #desp, .A
MOVE op1, [.A]
```

3.2. Tratamiento de las Strings

Las strings se tratan de manera que cada cadena de texto se almacena en el ensamblador con una etiqueta generada, esto es posible gracias a la instrucción DATA, en el caso de que la cadena este implícita en el código.

En el caso en el que la cadena es recibida por medio de input, se reservan 64 espacios de memoria con la instrucción RES 64 y una etiqueta para que se almacenene la string.

Todas las cadenas se inicializan a la string vacia, ¨¨.

4. Diseño del Entorno de Ejecución

4.1. Esquema de la memoria en ejecución

La asignación de la memoria se realiza como describe la siguiente figura:

Figura 1: Estructuración de las zonas de memoria.

La primera parte de la memoria será la zona de Datos Estáticos, en donde se cargarán variables globales y datos temporales de la ejecución de main. La segunda zona son las instrucciones

en ensamblador. E inmediatamente debajo de ellas, está la zona de pila, que comienza justo después de la última instruccion emitida, y carga todos los Registros de Activación a partir de ellas. Esta última crece hacia posiciones crecientes de memoria.

Utilizamos los registros .IX para acceder a la zona de pila, y .IY para la zona de datos estáticos, ya que estos aceptan direccionamientos relativos.

4.2. Diseño del Registro de Activación

El Registro de Activación será apilado en la zona de memoria de la pila una consecutivamente de la otra. Está compuesto de los siguientes campos:

- Estado de la máquina: Contiene la información necesaria para restaurar la ejecución por donde se dejó antes de la llamada a la subrutina. En nuestro caso solo contendrá la dirección de retorno.
- Parámetros: Contiene los valores necesarios con los que se inicializan los parámetros que utiliza la función.
- Variables Locales: Todas las variables que se declaren en la función, se almacenarán sus valores en este campo.
- Datos Temporales: Aquí se almacenarán valores como constantes enteras, operaciones aritméticas o evaluaciones de condiciones booleanas o de parámetros a llamadas de otras funciones.
- Valor Devuelto: El llamado cargará en este campo el valor devuelto

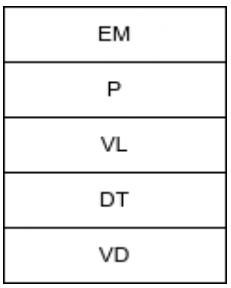

Figura 2: Campos del Registro de Activación.

Los campos EM y P los carga el llamante, el resto los usa la propia función llamada. Y el valor devuelto se encarga de ubicarlo el llamado, para que lo pueda recojer el llamante posteriormente. En la práctica, los campos de variables locales y datos temporales se entremezclan entre ellos, ya que la función los utiliza de forma indiscriminada.

A. Casos de Prueba

A.1. Prueba Funcional 1

Código fuente:

```
let a int;
let   b  int  ;
let bbb boolean ;
a = 3;
b=a                                                                    ;
let c boolean ;
c = a == b;
if (c) b %= 1;
if (b == a) b = 44;
a = a + b;
print a ;
print b;
```

Fichero de lenguaje intermedio generado:

```
:= 3 null 0.4
:= 0.4 null 0.1
:= 0.1 null 0.2
if== 0.1 0.2 goto 2
:= 0 null 0.6
goto null null 1
:= 1 null 0.6
:= 0.6 null 0.5
if 0.5 null et01
:= 1 null 0.7
% 0.7 0.2 0.2
: et01 null null
if== 0.2 0.1 goto 2
:= 0 null 0.8
goto null null 1
:= 1 null 0.8
if 0.8 null et02
:= 44 null 0.9
:= 0.9 null 0.2
: et02 null null
:= 0.2 null 0.10
+ 0.1 0.10 0.11
:= 0.11 null 0.1
print 0.1 null null
print 0.2 null null
```

Fichero de lenguaje objeto generado:

```
MOVE #inicio_estaticas, .IY
MOVE #inicio_pila, .IX
BR /main

main:
    MOVE #3, #4[.IY]
    MOVE #4[.IY], #1[.IY]
    MOVE #1[.IY], #2[.IY]
    CMP #1[.IY],#2[.IY]
    BZ $5MOVE0,6[.IY]BR3
    MOVE #1, #6[.IY]
    MOVE #6[.IY], #5[.IY]
    CMP #5[.IY], #1
```

