
Traductores de Lenguajes

MEMORIA FINAL

Grupo 55

Daniel Tomás Sánchez
Aarón Cabero Blanco
Alejandro Cuadrón Lafuente

Curso 2020/2021

Índice

1	Introducción	3
2	Diseño Analizador Léxico	4
2.1	Tokens	4
2.2	Gramática Regular	5
2.3	Autómata Finito Determinista	5
2.4	Acciones Semánticas	6
2.5	Errores	7
3	Diseño Analizador Sintáctico	8
3.1	Gramática	8
3.2	Tabla LR(1)	9
4	Diseño Analizador Semántico	12
5	Diseño Tabla de Símbolos	17
6	Anexo	18
6.1	Casos de prueba correctos	18
6.1.1	Prueba correcta 1	18
6.1.2	Prueba correcta 2	21
6.1.3	Prueba correcta 3	23
6.1.4	Prueba correcta 4	25
6.1.5	Prueba correcta 5	28
6.2	Casos de prueba erróneos	32
6.2.1	Prueba errónea 1	32
6.2.2	Prueba errónea 2	32
6.2.3	Prueba errónea 3	32
6.2.4	Prueba errónea 4	32
6.2.5	Prueba errónea 5	33
7	Diseño del generador de código intermedio	34
8	Diseño del registro de activación	38
9	Diseño del código objeto	39
9.1	Métodos auxiliares	39
9.2	Aritméticas y lógicas	40
9.3	Asignaciones	40
9.4	Salto y etiquetas	41

9.5	Paso de parámetros	41
9.6	Llamadas	42
9.7	Sentencias de retorno	43
9.8	Alerts e inputs	43
10	Anexo	45
10.1	Prueba 1 - test del for	45
10.2	Prueba 2 - test de operaciones entrada y salida	60
10.3	Prueba 3 - test genérico	70
10.4	Prueba 4 - test recursivo	76
10.5	Prueba 5 - test llamadas a funciones	83
11	Referencias	95

1 | Introducción

Hemos decidido usar, como lenguaje de programación, Python, ya que nos apetecía aprender un lenguaje nuevo, además de que es muy usado en la industria.

El trabajo completo, tanto el léxico, como el sintáctico y el semántico, ha sido realizado con la herramienta o librería externa "SLY"[1].

Opciones de grupo:

- Sentencias: Sentencia repetitiva (**for**)
- Operadores especiales: Post-auto-decremento (**-- como sufijo**)
- Técnicas de Análisis Sintáctico: **Ascendente**
- Comentarios: Comentario de bloque (**/ * */**)
- Cadenas: Con comillas dobles (**" "**)

2 | Diseño Analizador Léxico

2.1 Tokens

▪ Identificador	<ID, punteroTS>
▪ Constante entera	<CTEENTERA, valor>
▪ Cadena de caracteres	<CADENA, lexema>
▪ false	<CTELOGICA, 0>
▪ true	<CTELOGICA, 1>
▪ Palabra reservada Number	<NUMBER, ->
▪ Palabra reservada String	<STRING, ->
▪ Palabra reservada Boolean	<BOOLEAN, ->
▪ Palabra reservada Let	<LET, ->
▪ Palabra reservada Alert	<ALERT, ->
▪ Palabra reservada Input	<INPUT, ->
▪ Palabra reservada Function	<FUNCTION, ->
▪ Palabra reservada Return	<RETURN, ->
▪ Palabra reservada If	<IF, ->
▪ Palabra reservada For	<FOR, ->
▪ --	<OPESP, ->
▪ -	<OPARIT, ->
▪ =	<OPASIG, ->
▪ ==	<OPREL, ->
▪ &&	<OPLOG, ->
▪ (<ABPAREN, ->
▪)	<CEAPAREN, ->
▪ {	<ABLLAVE, ->
▪ }	<CELLAVE, ->
▪ ,	<COMA, ->
▪ ;	<PUNTOYCOMA, ->

2.2 Gramática Regular

Axioma = A

$A \rightarrow \text{del } A \mid d D \mid " S \mid / C \mid l I \mid -M \mid = E \mid \& N \mid (\mid) \mid \{ \mid \} \mid ; \mid ,$

$D \rightarrow d D \mid \lambda$

$S \rightarrow " \mid c S$

$C \rightarrow * C'$

$C' \rightarrow * C'' \mid c C'$

$C'' \rightarrow / A \mid c C'$

$I \rightarrow d I \mid l I \mid _ I \mid \lambda$

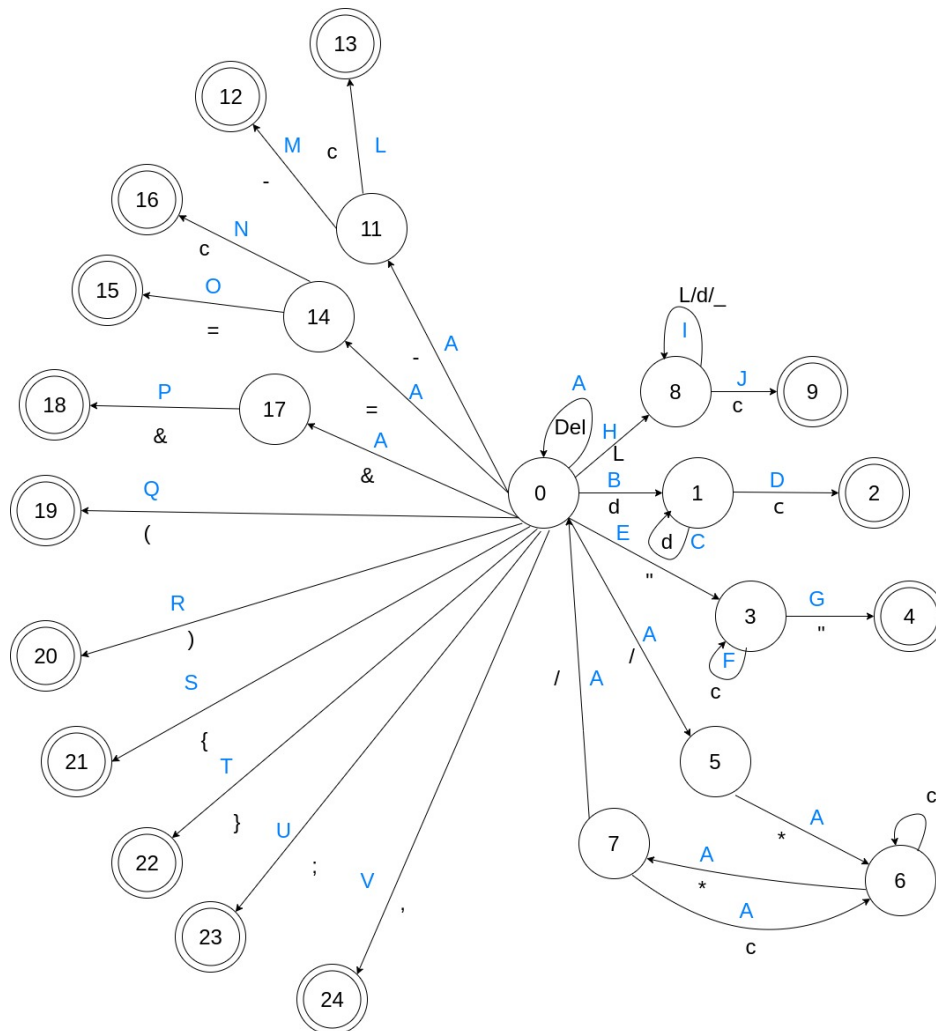
$M \rightarrow - \mid \lambda$

$E \rightarrow = \mid \lambda$

$N \rightarrow \&$

Siendo d un dígito, l una letra, c cualquier otro carácter y del un delimitador.

2.3 Autómata Finito Determinista



2.4 Acciones Semánticas

```
A: leer
B: number = int(d), leer
C: number = number * 10 + int(d), leer
D: if number > 32767
    pError("Número fuera de rango")
    else
        genToken(CTEENTERA, number);
E: string = "", contador = 0, leer
F: string = string + otroCS, contador++, leer
G: if contador > 64
    pError("Cadena demasiado larga")
    else
        genToken(CADENA, string)
    leer
H: string = l, leer
I: string = string + l/D/_ , leer
J: if palabrasReservadas.contains(string)
    if string == "number"
        genToken(NUMBER, -)
    elif string == "string"
        genToken(STRING, -)
    elif string == "boolean"
        genToken(BOOLEAN, -)
    elif string == "let"
        genToken(LET, -)
    elif string == "alert"
        genToken(ALERT, -)
    elif string == "input"
        genToken(INPUT, -)
    elif string == "return"
        genToken(RETURN, -)
    elif string == "if"
        genToken(IF, -)
    else
        genToken(FOR, -)
```

```

else    // palabrasReservadas.contains(string) = False
    puntero = TS.get(string)
    if zona_decl == True
        if puntero != None
            pError("Identificador ya declarado")
        else
            TS.update(string)
            puntero = TS.get(string)
            genToken(ID, puntero)
    else
        if puntero == None
            TS.update(string)
            puntero = TS.get(string)
            genToken(ID, puntero)
        else
            genToken(ID, puntero)

L: genToken(OPARIT, -)
M: genToken(OPESP, -), leer
N: genToken(OPASIG, -)
O: genToken(OPREL, -), leer
P: genToken(OPLOG, -), leer
Q: genToken(ABPAREN, -), leer
R: genToken(CEPAREN, -), leer
S: genToken(ABLLAVE, -), leer
T: genToken(CELLAVE, -), leer
U: genToken(COMA, -), leer
V: genToken(PUNTOYCOMA, -), leer
W: genToken(EOF, -), leer

```

2.5 Errores

Error léxico (siempre se lanza cuando el analizador léxico encuentra un error).

1. Cadena con longitud mayor de 64 caracteres.
2. Número fuera de rango (mayor de 32767).
3. Identificador ya declarado.
4. Carácter ilegal.

Todo error va acompañado de la *línea* y *columna* en el que se ha encontrado dicho error.

3 | Diseño Analizador Sintáctico

3.1 Gramática

Axioma = B

No Terminales = { A B C D E F G H I J K L M N O P Q R S T U V W F1 F2 F3 }

Terminales = { && == - -- () = , ; id ent cad log let alert input return for if number
boolean string function }

Producciones = {

B → D

D → F D

D → G D

D → λ

G → if (E) S

G → S

S → H ;

H → id (I)

I → E J

I → λ

J → , E J

J → λ

S → K ;

K → id = E

S → alert (E) ;

S → input (id) ;

S → return L ;

L → E

L → λ

G → let M T id ;

M → λ

T → number

T → boolean

T → string

G → for (N ; E ; O) C

N → K

N → λ

O → K

O → -- id

O → λ

C → G C

C → λ

F → F1 F2 F3

F1 → function P Q id

P → λ

Q → T

Q → λ

F2 → (A)

A → T id AA

A → λ

AA → , T id AA

AA → λ

F3 → C

E → E && R

E → R

R → R == U

R → U

U → U - V

U → V

V → -- id

V → id

V → (E)

V → H

V → ent

V → cad

V → log

}

3.2 Tabla LR(1)

[illegible]

10

[illegible]

4 | Diseño Analizador Semántico

Acción semántica previa a empezar a funcionar el procesador:

```
{ TS_g = crear_TS()
  TS_actual = TS_g
  desp = 0
  zona_decl = false }
```

```
B → D {}
D → F D {}
D → G D {}
D → lambda {}
```

```
G → if ( E ) S { if E.tipo != "lógico"
                  then error(1) }
G → S {}
S → H ; {}
H → id ( I ) { if busca_tipo_TS(id.pos) != "función"
                then error(15)
                else if longitud(I.tipo) != busca_num_params_TS(id.pos)
                  then error(2)
                else if busca_tipo_TS(id.pos) != I.tipo
                  then error(3)
                else
                  H.tipo = busca_tipo_devuelto_TS(id.pos) }
I → E J { if longitud(J.tipo) == 0
          then I.tipo = E.tipo
          else
            I.tipo = E.tipo x J.tipo }
J → , E J1 { if longitud(J1.tipo) == 0
              then J.tipo = E.tipo
              else
                J.tipo = E.tipo x J1.tipo }
J → lambda {}
I → lambda { I.tipo = void }

S → K ; {}
K → id = E { if busca_tipo_TS(id.pos) != None
              añade_tipo_TS(id.pos, "entero")
              añade_desp_TS(id.pos, desp)
              desp+=2
              if busca_tipo_TS(id.pos) != E.tipo
                then error(10) }
```

```

S → alert ( E ) ; { if E.tipo != "cadena" && E.tipo != "entero"
                    then error(4) }
S → input ( id ) ; { if busca_tipo_TS(id.pos) != None
                    añade_tipo_TS(id.pos, "entero")
                    añade_desp_TS(id.pos, desp)
                    desp+=2
                    if busca_tipo_TS(id.pos) != "cadena"
                      && busca_tipo_TS(id.pos) != "entero"
                      then error(5) }
S → return L ; { if zona_function != true
                then error(7)}
                else if L.tipo != tipo_return
                then error(9) }
L → E { L.tipo = E.tipo }
L → lambda { L.tipo = void }

```

```

G → let M T id ; { añadir_tipo_TS(id.pos, T.tipo)
                  añadir_desp_TS(id.pos, desp)
                  desp += T.ancho
                  zona_declaración = false }
M → lambda { zona_declaracion = true }
T → number { T.tipo = "entero"
            T.ancho = 2} // size_of("entero")
T → boolean { T.tipo = "lógico"
             T.ancho = 2} // size_of("lógico")
T → string { T.tipo = "cadena"
            T.ancho = 128} // size_of("cadena")

```

```

G → for ( N ; E ; O ) { C } { if E.tipo != "lógico"
                              then error(6) }
N → K {}
N → lambda {}
O → K {}
O → -- id { if busca_tipo_TS(id.pos) != None
            añade_tipo_TS(id.pos, "entero")
            añade_desp_TS(id.pos, desp)
            desp+=2
            if busca_tipo_TS(id.pos) != "entero"
              then error(11) }
O → lambda {}
C → G C {}
C → lambda {}