```
    BNZ /et01
    MOVE #1, #7[.IY]
    MOD #2[.IY], #7[.IY]
    MOVE .A, #2[.IY]
et01: NOP
    CMP #2[.IY],#1[.IY]
    BZ 5MOVE0,8[.IY]BR3
    MOVE #1, #8[.IY]
    CMP #8[.IY], #1
    BNZ /et02
    MOVE #44, #9[.IY]
    MOVE #9[.IY], #2[.IY]
et02: NOP
    MOVE #2[.IY], #10[.IY]
    ADD #1[.IY], #10[.IY]
    MOVE .A, #11[.IY]
    MOVE #11[.IY], #1[.IY]
    WRINT #1[.IY]
    WRINT #2[.IY]
    HALT

inicio_estaticas: RES 11


inicio_pila: NOP

END
; una obra de diriG studios(c) 2023.
```

Fichero de Tabla de simbolos generado:

```
#0:
* LEXEMA : 'a'
  ATRIBUTOS :
    +tipo : 'constEnt'
    +despl : 0
* LEXEMA : 'b'
  ATRIBUTOS :
    +tipo : 'constEnt'
    +despl : 1
* LEXEMA : 'bbb'
  ATRIBUTOS :
    +tipo : 'booleanR'
    +despl : 2
* LEXEMA : 'temp31'
  ATRIBUTOS :
    +tipo : 'constEnt'
    +despl : 3
* LEXEMA : 'c'
  ATRIBUTOS :
    +tipo : 'booleanR'
    +despl : 4
* LEXEMA : 'temp51'
  ATRIBUTOS :
    +tipo : 'booleanR'
    +despl : 5
* LEXEMA : 'temp61'
  ATRIBUTOS :
    +tipo : 'constEnt'
    +despl : 6
* LEXEMA : 'temp71'
  ATRIBUTOS :
    +tipo : 'booleanR'
```

```
    +despl : 7
* LEXEMA : 'temp81'
  ATRIBUTOS :
    +tipo : 'constEnt'
    +despl : 8
* LEXEMA : 'temp91'
  ATRIBUTOS :
    +tipo : 'constEnt'
    +despl : 9
* LEXEMA : 'temp101'
  ATRIBUTOS :
    +tipo : 'constEnt'
    +despl : 10
```

En este primer ejemplo probamos la declaración de variables, la asignación, con y sin resto, las comparaciones y los ïf"simples, asi como el print.

A.2. Prueba Funcional 2

Código fuente:

```
let a    int   ;
let b     int  ;
let number int ;
print  "Introduce el primer operando" ;
input a;
print "Introduce el segundo operando";input
  b;
function operacion int(int num2,int num1)
{
  let  res int;
  res=num1+num2;
  return ((res));
}
number = 0;
print operacion(b,a);
```

Fichero de lenguaje intermedio generado:

```
:= "Introduce el primer operando" null 0.4
print 0.4 null null
input 0.1 null null
:= "Introduce el segundo operando" null 0.5
print 0.5 null null
input 0.2 null null
: Etoperacion null null
:= 1.1 null 1.4
+ 1.2 1.4 1.5
:= 1.5 null 1.3
return 1.3 null null
return null null null
:= 0 null 0.7
:= 0.7 null 0.3
param 0.2 null null
param 0.1 null null
call Etoperacion null 0.8
print 0.8 null null
```

Fichero de lenguaje objeto generado:

```
MOVE #inicio_estaticas, .IY
MOVE #inicio_pila, .IX
```

```
BR /main

main:
    MOVE #data01, #4[.IY]
    ADD #0, #4[.IY]
    WRSTR [.A]
    ININT #1[.IY]
    MOVE #data02, #5[.IY]
    ADD #0, #5[.IY]
    WRSTR [.A]
    ININT #2[.IY]
    MOVE #0, #7[.IY]
    MOVE #7[.IY], #3[.IY]
    ADD #0, .IX
    ADD #1, .A
    MOVE #2[.IY], [.A]
    ADD #0, .IX
    ADD #2, .A
    MOVE #1[.IY], [.A]
    MOVE #dir_ret_1, [.IX]
    BR /Etoperacion
dir_ret_1:  NOP
    MOVE [.A], #8[.IY]
    MOVE #inicio_pila, .IX
    WRINT #8[.IY]
    HALT

Etoperacion:  NOP
    MOVE #1[.IX], #4[.IX]
    ADD #2[.IX], #4[.IX]
    MOVE .A, #5[.IX]
    MOVE #5[.IX], #3[.IX]
    SUB #Tam_RA_Etoperacion, #1
    ADD .A, .IX;
    MOVE #3[.IX], [.A]
    BR [.IX]

    BR [.IX]

Tam_RA_Etoperacion: EQU 7
inicio_estaticas: RES 8

data01: DATA "Introduce el primer operando "
data02: DATA "Introduce el segundo operando "

inicio_pila: NOP

END
; una obra de diriG studios(c) 2023.
```

Fichero de Tabla de simbolos generado:

```
#1:
* LEXEMA : 'num2'
  ATRIBUTOS :
    +tipo : 'constEnt'
    +despl : 0
* LEXEMA : 'num1'
  ATRIBUTOS :
    +tipo : 'constEnt'
    +despl : 1
* LEXEMA : 'res'
  ATRIBUTOS :
```