```

```

F → F1 F2 F3 { destruye_TS (TS_l)
                zona_function = false
                TS_actual = TS_g
                desp = desp_g
                tipo_return = NULL }
F1 → function P Q id { TS_l = crear_TS()
                      TS_actual = TS_l
                      desp_g = desp
                      desp = 0
                      pos_id_fun = id.pos
                      zona_function = true
                      tipo_return = Q.tipo
                      añadir_tipo_devuelto_TS(id.pos, tipo_return)
                      añadir_tipo_TS(id.pos, "función")
                      inserta_et_TS (id.pos, nueva_et ()) }
P → lambda {}
Q → T { zona_declaracion = true
        Q.tipo = T.tipo }
Q → lambda { zona_declaracion = true
            Q.tipo = void }

F2 → ( A ) { añadir_param_TS(pos_id_fun, A.tipo, longitud(A.tipo))
            zona_declaración = false }
A → T id AA { añadir_tipo_TS(id.pos, T.tipo)
              añadir_desp_TS(id.pos, desp)
              desp += T.ancho
              if longitud(AA.tipo) == 0
                then A.tipo = T.tipo
              else
                A.tipo = T.tipo x AA.tipo }
A → lambda { A.tipo = void }
AA → , T id AA1 { añadir_tipo_TS(id.pos, T.tipo)
                  añadir_desp_TS(id.pos, desp)
                  desp += T.ancho
                  if longitud(AA1.tipo) == 0
                    then AA.tipo = T.tipo
                  else
                    AA.tipo = T.tipo x AA1.tipo }
AA → lambda {}

F3 → { C } {}

E → E1 && R { if E1.tipo != "lógico" || R.tipo != "lógico"
              then error(12)
              else
                E.tipo = "lógico" }
E → R {E.tipo = R.tipo }

```

```

R → R1 == U { if R1.tipo != "entero" || U.tipo != "entero"
               then error(13)
               else
                 E.tipo = "lógico" }
R → U { R.tipo = U.tipo }
U → U1 - V { if U1.tipo != "entero" || V.tipo != "entero"
              then error(14)
              else
                U.tipo = "entero" }
U → V { U.tipo = V.tipo }
V → -- id { if busca_tipo_TS(id.pos) != None
             añade_tipo_TS(id.pos, "entero")
             añade_desp_TS(id.pos, desp)
             desp+=2
             if busca_tipo_TS(id.pos) != "entero"
             then error(15)
             else
               V.tipo = "entero" }
V → id { if busca_tipo_TS(id.pos) != None
          añade_tipo_TS(id.pos, "entero")
          añade_desp_TS(id.pos, desp)
          desp+=2
          V.tipo = busca_tipo_TS(id.pos) }
V → ( E ) { V.tipo = E.tipo }
V → H { V.tipo = H.tipo }
V → ent { V.tipo = "entero" }
V → cad { V.tipo = "cadena" }
V → log { V.tipo = "lógico" }

error_code_dict = {
  1: "La condición debe ser un lógico",
  2: "El número de parámetros introducidos no son los esperados, \
      deberían ser {busca_num_params_TS(id.pos)}",
  3: "El tipo de los parámetros no es el esperado, \
      se esperaban {busca_tipo_params_TS(id.pos)}",
  4: "La expresión introducida no es una cadena o un entero",
  5: "La variable introducida no es de tipo cadena o entero",
  6: "La condición debe ser un lógico",
  7: "No puede haber una sentencia return fuera de una función",
  8: "No se permite la definición de funciones anidadas",
  9: "El tipo de retorno no corresponde con el tipo de retorno de la función, \
      se esperaba {tipo_return}",
  10: "El tipo de la variable a asignar no corresponde con el tipo asignado",
  11: "El operador especial "--" solo trabaja con tipos de datos enteros",
  12: "El operador lógico "&&" solo trabaja con tipos de datos lógicos",
  13: "El operador de relación "==" solo trabaja con tipos de datos enteros",
  14: "El operador aritmético "-" solo trabaja con tipos de datos enteros",
  15: "La variable no se puede invocar como una función"
}

```



```
function error(error_code):  
    res = ""  
    if error_code <= 6:  
        res = "ErrorDeAtributo: "  
    elif error_code >= 9:  
        res = "ErrorDeTipado: "  
    else  
        res = "NoImplementado: "  
    res = res + error_code_dict.get(error_code, default="Código no válido")  
    print(res)
```

5 | Diseño Tabla de Símbolos

La TS se compone de una lista de tablas, una de ellas es global y se crea al empezar, mientras que el resto son locales y se van creando a medida que avanza la compilación. Están ordenadas en orden de creación. Cada tabla tiene un flag que indica si la tabla existe (está activa) o se ha eliminado, pero realmente no las eliminamos para posteriormente imprimirlas.

A su vez cada tabla contiene una lista de diccionarios, cada diccionario simboliza una entrada en la tabla de símbolos. Un diccionario es un hashmap que tiene como claves la palabra "lexema" y los tipos de atributos que le corresponda (por ejemplo "Tipo" ó "Despl"), como valores tiene el valor del lexema y de sus atributos en si mismos (por ejemplo "main" ó "entero").

6 | Anexo

6.1 Casos de prueba correctos

6.1.1 Prueba correcta 1

Programa introducido:

```
1 let string texto;  
2 let string textoAux;  
3 textoAux = texto;  
4 alert  
5     (textoAux);
```

Tokens:

<FUNCTION , >
<NUMBER , >
<ID , 0>
<ABPAREN , >
<NUMBER , >
<ID , 0>
<CEPAREN , >
<ABLLAVE , >
<IF , >
<ABPAREN , >
<ID , 0>
<OPREL , >
<CTEENTERA , 0>
<CEPAREN , >
<RETURN , >
<CTEENTERA , 1>
<PUNTOYCOMA , >
<RETURN , >
<ID , 0>
<OPARIT , >
<ID , 0>
<ABPAREN , >
<ID , 0>
<OPARIT , >
<CTEENTERA , 1>
<CEPAREN , >
<PUNTOYCOMA , >
<CELLAVE , >

Tabla de símbolos:

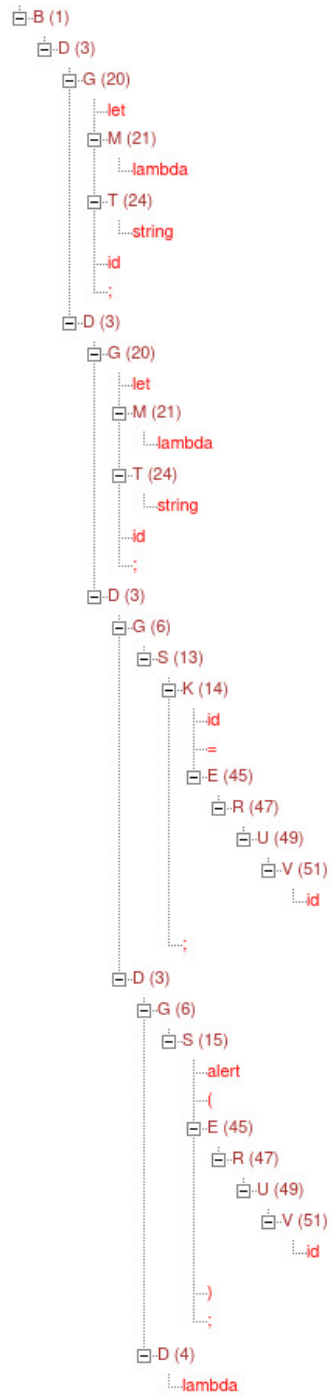
CONTENIDO DE LA TABLA 0 :

* LEXEMA : 'texto'
ATRIBUTOS :
+ Tipo : 'cadena'
+ Despl : '0'

* LEXEMA : 'textoAux'
ATRIBUTOS :

+ Tipo : 'cadena'
+ Despl : '64'

VAST:



6.1.2 Prueba correcta 2

Programa introducido:

```
1 function number Factorial (number n)
2 {
3     if (n == 0) return 1;
4     return n - Factorial (n - 1);
5 }
```

Tokens:

<FUNCTION , >
<NUMBER , >
<ID , 0>
<ABPAREN , >
<NUMBER , >
<ID , 0>
<CEPAREN , >
<ABLLAVE , >
<IF , >
<ABPAREN , >
<ID , 0>
<OPREL , >
<CTEENTERA , 0>
<CEPAREN , >
<RETURN , >
<CTEENTERA , 1>
<PUNTOYCOMA , >
<RETURN , >
<ID , 0>
<OPARIT , >
<ID , 0>
<ABPAREN , >
<ID , 0>
<OPARIT , >
<CTEENTERA , 1>
<CEPAREN , >
<PUNTOYCOMA , >
<CELLAVE , >

Tabla de símbolos:

CONTENIDO DE LA TABLA 0 :

* LEXEMA : 'Factorial'
ATRIBUTOS :
+ Tipo : 'funcion'
+ TipoRetorno : 'entero'
+ EtiqFuncion : 'Et_Fun_0'
+ numParam : '1'
+ TipoParam1 : 'entero'

```
*      LEXEMA : 'n'
      ATRIBUTOS :
      + Tipo : 'entero'
      + Despl : '0'
```

```

graph TD
    B1["B (1)"] --> D2["D (2)"]
    D2 --> F33["F (33)"]
    F33 --> F134["F1 (34)"]
    F134 --> function["function"]
    F134 --> P35["P (35)"]
    P35 --> lambda1["lambda"]
    lambda1 --> Q36["Q (36)"]
    Q36 --> T22["T (22)"]
    T22 --> number["number"]
    T22 --> id1["id"]
    T22 --> F238["F2 (38)"]
    F238 --> A39["A (39)"]
    A39 --> T22_2["T (22)"]
    T22_2 --> number_2["number"]
    T22_2 --> id2["id"]
    T22_2 --> AA42["AA (42)"]
    AA42 --> lambda2["lambda"]
    F238 --> F343["F3 (43)"]
    F343 --> C31["C (31)"]
    C31 --> G5["G (5)"]
    G5 --> if["if"]
    if --> E45["E (45)"]
    E45 --> R46["R (46)"]
    R46 --> R47["R (47)"]
    R47 --> U49["U (49)"]
    U49 --> V51["V (51)"]
    V51 --> id3["id"]
    if --> E45_2["E (45)"]
    E45_2 --> R46_2["R (46)"]
    R46_2 --> U49_2["U (49)"]
    U49_2 --> V54["V (54)"]
    V54 --> ent1["ent"]
    if --> S17["S (17)"]
    S17 --> return["return"]
    S17 --> L18["L (18)"]
    L18 --> E45_3["E (45)"]
    E45_3 --> R47_2["R (47)"]
    R47_2 --> U49_3["U (49)"]
    U49_3 --> V54_2["V (54)"]
    V54_2 --> ent2["ent"]
    S17 --> C31_2["C (31)"]
    C31_2 --> G6["G (6)"]
    G6 --> S17_2["S (17)"]
    S17_2 --> return_2["return"]
    S17_2 --> L18_2["L (18)"]
    L18_2 --> E45_4["E (45)"]
    E45_4 --> R47_3["R (47)"]
    R47_3 --> U48["U (48)"]
    U48 --> U49_4["U (49)"]
    U49_4 --> V51_2["V (51)"]
    V51_2 --> id4["id"]
    S17_2 --> V53["V (53)"]
    V53 --> H8["H (8)"]
    H8 --> id5["id"]
    S17_2 --> I9["I (9)"]
    I9 --> E45_5["E (45)"]
    E45_5 --> R47_4["R (47)"]
    R47_4 --> U48_2["U (48)"]
    U48_2 --> U49_5["U (49)"]
    U49_5 --> V51_3["V (51)"]
    V51_3 --> id6["id"]
    I9 --> V54_3["V (54)"]
    V54_3 --> ent3["ent"]
    I9 --> J11["J (11)"]
    J11 --> lambda3["lambda"]
    S17_2 --> C32["C (32)"]
    C32 --> lambda4["lambda"]
    S17_2 --> D4["D (4)"]
    D4 --> lambda5["lambda"]

```

6.1.3 Prueba correcta 3

Programa introducido:

```
1 let boolean z;  
2 for (b=1;z; )  
3 {  
4 alert (88);  
5 }
```

Tokens:

<LET , >
<BOOLEAN , >
<ID , 0>
<PUNTOYCOMA , >
<FOR , >
<ABPAREN , >
<ID , 1>
<OPASIG , >
<CTEENTERA , 1>
<PUNTOYCOMA , >
<ID , 0>
<PUNTOYCOMA , >
<CEPAREN , >
<ABLLAVE , >
<ALERT , >
<ABPAREN , >
<CTEENTERA , 88>
<CEPAREN , >
<PUNTOYCOMA , >
<CELLAVE , >

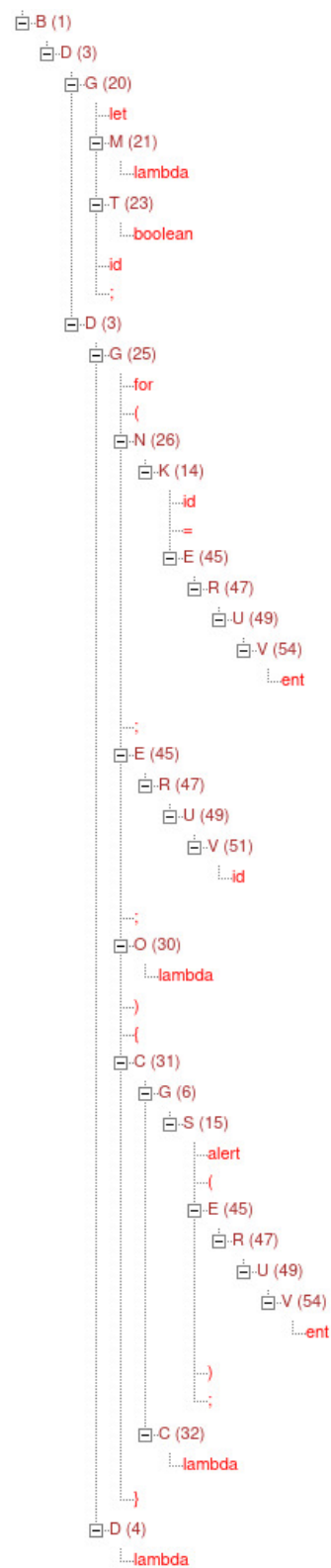
Tabla de símbolos:

CONTENIDO DE LA TABLA 0 :

* LEXEMA : 'z'
ATRIBUTOS :
+ Tipo : 'logico'
+ Despl : '0'

* LEXEMA : 'b'
ATRIBUTOS :
+ Tipo : 'entero'
+ Despl : '1'

VASt:



6.1.4 Prueba correcta 4

Programa introducido:

```
1  /* prueba
2      correcta */
3  let boolean b; let number x;
4  input (z);
5  alert (z);
6  x=z;
7  alert (z-1);
8  b=b&&b;if (b)
9  x =
10     x - 6
11     - z
12     - 1
13     - (2
14     - y
15     - 6);
```