```
    +tipo : 'constEnt'
    +despl : 2
* LEXEMA : 'temp31'
  ATRIBUTOS :
    +tipo : 'constEnt'
    +despl : 3
* LEXEMA : 'temp41'
  ATRIBUTOS :
    +tipo : 'constEnt'
    +despl : 4
#0:
* LEXEMA : 'a'
  ATRIBUTOS :
    +tipo : 'constEnt'
    +despl : 0
* LEXEMA : 'b'
  ATRIBUTOS :
    +tipo : 'constEnt'
    +despl : 1
* LEXEMA : 'number'
  ATRIBUTOS :
    +tipo : 'constEnt'
    +despl : 2
* LEXEMA : 'temp31'
  ATRIBUTOS :
    +tipo : 'cadena'
    +despl : 3
* LEXEMA : 'temp41'
  ATRIBUTOS :
    +tipo : 'cadena'
    +despl : 4
* LEXEMA : 'operacion'
  ATRIBUTOS :
    +tipo : 'funcion'
     +numParam : 2
      +TipoParam1 : 'constEnt'
       +ModoParam1 : 1
      +TipoParam2 : 'constEnt'
       +ModoParam2 : 1
    +TipoRetorno : 'constEnt'
    +EtiqFuncion : 'Etoperacion01'
* LEXEMA : 'temp61'
  ATRIBUTOS :
    +tipo : 'constEnt'
    +despl : 5
* LEXEMA : 'temp71'
  ATRIBUTOS :
    +tipo : 'constEnt'
    +despl : 6
```

En este segundo ejemplo probamos la declaración de variables, el print de cadenas, las llamadas a funciones que retornan valor, y el uso de print de un entero con llamada a una función.

A.3. Prueba Funcional 3

Código fuente:

```
let texto string;
function alert (string m_s_g)
{
  print "Mensaje introducido:";
```

```
  print m_s_g;
}
function pideTexto ()
{
  print "Introduce un texto corto";
  input texto;
}
pideTexto();
let  textoAux  string;textoAux = texto;
alert
 (
 textoAux
 )
 ;
```

Fichero de lenguaje intermedio generado:

```
:= "" null 0.1
: Etalert null null
:= "Mensaje introducido:" null 1.2
print 1.2 null null
print 1.1 null null
return null null null
: EtpideTexto null null
:= "Introduce un texto corto" null 2.1
print 2.1 null null
input 0.1 null null
return null null null
call EtpideTexto null null
:= "" null 0.4
:= 0.1 null 0.4
param 0.4 null null
call Etalert null null
```

Fichero de lenguaje objeto generado:

```
MOVE #inicio_estaticas, .IY
MOVE #inicio_pila, .IX
BR /main

main:
    MOVE #data01, #1[.IY]
    MOVE #dir_ret_1, [.IX]
    BR /EtpideTexto
dir_ret_1:  NOP
    MOVE #inicio_pila, .IX
    MOVE #data05, #4[.IY]
    MOVE #1[.IY], #4[.IY]
    ADD #0, .IX
    ADD #1, .A
    MOVE #4[.IY], [.A]
    MOVE #dir_ret_2, [.IX]
    BR /Etalert
dir_ret_2:  NOP
    MOVE #inicio_pila, .IX
    HALT

Etalert:  NOP
    MOVE #data02, #2[.IX]
    ADD #0, #2[.IX]
    WRSTR [.A]
    ADD #0, #1[.IX]
    WRSTR [.A]
```

```
28
29      BR [.IX]
30
31  EtpideTexto:  NOP
32      MOVE #data03, #1[.IX]
33      ADD #0, #1[.IX]
34      WRSTR [.A]
35      INSTR /data04
36      MOVE #data04, #1[.IY]
37
38      BR [.IX]
39
40  Tam_RA_Etalert: EQU 4
41  Tam_RA_EtpideTexto: EQU 3
42  inicio_estaticas: RES 4
43
44  data01: DATA " "
45  data02: DATA "Mensaje introducido: "
46  data03: DATA "Introduce un texto corto "
47  data04: RES 64
48  data05: DATA " "
49
50  inicio_pila: NOP
51
52  END
53  ; una obra de diriG studios(c) 2023.
```

Fichero de Tabla de simbolos generado:

```
1  #1:
2  * LEXEMA : 'm_s_g'
3    ATRIBUTOS :
4      +tipo : 'cadena'
5      +despl : 0
6  * LEXEMA : 'temp11'
7    ATRIBUTOS :
8      +tipo : 'cadena'
9      +despl : 1
10 #2:
11 * LEXEMA : 'temp01'
12   ATRIBUTOS :
13     +tipo : 'cadena'
14     +despl : 0
15 #0:
16 * LEXEMA : 'texto'
17   ATRIBUTOS :
18     +tipo : 'cadena'
19     +despl : 0
20 * LEXEMA : 'alert'
21   ATRIBUTOS :
22     +tipo : 'funcion'
23      +numParam : 1
24       +TipoParam1 : 'cadena'
25        +ModoParam1 : 1
26     +TipoRetorno : 'vacio'
27     +EtiqFuncion : 'Etalert01'
28 * LEXEMA : 'pideTexto'
29   ATRIBUTOS :
30     +tipo : 'funcion'
31      +numParam : 0
32     +TipoRetorno : 'vacio'
33     +EtiqFuncion : 'EtpideTexto01'
34 * LEXEMA : 'textoAux'
```

```
  ATRIBUTOS :
    +tipo : 'cadena'
    +despl : 1
```

En este ejemplo realizamos varias pruebas de llamada a funciones, con input's de string y print de ellos, además de probar el funcionamiento de tanto variables locales como globales.

A.4. Prueba Funcional 4

Código fuente:

```
function recursiva int (int a)
{
    switch (a)
    {
        case 1:
            recursiva(a + 1);
            break;
        case 2:
            print("segundo case");
        case 3:
            print("tercer case");
        break;
    }
    return 14;
}

let entero int;
entero = 14;
if (recursiva(1) == entero)
    print(entero);
```

Fichero de lenguaje intermedio generado:

```
: Etrecursiva null null
if== 1.2 1 2
if!= 1.1 1 et02
:= 1 null 1.3
:= 1.3 null 1.4
+ 1.1 1.4 1.5
param 1.5 null null
call Etrecursiva null null
goto et01 null null
:= 1 null 1.2
: et02 null null
if== 1.2 1 2
if!= 1.1 2 et03
:= "segundo case" null 1.6
print 1.6 null null
:= 1 null 1.2
: et03 null null
if== 1.2 1 2
if!= 1.1 3 et04
:= "tercer case" null 1.7
print 1.7 null null
goto et01 null null
:= 1 null 1.2
: et04 null null
: et01 null null
:= 14 null 1.8
return 1.8 null null
return null null null
```

```
:= 14 null 0.3
:= 0.3 null 0.2
:= 1 null 0.4
param 0.4 null null
call Etrecursiva null 0.5
if== 0.5 0.2 goto 2
:= 0 null 0.6
goto null null 1
:= 1 null 0.6
if 0.6 null et05
print 0.2 null null
: et05 null null
```

Fichero de lenguaje objeto generado:

```
MOVE #inicio_estaticas, .IY
MOVE #inicio_pila, .IX
BR /main

main:
    MOVE #14, #3[.IY]
    MOVE #3[.IY], #2[.IY]
    MOVE #1, #4[.IY]
    ADD #0, .IX
    ADD #1, .A
    MOVE #4[.IY], [.A]
    MOVE #dir_ret_2, [.IX]
    BR /Etrecursiva
dir_ret_2:  NOP
    MOVE [.A], #5[.IY]
    MOVE #inicio_pila, .IX
    CMP #5[.IY],#2[.IY]
    BZ $5
    MOVE #0, #6[.IY]
    BR $3
    MOVE #1, #6[.IY]
    CMP #6[.IY], #1
    BNZ /et05
    WRINT #2[.IY]
et05: NOP
    HALT

Etrecursiva:  NOP
    CMP #2[.IX],#1
    BZ $5
    CMP #1[.IX], #1
    BNZ /et02
    MOVE #1, #3[.IX]
    MOVE #3[.IX], #4[.IX]
    ADD #1[.IX], #4[.IX]
    MOVE .A, #5[.IX]
    ADD #Tam_RA_Etrecursiva, .IX
    ADD #1, .A
    MOVE #5[.IX], [.A]
    ADD #Tam_RA_Etrecursiva, .IX
    MOVE #dir_ret_1, [.A]
    MOVE .A, .IX
    BR /Etrecursiva
dir_ret_1:  NOP
    SUB .IX, #Tam_RA_Etrecursiva
    MOVE .A, .IX
    BR /et01
    MOVE #1, #2[.IX]
```

```
et02: NOP
    CMP #2[.IX],#1
    BZ $5
    CMP #1[.IX], #2
    BNZ /et03
    MOVE #data01, #6[.IX]
    ADD #0, #6[.IX]
    WRSTR [.A]
    MOVE #1, #2[.IX]
et03: NOP
    CMP #2[.IX],#1
    BZ $5
    CMP #1[.IX], #3
    BNZ /et04
    MOVE #data02, #7[.IX]
    ADD #0, #7[.IX]
    WRSTR [.A]
    BR /et01
    MOVE #1, #2[.IX]
et04: NOP
et01: NOP
    MOVE #14, #8[.IX]
    SUB #Tam_RA_Etrecursiva, #1
    ADD .A, .IX;
    MOVE #8[.IX], [.A]
    BR [.IX]

    BR [.IX]

Tam_RA_Etrecursiva: EQU 10
inicio_estaticas: RES 6

data01: DATA "segundo case "
data02: DATA "tercer case "

inicio_pila: NOP

END
; una obra de diriG studios(c) 2023.
```

Fichero de Tabla de simbolos generado:

```
#1:
* LEXEMA : 'a'
  ATRIBUTOS :
    +tipo : 'constEnt'
    +despl : 0
* LEXEMA : 'temp11'
  ATRIBUTOS :
    +tipo : 'constEnt'
    +despl : 1
* LEXEMA : 'temp21'
  ATRIBUTOS :
    +tipo : 'constEnt'
    +despl : 2
* LEXEMA : 'temp31'
  ATRIBUTOS :
    +tipo : 'constEnt'
    +despl : 3
* LEXEMA : 'temp41'
  ATRIBUTOS :
    +tipo : 'constEnt'
    +despl : 4
```

```
* LEXEMA : 'temp51'
  ATRIBUTOS :
    +tipo : 'cadena'
    +despl : 5
* LEXEMA : 'temp61'
  ATRIBUTOS :
    +tipo : 'cadena'
    +despl : 6
* LEXEMA : 'temp71'
  ATRIBUTOS :
    +tipo : 'constEnt'
    +despl : 7
#0:
* LEXEMA : 'recursiva'
  ATRIBUTOS :
    +tipo : 'funcion'
     +numParam : 1
      +TipoParam1 : 'constEnt'
       +ModoParam1 : 1
    +TipoRetorno : 'constEnt'
    +EtiqFuncion : 'Etrecursiva01'
* LEXEMA : 'entero'
  ATRIBUTOS :
    +tipo : 'constEnt'
    +despl : 0
* LEXEMA : 'temp21'
  ATRIBUTOS :
    +tipo : 'constEnt'
    +despl : 1
* LEXEMA : 'temp31'
  ATRIBUTOS :
    +tipo : 'constEnt'
    +despl : 2
* LEXEMA : 'temp41'
  ATRIBUTOS :
    +tipo : 'constEnt'
    +despl : 3
* LEXEMA : 'temp51'
  ATRIBUTOS :
    +tipo : 'booleanR'
    +despl : 4
```

En este ejemplo realizamos llamadas recursivas a una función para probar el correcto funcionamiento de la recursividad.

A.5. Prueba Funcional 5

Código fuente:

```
let  booleano boolean;
function bisiesto boolean (int a)
{ let bis string;
  print "Es bisiesto?";
  input bis;
  return ((a + 4 == 0));
}
function dias int (int m, int a)
{
  switch (m)
  {
    case 1: case 3: case 5: case 7: case 8: case 10: case 12:
```

```
      return 31; break;
    case 4: case 6: case 9: case 11:
      return 30;
    case 2: if (bisiesto (a)) return 29;
      return(28);

  }
}
function esFechaCorrecta boolean (int d, int m, int a)
{
  return (d == dias (m, a));
}
function demo ()
{

  if (esFechaCorrecta(31, 08, 2022)) print 9999;
  print("   se ha comprobado      la fecha");
  return;
}
let abc int;
demo();
```

Fichero de lenguaje intermedio generado:

```
: Etbisiesto null null
:= "" null 1.2
:= "Es bisiesto?" null 1.3
print 1.3 null null
input 1.2 null null
:= 4 null 1.4
:= 1.4 null 1.5
+ 1.1 1.5 1.6
:= 0 null 1.7
if== 1.6 1.7 goto 2
:= 0 null 1.8
goto null null 1
:= 1 null 1.8
return 1.8 null null
return null null null
: Etdias null null
if== 2.3 1 2
if!= 2.1 1 et02
:= 1 null 2.3
: et02 null null
if== 2.3 1 2
if!= 2.1 3 et03
:= 1 null 2.3
: et03 null null
if== 2.3 1 2
if!= 2.1 5 et04
:= 1 null 2.3
: et04 null null
if== 2.3 1 2
if!= 2.1 7 et05
:= 1 null 2.3
: et05 null null
if== 2.3 1 2
if!= 2.1 8 et06
:= 1 null 2.3
: et06 null null
if== 2.3 1 2
if!= 2.1 10 et07
:= 1 null 2.3
```