Tokens:

```
<LET , >
<BOOLEAN , >
<ID , 0>
<PUNTOYCOMA , >
<LET , >
<NUMBER , >
<ID , 1>
<PUNTOYCOMA , >
<INPUT , >
<ABPAREN , >
<ID , 2>
<CEPAREN , >
<PUNTOYCOMA , >
<ALERT , >
<ABPAREN , >
<ID , 2>
<CEPAREN , >
<PUNTOYCOMA , >
<ID , 1>
<OPASIG , >
<ID , 2>
<PUNTOYCOMA , >
<ALERT , >
<ABPAREN , >
<ID , 2>
<OPARIT , >
<CTEENTERA , 1>
<CEPAREN , >
<PUNTOYCOMA , >
<ID , 0>
<OPASIG , >
<ID , 0>
<OPLOG , >
```

<ID , 0>
 <PUNTOYCOMA , >
 <IF , >
 <ABPAREN , >
 <ID , 0>
 <CEPAREN , >
 <ID , 1>
 <OPASIG , >
 <ID , 1>
 <OPARIT , >
 <CTEENTERA , 6>
 <OPARIT , >
 <ID , 2>
 <OPARIT , >
 <CTEENTERA , 1>
 <OPARIT , >
 <ABPAREN , >
 <CTEENTERA , 2>
 <OPARIT , >
 <ID , 3>
 <OPARIT , >
 <CTEENTERA , 6>
 <CEPAREN , >
 <PUNTOYCOMA , >

Tabla de símbolos:

CONTENIDO DE LA TABLA 0 :

* LEXEMA : 'b'
 ATRIBUTOS :
 + Tipo : 'logico'
 + Despl : '0'

* LEXEMA : 'x'
 ATRIBUTOS :
 + Tipo : 'entero'
 + Despl : '1'

* LEXEMA : 'z'
 ATRIBUTOS :
 + Tipo : 'entero'
 + Despl : '2'

* LEXEMA : 'y'
 ATRIBUTOS :
 + Tipo : 'entero'
 + Despl : '3'

VASt:

6.1.5 Prueba correcta 5

Programa introducido:

```
1 function FuncionSentencia (number b, boolean z)
2 {
3     for (b=0;true; --b )
4     {
5         alert ("hola");
6     }
7 }
8 function Funcion (number x, boolean b)
9 {
10     if ( x == 0 ) FuncionSentencia(x,b);
11     alert
12         (cadena); return;
13 }
```

Tokens:

<FUNCTION , >
<ID , 0>
<ABPAREN , >
<NUMBER , >
<ID , 0>
<COMA , >
<BOOLEAN , >
<ID , 1>
<CEPAREN , >
<ABLLAVE , >
<FOR , >
<ABPAREN , >
<ID , 0>
<OPASIG , >
<CTEENTERA , 0>
<PUNTOYCOMA , >
<CTELOGICA , 1>
<PUNTOYCOMA , >
<OPESP , >
<ID , 0>
<CEPAREN , >
<ABLLAVE , >
<ALERT , >
<ABPAREN , >
<CADENA , "hola">
<CEPAREN , >
<PUNTOYCOMA , >
<CELLAVE , >
<CELLAVE , >
<FUNCTION , >
<ID , 1>
<ABPAREN , >
<NUMBER , >
<ID , 0>
<COMA , >
<BOOLEAN , >

```

<ID , 1>
<CEPAREN , >
<ABLLAVE , >
<IF , >
<ABPAREN , >
<ID , 0>
<OPREL , >
<CTEENTERA , 0>
<CEPAREN , >
<ID , 0>
<ABPAREN , >
<ID , 0>
<COMA , >
<ID , 1>
<CEPAREN , >
<PUNTOYCOMA , >
<ALERT , >
<ABPAREN , >
<ID , 2>
<CEPAREN , >
<PUNTOYCOMA , >
<RETURN , >
<PUNTOYCOMA , >
<CELLAVE , >

```

Tabla de símbolos:

CONTENIDO DE LA TABLA 0 :

```

*      LEXEMA : 'FuncionSentencia'
      ATRIBUTOS :
      + Tipo : 'funcion'
      + TipoRetorno : 'void'
      + EtiqFuncion : 'Et_Fun_0'
      + numParam : '2'
      + TipoParam1 : 'entero'
      + TipoParam2 : 'logico'

```

```

-----
*      LEXEMA : 'Funcion'
      ATRIBUTOS :
      + Tipo : 'funcion'
      + TipoRetorno : 'void'
      + EtiqFuncion : 'Et_Fun_1'
      + numParam : '2'
      + TipoParam1 : 'entero'
      + TipoParam2 : 'logico'

```

```

-----
*      LEXEMA : 'cadena'
      ATRIBUTOS :
      + Tipo : 'entero'
      + Despl : '0'
-----

```

CONTENIDO DE LA TABLA 1 :

* LEXEMA : 'b'
ATRIBUTOS :
+ Tipo : 'entero'
+ Despl : '0'

* LEXEMA : 'z'
ATRIBUTOS :
+ Tipo : 'logico'
+ Despl : '1'

CONTENIDO DE LA TABLA 2 :

* LEXEMA : 'x'
ATRIBUTOS :
+ Tipo : 'entero'
+ Despl : '0'

* LEXEMA : 'b'
ATRIBUTOS :
+ Tipo : 'logico'
+ Despl : '1'

VASt:

6.2 Casos de prueba erróneos

6.2.1 Prueba errónea 1

Programa introducido:

```
1 let number id;  
2 id(2);
```

Mensaje de error:

Error en la línea 2:

La variable no se puede invocar como una función, con argumentos

6.2.2 Prueba errónea 2

Programa introducido:

```
1 let boolean id;  
2 res = --id ;
```

Mensaje de error:

Error en la línea 2:

El operador especial '--' solo trabaja con tipos de datos enteros

6.2.3 Prueba errónea 3

Programa introducido:

```
1 function Funcion (number a, boolean b, string c){}  
2 Funcion(a,a,a);
```

Mensaje de error:

Error en la línea 2:

El tipo de los parámetros no es el esperado, se esperaban "'entero', 'logico', 'cadena'"

6.2.4 Prueba errónea 4

Programa introducido:

```
1 let string texto;  
2 function pideTexto ()  
3 {  
4   alert ("Introduce un texto");  
5   input (texto);  
6 }  
7  
8   return;
```

Mensaje de error:

Error en la línea 8:

No puede haber una sentencia return fuera de una función

6.2.5 Prueba errónea 5

Programa introducido:

```
1 let boolean bool;  
2 alert(bool);
```

Mensaje de error:

Error en la línea 2:

La expresión introducida no es una cadena o un entero

7 | Diseño del generador de código intermedio

Para diseñar el generador de código intermedio nos basamos en la gramática empleada en la práctica de PDL. El diseño resultante es el siguiente:

Axioma = B

NoTerminales = { A AA B C D E F F1 F2 F3 G H I J K L M N O P Q R S T U V }

Terminales = { && == - -- () = ; , { } id ent cad log let alert input return for if
number boolean string function }

B → D {

 B.cod = D.cod

}

D → F D {

 D.cod = F.cod || D.cod

}

D → G D {

 D.cod = G.cod || D.cod

}

D → lambda {

 D.cod = vacio

}

G → if (E) S {

 G.desp = nuevaeti()