```
: et07 null null
if== 2.3 1 2
if!= 2.1 12 et08
:= 31 null 2.4
return 2.4 null null
goto et01 null null
:= 1 null 2.3
: et08 null null
if== 2.3 1 2
if!= 2.1 4 et09
:= 1 null 2.3
: et09 null null
if== 2.3 1 2
if!= 2.1 6 et010
:= 1 null 2.3
: et010 null null
if== 2.3 1 2
if!= 2.1 9 et011
:= 1 null 2.3
: et011 null null
if== 2.3 1 2
if!= 2.1 11 et012
:= 30 null 2.5
return 2.5 null null
:= 1 null 2.3
: et012 null null
if== 2.3 1 2
if!= 2.1 2 et013
param 2.2 null null
call Etbisiesto null 2.6
if 2.6 null et014
:= 29 null 2.7
return 2.7 null null
: et014 null null
:= 28 null 2.8
return 2.8 null null
:= 1 null 2.3
: et013 null null
: et01 null null
return null null null
: EtesFechaCorrecta null null
param 3.2 null null
param 3.3 null null
call Etdias null 3.4
if== 3.1 3.4 goto 2
:= 0 null 3.5
goto null null 1
:= 1 null 3.5
return 3.5 null null
return null null null
: Etdemo null null
:= 31 null 4.1
:= 8 null 4.2
:= 2022 null 4.3
param 4.1 null null
param 4.2 null null
param 4.3 null null
call EtesFechaCorrecta null 4.4
if 4.4 null et015
:= 9999 null 4.5
print 4.5 null null
: et015 null null
```

```
:= "   se ha comprobado      la fecha" null 4.6
print 4.6 null null
return null null null
return null null null
call Etdemo null null
```

Fichero de lenguaje objeto generado:

```
MOVE #inicio_estaticas, .IY
MOVE #inicio_pila, .IX
BR /main

main:
    MOVE #dir_ret_4, [.IX]
    BR /Etdemo
dir_ret_4:  NOP
    MOVE #inicio_pila, .IX
    HALT

Etbisiesto: NOP
    MOVE #data01, #2[.IX]
    MOVE #data02, #3[.IX]
    ADD #0, #3[.IX]
    WRSTR [.A]
    INSTR /data03
    MOVE #data03, #2[.IX]
    MOVE #4, #4[.IX]
    MOVE #4[.IX], #5[.IX]
    ADD #1[.IX], #5[.IX]
    MOVE .A, #6[.IX]
    MOVE #0, #7[.IX]
    CMP #6[.IX],#7[.IX]
    BZ $5
    MOVE #0, #8[.IX]
    BR $3
    MOVE #1, #8[.IX]
    SUB #Tam_RA_Etbisiesto, #1
    ADD .A, .IX;
    MOVE #8[.IX], [.A]
    BR [.IX]

    BR [.IX]

EtesFechaCorrecta:  NOP
    ADD #Tam_RA_EtesFechaCorrecta, .IX
    ADD #1, .A
    MOVE #2[.IX], [.A]
    ADD #Tam_RA_EtesFechaCorrecta, .IX
    ADD #2, .A
    MOVE #3[.IX], [.A]
    ADD #Tam_RA_EtesFechaCorrecta, .IX
    MOVE #dir_ret_2, [.A]
    MOVE .A, .IX
    BR /Etdias
dir_ret_2:  NOP
    SUB #Tam_RA_Etdias, #1
    ADD .A, .IX
    MOVE [.A], .R9
    SUB .IX, #Tam_RA_EtesFechaCorrecta
    MOVE .A, .IX
    MOVE .R9, #4[.IX]
    CMP #1[.IX],#4[.IX]
    BZ $5
```

```
    MOVE #0, #5[.IX]
    BR $3
    MOVE #1, #5[.IX]
    SUB #Tam_RA_EtesFechaCorrecta, #1
    ADD .A, .IX;
    MOVE #5[.IX], [.A]
    BR [.IX]

    BR [.IX]

Etdias: NOP
    CMP #3[.IX],#1
    BZ $5
    CMP #1[.IX], #1
    BNZ /et02
    MOVE #1, #3[.IX]
et02: NOP
    CMP #3[.IX],#1
    BZ $5
    CMP #1[.IX], #3
    BNZ /et03
    MOVE #1, #3[.IX]
et03: NOP
    CMP #3[.IX],#1
    BZ $5
    CMP #1[.IX], #5
    BNZ /et04
    MOVE #1, #3[.IX]
et04: NOP
    CMP #3[.IX],#1
    BZ $5
    CMP #1[.IX], #7
    BNZ /et05
    MOVE #1, #3[.IX]
et05: NOP
    CMP #3[.IX],#1
    BZ $5
    CMP #1[.IX], #8
    BNZ /et06
    MOVE #1, #3[.IX]
et06: NOP
    CMP #3[.IX],#1
    BZ $5
    CMP #1[.IX], #10
    BNZ /et07
    MOVE #1, #3[.IX]
et07: NOP
    CMP #3[.IX],#1
    BZ $5
    CMP #1[.IX], #12
    BNZ /et08
    MOVE #31, #4[.IX]
    SUB #Tam_RA_Etdias, #1
    ADD .A, .IX;
    MOVE #4[.IX], [.A]
    BR [.IX]
    BR /et01
    MOVE #1, #3[.IX]
et08: NOP
    CMP #3[.IX],#1
    BZ $5
    CMP #1[.IX], #4
```

```
    BNZ /et09
    MOVE #1, #3[.IX]
et09: NOP
    CMP #3[.IX],#1
    BZ $5
    CMP #1[.IX], #6
    BNZ /et010
    MOVE #1, #3[.IX]
et010:  NOP
    CMP #3[.IX],#1
    BZ $5
    CMP #1[.IX], #9
    BNZ /et011
    MOVE #1, #3[.IX]
et011:  NOP
    CMP #3[.IX],#1
    BZ $5
    CMP #1[.IX], #11
    BNZ /et012
    MOVE #30, #5[.IX]
    SUB #Tam_RA_Etdias, #1
    ADD .A, .IX;
    MOVE #5[.IX], [.A]
    BR [.IX]
    MOVE #1, #3[.IX]
et012:  NOP
    CMP #3[.IX],#1
    BZ $5
    CMP #1[.IX], #2
    BNZ /et013
    ADD #Tam_RA_Etdias, .IX
    ADD #1, .A
    MOVE #2[.IX], [.A]
    ADD #Tam_RA_Etdias, .IX
    MOVE #dir_ret_1, [.A]
    MOVE .A, .IX
    BR /Etbisiesto
dir_ret_1:  NOP
    SUB #Tam_RA_Etbisiesto, #1
    ADD .A, .IX
    MOVE [.A], .R9
    SUB .IX, #Tam_RA_Etdias
    MOVE .A, .IX
    MOVE .R9, #6[.IX]
    CMP #6[.IX], #1
    BNZ /et014
    MOVE #29, #7[.IX]
    SUB #Tam_RA_Etdias, #1
    ADD .A, .IX;
    MOVE #7[.IX], [.A]
    BR [.IX]
et014:  NOP
    MOVE #28, #8[.IX]
    SUB #Tam_RA_Etdias, #1
    ADD .A, .IX;
    MOVE #8[.IX], [.A]
    BR [.IX]
    MOVE #1, #3[.IX]
et013:  NOP
et01: NOP

    BR [.IX]
```

```
180
181 Etdemo: NOP
182     MOVE #31, #1[.IX]
183     MOVE #8, #2[.IX]
184     MOVE #2022, #3[.IX]
185     ADD #Tam_RA_Etdemo, .IX
186     ADD #1, .A
187     MOVE #1[.IX], [.A]
188     ADD #Tam_RA_Etdemo, .IX
189     ADD #2, .A
190     MOVE #2[.IX], [.A]
191     ADD #Tam_RA_Etdemo, .IX
192     ADD #3, .A
193     MOVE #3[.IX], [.A]
194     ADD #Tam_RA_Etdemo, .IX
195     MOVE #dir_ret_3, [.A]
196     MOVE .A, .IX
197     BR /EtesFechaCorrecta
198 dir_ret_3:  NOP
199     SUB #Tam_RA_EtesFechaCorrecta, #1
200     ADD .A, .IX
201     MOVE [.A], .R9
202     SUB .IX, #Tam_RA_Etdemo
203     MOVE .A, .IX
204     MOVE .R9, #4[.IX]
205     CMP #4[.IX], #1
206     BNZ /et015
207     MOVE #9999, #5[.IX]
208     WRINT #5[.IX]
209 et015:  NOP
210     MOVE #data04, #6[.IX]
211     ADD #0, #6[.IX]
212     WRSTR [.A]
213     BR [.IX]
214
215     BR [.IX]
216
217 Tam_RA_Etbisiesto: EQU 10
218 Tam_RA_Etdias: EQU 10
219 Tam_RA_EtesFechaCorrecta: EQU 7
220 Tam_RA_Etdemo: EQU 8
221 inicio_estaticas: RES 6
222
223 data01: DATA " "
224 data02: DATA "Es bisiesto? "
225 data03: RES 64
226 data04: DATA " se ha comprobado la fecha "
227
228 inicio_pila: NOP
229
230 END
231 ; una obra de diriG studios(c) 2023.
```

Fichero de Tabla de simbolos generado:

```
1 #1:
2 * LEXEMA : 'a'
3   ATRIBUTOS :
4     +tipo : 'constEnt'
5     +despl : 0
6 * LEXEMA : 'bis'
7   ATRIBUTOS :
8     +tipo : 'cadena'
```