 G.cod = E.cod || gen(if, E.lugar, =, 0, goto, G.desp) || S.cod || gen(, G.desp)

}

G → S {

 G.cod = S.cod

}

S → H ; {

 S.cod = H.cod

}

H → id (I) {

 H.lugar = nuevatem()

 H.cod = I.codE || I.codP || gen(H.lugar, =, call, buscaEtiquetaTS(id.pos))

}

I → E J {

 I.codE = E.cod || J.codE

 I.codP = gen(param, E.lugar) || J.codP

}

J → , E J1 {

 J.codE = E.cod || J1.codE

 J.codP = gen(param, E.lugar) || J1.codP

}

J → lambda {

 J.codE = vacio

 J.codP = vacio

}

I → lambda {

 I.codE = vacio

```

    I.codP = vacio
  }

S → K ; {
  S.cod = K.cod
}
K → id = E {
  K.cod = E.cod || gen(buscaLugarTS(id.pos), =, E.lugar)
}

S → alert ( E ) ; {
  S.cod = E.cod || gen(alert, E.lugar)
}
S → input ( id ) ; {
  S.cod = gen(input, buscaLugarTS(id.pos))
}

S → return L ; {
  if( L.cod = vacio )
    S.cod = gen(return)
  else
    S.cod = L.cod || gen(return, L.lugar)
}
L → E {
  L.cod = E.cod
  L.lugar = E.lugar
}
L → lambda {
  L.cod = vacio
}

G → let M T id ; {
  G.cod = vacio
}
M → lambda { }
T → number { }
T → boolean { }
T → string { }

G → for ( N ; E ; O ) { C } {
  G.inicio = nuevaetiq()
  G.desp = nuevaetiq()
  G.cod = N.cod || gen(:, G.inicio) || E.cod || gen(if, E.lugar, =, 1, goto, G.desp) ||
    C.cod || O.cod || gen(goto, G.inicio) || gen(:, G.desp)
}
N → K {
  N.cod = K.cod
}
N → lambda {
  N.cod = vacio
}

```

```

O → K {
  O.cod = K.cod
}
O → -- id {
  O.lugar = nuevatemp()
  id.lugar = buscaLugarTS(id.pos)
  O.cod = gen(id.lugar, =, id.lugar, -, 1) || gen(O.lugar, =, id.lugar)
}
O → lambda {
  O.cod = vacio
}

C → G C1 {
  C.cod = G.cod || C1.cod
}
C → lambda {
  C.cod = vacio
}

F → F1 F2 F3 {
  F.cod = F1.cod || F2.cod || F3.cod || gen(return)
}
F1 → function P Q id {
  F1.cod = gen(:, buscaEtiquetaTS(id.pos))
}
P → lambda { }
Q → T { }
Q → lambda { }

F2 → ( A ) {
  F2.cod = vacio
}
A → T id AA { }
A → lambda { }
AA → , T id AA { }
AA → lambda { }

F3 → { C } {
  F3.cod = C.cod
}

E → E1 && R {
  E.lugar = nuevatemp()
  E.cod = E1.cod || R.cod || gen(E.lugar, =, E1.lugar, AND, R.lugar)
}
E → R {
  E.lugar = R.lugar
  E.cod = R.cod
}

R → R1 == U {
  R.true = nuevaetiq()
  R.lugar = nuevatemp()
  R.despues = nuevaetiq()
}

```

```

    R.cod = R1.cod || U.cod || gen(if, R1.lugar, =, U.lugar, goto, R.true) ||
      gen(R.lugar, =, 0) || gen(goto, R.después) ||
      gen(:, R.true) || gen(R.lugar, =, 1) || gen(:, R.despues)
  }
R → U {
  R.cod = U.cod
  R.lugar = U.lugar
}

U → U1 - V {
  U.lugar = nuevatemp()
  U.cod = U1.cod || V.cod || gen(U.lugar, =, U1.lugar, -, V.lugar)
}
U → V {
  U.lugar = V.lugar
  U.cod = V.cod
}

V → -- id {
  V.lugar = nuevatemp()
  id.lugar = buscaLugarTS(id.pos)
  V.cod = gen(id.lugar, =, id.lugar, -, 1) || gen(V.lugar, =, id.lugar)
}
V → id {
  V.lugar = buscaLugarTS(id.pos)
  V.cod = vacio
}
V → ( E ) {
  V.lugar = E.lugar
  V.cod = E.cod
}
V → H {
  V.lugar = H.lugar
  V.cod = H.cod
}

V → ent {
  V.lugar = nuevatemp()
  V.cod = gen(V.lugar, =, ent.valor)
}
V → cad {
  V.lugar = nuevatemp()
  V.cod = gen(V.lugar, =, cad.valor)
}
V → log {
  V.lugar = nuevatmp()
  V.cod = gen(V.lugar, =, log.valor)
}

```

8 | Diseño del registro de activación

El registro de activación considerado para la práctica es el siguiente:

Estado de la máquina	EM
Parámetros actuales	P
Variables locales	VL
Datos temporales	DT
Valor devuelto	VD

Se han suprimido tanto el PC como el PA. El primero se debe a que los diferentes RAs se almacenan en pila uno encima del anterior, por lo tanto no es necesario almacenar la dirección del antiguo RA que llamó a este. El segundo se ha suprimido porque al no haber funciones anidadas en nuestro lenguaje no hay necesidad de almacenar un PA.

El campo valor devuelto (VD), se ha implementado en la práctica usando un registro. Es decir, en lugar de extraer el resultado de la función del campo VD, se almacena en un registro y de este registro se extrae dicho valor. Más adelante mostraré en CO su uso.

9 | Diseño del código objeto

9.1 Métodos auxiliares

Método auxiliar	Código objeto
store_in_reg(oper, REG, mode)	<pre>if oper is global: if mode is direccion: ADD #DESP, .IY MOVE .A, REG else if mode is valor: ADD #DESP, .IY MOVE .A, .R9 MOVE [.R9], REG else if oper is local: if mode is direccion: ADD #DESP, .IX MOVE .A, REG else if mode is valor: ADD #DESP, .IX MOVE .A, .R9 MOVE [.R9], REG else if oper is entero: if mode is valor: MOVE #LITERAL, REG else if oper is cadena: if mode is direccion: MOVE #etiQCad, REG</pre>
copy_str(REG_ORIGEN, REG_DESTINO)	<pre>copiaN: NOP MOVE [REG_ORIGEN], .R9 MOVE .R9, [REG_DESTINO] ADD #1, REG_ORIGEN MOVE .A, REG_ORIGEN ADD #1, REG_DESTINO MOVE .A, REG_DESTINO CMP #0, .R9 BNZ copiaN</pre>

9.2 Aritméticas y lógicas

Operación "AND" lógico:

Código intermedio	Código objeto
[=and, op1, op2, res]	store_in_reg(op1, .R1, Value) store_in_reg(op2, .R2, Value) store_in_reg(res, .R3, Dir) AND .R1, .R2 MOVE .A, [.R3]

Operación resta aritmética:

Código intermedio	Código objeto
[=-, op1, op2, res]	store_in_reg(op1, .R1, Value) store_in_reg(op2, .R2, Value) store_in_reg(res, .R3, Dir) SUB .R1, .R2 MOVE .A, [.R3]

9.3 Asignaciones

Asignación de un entero o lógico:

Código intermedio	Código objeto
[=EL, op1, , res]	store_in_reg(op1, .R1, Value) store_in_reg(res, .R3, Dir) MOVE .R1, [.R3]

Asignación de una cadena:

Código intermedio	Código objeto
[=Cad, op1, , res]	store_in_reg(op1, .R1, Dir) store_in_reg(res, .R3, Dir) copy_str(.R1, .R3)

9.4 Saltos y etiquetas

Añadir una etiqueta:

Código intermedio	Código objeto
[:, etiqueta, ,]	etiqueta: NOP

Salto a una etiqueta:

Código intermedio	Código objeto
[goto, , , etiqueta]	BR /etiqueta

Salto en caso de igual:

Código intermedio	Código objeto
[if=goto, op1, op2, etiqueta]	store_in_reg(op1, .R1, Value) store_in_reg(op2, .R2, Value) CMP .R1 .R2 BZ /etiqueta

9.5 Paso de parámetros

Paso de un entero o lógico:

Código intermedio	Código objeto
[paramEL, op1, ,]	store_in_reg(op1, .R1, Dir) ADD #Tam_RA_Actual, .IX ADD #X, .A MOVE [.R1], [.A]

Paso de una cadena:

Código intermedio	Código objeto
[paramCAD, op1, ,]	store_in_reg(op1, .R1, Dir) ADD #Tam_RA_Actual, .IX ADD #X, .A MOVE A, .R3 copy_str(.R1,.R3)

9.6 Llamadas

Llamada a una funcion que no devuelve nada:

Código intermedio	Código objeto
[callVoid,etiqueta , ,]	ADD #Tam_RA_Actual, .IX MOVE #Dir_Ret, [.A] MOVE .A, .IX BR /etiqueta Dir_Ret: NOP SUB .IX, #Tam_RA_Actual MOVE .A, .IX

Llamada a una funcion devuelve entero o logico:

Código intermedio	Código objeto
[callValueEL, etiqueta, , res]	ADD #Tam_RA_Actual, .IX MOVE #Dir_Ret, [.A] MOVE .A, .IX BR /etiqueta Dir_Ret: NOP SUB .IX, #Tam_RA_Actual MOVE .A, .IX store_in_reg(res, .R3, Dir) MOVE .Reg_Ret, [.R3]

Llamada a una funcion devuelve una cadena:

Código intermedio	Código objeto
[callValueCAD, etiqueta, , res]	ADD #Tam_RA_Actual, .IX MOVE #Dir_Ret, [.A] MOVE .A, .IX BR /etiqueta Dir_Ret: NOP SUB .IX, #Tam_RA_Actual MOVE .A, .IX store_in_reg(res, .R3, Dir) copy_str(.Reg_Ret, .R3)

9.7 Sentencias de retorno

Return sin valor devuelto:

Código intermedio	Código objeto
[returnVoid, ,]	BR [.IX]

Return de entero o lógico:

Código intermedio	Código objeto
[returnEL, op1, ,]	store_in_reg(op1, .Reg_Ret, Value) BR [.IX]

Return de una cadena:

Código intermedio	Código objeto
[returnCAD, op1, ,]	store_in_reg(op1, .Reg_Ret, Dir) BR [.IX]

9.8 Alerts e inputs

Alert de un entero:

Código intermedio	Código objeto
[alertEnt, op1, ,]	store_in_reg(op1, .Reg_Aux, Value) WRINT .Reg_Aux

Alert de una cadena:

Código intermedio	Código objeto
[alertCad, op1, ,]	store_in_reg(op1, .Reg_Aux, Dir) WRSTR .Reg_Aux

Input de un entero:

Código intermedio	Código objeto
[inputEnt, op1, ,]	store_in_reg(op1, .Reg_Aux, Dir) ININT .Reg_Aux

Input de una cadena:

Código intermedio	Código objeto
[inputCad, op1, ,]	store_in_reg(op1, .Reg_Aux, Dir) INSTR .Reg_Aux

10 | Anexo

A continuación se listan las cinco pruebas solicitadas.

10.1 Prueba 1 - test del for

Contenido de la prueba:

```
1 let number numero;
2 let string s;
3 numero = 5;
4 s = "hola\n";
5 function number f(number n, string saludo){
6     let boolean seguir;
7     seguir = true;
8     for(global = 0; seguir; --n){
9         alert(saludo);
10        alert(n);
11        alert("\n");
12        if (n == 0)
13            seguir = false;
14    }
15    return n;
16 }
17 alert("El ultimo valor de numero deberia ser -1\n");
18 numero = f(numero,s);
19 alert("El ultimo valor de numero es:\n");
20 alert(numero);
```

Contenido del CI:

```
1 ; ----- Inicializacion variables globales no inicializadas -----
2 [=EL, 0, , global]
3 ; ----- Fin de inicializacion de variables globales no inicializadas -----
4
5 ; -----Codigo de las funciones -----
6
7 ; ----- Inicio de funcion
8 [:, #EtiqFun0_f, , ]
9 [=EL, 0, , seguir]
10
11 ; Inicio de asignacion
12 [=EL, 1, , ~Temp2]
13 [=EL, ~Temp2, , seguir]
14 ; Fin de asignacion
15
16
```

```

17 ; ---- Inicio de for
18
19 ; Inicio de inicializacion
20
21 ; Inicio de asignacion
22 [=EL, 0, , ~Temp3]
23 [=EL, ~Temp3, , global]
24 ; Fin de asignacion
25
26 ; Fin de inicializacion
27
28 [:, #Etiq3, , ]
29
30 ; Inicio de condicion
31 ; Fin de condicion
32
33 [if=goto, seguir, 0, #Etiq4]
34
35 ; Inicio del cuerpo
36
37 ; Inicio de llamada a alert
38 [alertCad, saludo, , ]
39 ; Fin de llamada a alert
40
41
42 ; Inicio de llamada a alert
43 [alertEnt, n, , ]
44 ; Fin de llamada a alert
45
46
47 ; Inicio de llamada a alert
48 [=Cad, "\n", , ~Temp4]
49 [alertCad, ~Temp4, , ]
50 ; Fin de llamada a alert
51
52
53 ; ---- Inicio de if simple
54
55 ; Inicio de condicion
56
57 ; Inicio de operador de igualdad
58 [=EL, 0, , ~Temp5]
59 [if=goto, n, ~Temp5, #Etiq0]
60 [=EL, 0, , ~Temp6]
61 [goto, , , #Etiq1]
62 [:, #Etiq0, , ]
63 [=EL, 1, , ~Temp6]
64 [:, #Etiq1, , ]
65 ; Inicio de operador de igualdad

```

```

66
67 ; Fin de condicion
68
69 [if=goto, ~Temp6, 0, #Etiqu2]
70
71 ; Inicio de sentencia
72
73 ; Inicio de asignacion
74 [=EL, 0, , ~Temp7]
75 [=EL, ~Temp7, , seguir]
76 ; Fin de asignacion
77
78 ; Fin de sentencia
79
80 [:, #Etiqu2, , ]
81 ; ---- Fin de if simple
82
83 ; Fin del cuerpo
84
85
86 ; Inicio de actualización
87
88 ; Inicio de --id sin asignacion
89 [=, n, 1, n]
90 ; Fin de --id sin asignacion
91
92 ; Fin de actualizacion
93
94 [goto, , , #Etiqu3]
95 [:, #Etiqu4, , ]
96 ; ---- Fin de for
97
98 [returnEL, n, , ]
99
100 [returnVoid, , , ]
101 ; ----- Fin de funcion
102
103 ; ----- Fin de codigo de las funciones-----
104
105 [=EL, 0, , numero]
106 [=Cad, "", , s]
107
108 ; Inicio de asignacion
109 [=EL, 5, , ~Temp0]
110 [=EL, ~Temp0, , numero]
111 ; Fin de asignacion
112
113
114 ; Inicio de asignacion