```
    +despl : 1
* LEXEMA : 'temp21'
  ATRIBUTOS :
    +tipo : 'cadena'
    +despl : 2
* LEXEMA : 'temp31'
  ATRIBUTOS :
    +tipo : 'constEnt'
    +despl : 3
* LEXEMA : 'temp41'
  ATRIBUTOS :
    +tipo : 'constEnt'
    +despl : 4
* LEXEMA : 'temp51'
  ATRIBUTOS :
    +tipo : 'constEnt'
    +despl : 5
* LEXEMA : 'temp61'
  ATRIBUTOS :
    +tipo : 'constEnt'
    +despl : 6
* LEXEMA : 'temp71'
  ATRIBUTOS :
    +tipo : 'booleanR'
    +despl : 7
#2:
* LEXEMA : 'm'
  ATRIBUTOS :
    +tipo : 'constEnt'
    +despl : 0
* LEXEMA : 'a'
  ATRIBUTOS :
    +tipo : 'constEnt'
    +despl : 1
* LEXEMA : 'temp21'
  ATRIBUTOS :
    +tipo : 'constEnt'
    +despl : 2
* LEXEMA : 'temp31'
  ATRIBUTOS :
    +tipo : 'constEnt'
    +despl : 3
* LEXEMA : 'temp41'
  ATRIBUTOS :
    +tipo : 'constEnt'
    +despl : 4
* LEXEMA : 'temp51'
  ATRIBUTOS :
    +tipo : 'booleanR'
    +despl : 5
* LEXEMA : 'temp61'
  ATRIBUTOS :
    +tipo : 'constEnt'
    +despl : 6
* LEXEMA : 'temp71'
  ATRIBUTOS :
    +tipo : 'constEnt'
    +despl : 7
#3:
* LEXEMA : 'd'
  ATRIBUTOS :
    +tipo : 'constEnt'
```

```
    +despl : 0
* LEXEMA : 'm'
  ATRIBUTOS :
    +tipo : 'constEnt'
    +despl : 1
* LEXEMA : 'a'
  ATRIBUTOS :
    +tipo : 'constEnt'
    +despl : 2
* LEXEMA : 'temp31'
  ATRIBUTOS :
    +tipo : 'constEnt'
    +despl : 3
* LEXEMA : 'temp41'
  ATRIBUTOS :
    +tipo : 'booleanR'
    +despl : 4
#4:
* LEXEMA : 'temp01'
  ATRIBUTOS :
    +tipo : 'constEnt'
    +despl : 0
* LEXEMA : 'temp11'
  ATRIBUTOS :
    +tipo : 'constEnt'
    +despl : 1
* LEXEMA : 'temp21'
  ATRIBUTOS :
    +tipo : 'constEnt'
    +despl : 2
* LEXEMA : 'temp31'
  ATRIBUTOS :
    +tipo : 'booleanR'
    +despl : 3
* LEXEMA : 'temp41'
  ATRIBUTOS :
    +tipo : 'constEnt'
    +despl : 4
* LEXEMA : 'temp51'
  ATRIBUTOS :
    +tipo : 'cadena'
    +despl : 5
#0:
* LEXEMA : 'booleano'
  ATRIBUTOS :
    +tipo : 'booleanR'
    +despl : 0
* LEXEMA : 'bisiesto'
  ATRIBUTOS :
    +tipo : 'funcion'
     +numParam : 1
      +TipoParam1 : 'constEnt'
       +ModoParam1 : 1
    +TipoRetorno : 'booleanR'
    +EtiqFuncion : 'Etbisiesto01'
* LEXEMA : 'dias'
  ATRIBUTOS :
    +tipo : 'funcion'
     +numParam : 2
      +TipoParam1 : 'constEnt'
       +ModoParam1 : 1
      +TipoParam2 : 'constEnt'
```

```
      +ModoParam2 : 1
    +TipoRetorno : 'constEnt'
    +EtiqFuncion : 'Etdias01'
* LEXEMA : 'esFechaCorrecta'
  ATRIBUTOS :
    +tipo : 'funcion'
     +numParam : 3
      +TipoParam1 : 'constEnt'
       +ModoParam1 : 1
      +TipoParam2 : 'constEnt'
       +ModoParam2 : 1
      +TipoParam3 : 'constEnt'
       +ModoParam3 : 1
    +TipoRetorno : 'booleanR'
    +EtiqFuncion : 'EtesFechaCorrecta01'
* LEXEMA : 'demo'
  ATRIBUTOS :
    +tipo : 'funcion'
     +numParam : 0
    +TipoRetorno : 'vacio'
    +EtiqFuncion : 'Etdemo01'
* LEXEMA : 'abc'
  ATRIBUTOS :
    +tipo : 'constEnt'
    +despl : 1
```

En esta prueba se realiza todo tipo de operaciones, como sumas con llamadas a funciones, comparaciones con llamadas a funciones, switch sin break, anidamiento de funciones, entre otras cosas.