```



```

115 [=Cad, "hola\n", , ~Temp1]
116 [=Cad, ~Temp1, , s]
117 ; Fin de asignacion
118
119
120 ; Inicio de llamada a alert
121 [=Cad, "El ultimo valor de numero deberia ser -1\n", , ~Temp8]
122 [alertCad, ~Temp8, , ]
123 ; Fin de llamada a alert
124
125
126 ; Inicio de asignacion
127
128 ; ---- Inicio de llamada a funcion
129
130 ; Inicio de paso de parámetros
131 [paramEL, numero, , ]
132 [paramCad, s, , ]
133 ; Fin de paso de parámetros
134
135 [callValueEL, #EtiquFun0_f, , ~Temp9]
136 ; ---- Fin de llamada a funcion
137
138 [=EL, ~Temp9, , numero]
139 ; Fin de asignacion
140
141
142 ; Inicio de llamada a alert
143 [=Cad, "El ultimo valor de numero es:\n", , ~Temp10]
144 [alertCad, ~Temp10, , ]
145 ; Fin de llamada a alert
146
147
148 ; Inicio de llamada a alert
149 [alertEnt, numero, , ]
150 ; Fin de llamada a alert

```

Contenido del CO:

```

1          ORG 0
2          MOVE #beginED, .IY
3          MOVE #beginStack, .IX
4          BR /main
5
6          ; ----- Inicializacion variables globales no inicializadas -----
7
8          ; Valor de Oper1 en R1
9
10         MOVE #0, .R1

```

```

11
12         ; Direccion de Res en R3
13
14         ADD  #130, .IY
15         MOVE .A, .R3
16         MOVE .R1, [.R3] ; Valor de Oper1(R1) a Res(direccion a donde
17
18         ; ----- Fin de inicializacion de variables globales no inicializadas -----
19
20
21         ; ----- Codigo de las funciones -----
22
23         ; ----- Inicio de funcion
24 EtiqFun0_f:      NOP
25
26         ; Valor de Oper1 en R1
27
28         MOVE  #0, .R1
29
30         ; Direccion de Res en R3
31
32         ADD  #66, .IX
33         MOVE .A, .R3
34         MOVE .R1, [.R3] ; Valor de Oper1(R1) a Res(direccion a donde
35
36         ; Inicio de asignacion
37
38         ; Valor de Oper1 en R1
39
40         MOVE  #1, .R1
41
42         ; Direccion de Res en R3
43
44         ADD  #67, .IX
45         MOVE .A, .R3
46         MOVE .R1, [.R3] ; Valor de Oper1(R1) a Res(direccion a donde
47
48         ; Valor de Oper1 en R1
49
50         ADD  #67, .IX
51         MOVE .A, .R9
52         MOVE [.R9], .R1
53
54         ; Direccion de Res en R3
55
56         ADD  #66, .IX
57         MOVE .A, .R3
58         MOVE .R1, [.R3] ; Valor de Oper1(R1) a Res(direccion a donde
59

```

```

60      ; Fin de asignacion
61
62
63      ; ---- Inicio de for
64
65      ; Inicio de inicializacion
66
67      ; Inicio de asignacion
68
69          ; Valor de Oper1 en R1
70
71          MOVE    #0, .R1
72
73          ; Direccion de Res en R3
74
75          ADD     #68, .IX
76          MOVE    .A, .R3
77          MOVE    .R1, [.R3] ; Valor de Oper1(R1) a Res(direccion a donde
78
79          ; Valor de Oper1 en R1
80
81          ADD     #68, .IX
82          MOVE    .A, .R9
83          MOVE    [.R9], .R1
84
85          ; Direccion de Res en R3
86
87          ADD     #130, .IY
88          MOVE    .A, .R3
89          MOVE    .R1, [.R3] ; Valor de Oper1(R1) a Res(direccion a donde
90
91      ; Fin de asignacion
92
93
94      ; Fin de inicializacion
95
96      Etiq3:      NOP
97
98      ; Inicio de condicion
99
100     ; Fin de condicion
101
102         ADD     #66, .IX
103         MOVE    .A, .R9
104         MOVE    [.R9], .R1
105         MOVE    #0, .R2
106         CMP     .R1, .R2
107         BZ      /Etiq4
108

```

```

109 ; Inicio del cuerpo
110
111 ; Inicio de llamada a alert
112     ADD #2, .IX
113     MOVE .A, .R9
114     WRSTR [.R9]
115
116 ; Fin de llamada a alert
117
118
119 ; Inicio de llamada a alert
120     ADD #1, .IX
121     MOVE .A, .R9
122     MOVE [.R9], .R9
123     WRINT .R9
124
125 ; Fin de llamada a alert
126
127
128 ; Inicio de llamada a alert
129     MOVE #cad0_n, .R1
130     ADD #69, .IX
131     MOVE .A, .R3
132
133 ; Inicio bucle de copia de cadena
134 copia0:    NOP
135           MOVE [.R1], .R9
136           MOVE .R9, [.R3]
137           ADD #1, .R1
138           MOVE .A, .R1
139           ADD #1, .R3
140           MOVE .A, .R3
141           CMP #0, .R9
142           BNZ /copia0
143
144 ; Fin bucle de copia de cadena
145     ADD #69, .IX
146     MOVE .A, .R9
147     WRSTR [.R9]
148
149 ; Fin de llamada a alert
150
151
152 ; ---- Inicio de if simple
153
154 ; Inicio de condicion
155
156 ; Inicio de operador de igualdad
157

```

```

158         ; Valor de Oper1 en R1
159
160         MOVE    #0, .R1
161
162         ; Direccion de Res en R3
163
164         ADD     #133, .IX
165         MOVE    .A, .R3
166         MOVE    .R1, [.R3] ; Valor de Oper1(R1) a Res(direccion a donde
167         ADD     #1, .IX
168         MOVE    .A, .R9
169         MOVE    [.R9], .R1
170         ADD     #133, .IX
171         MOVE    .A, .R9
172         MOVE    [.R9], .R2
173         CMP     .R1, .R2
174         BZ      /Etiqu0
175
176         ; Valor de Oper1 en R1
177
178         MOVE    #0, .R1
179
180         ; Direccion de Res en R3
181
182         ADD     #134, .IX
183         MOVE    .A, .R3
184         MOVE    .R1, [.R3] ; Valor de Oper1(R1) a Res(direccion a donde
185         BR      /Etiqu1
186 Etiqu0:    NOP
187
188         ; Valor de Oper1 en R1
189
190         MOVE    #1, .R1
191
192         ; Direccion de Res en R3
193
194         ADD     #134, .IX
195         MOVE    .A, .R3
196         MOVE    .R1, [.R3] ; Valor de Oper1(R1) a Res(direccion a donde
197 Etiqu1:    NOP
198
199         ; Inicio de operador de igualdad
200
201
202         ; Fin de condicion
203
204         ADD     #134, .IX
205         MOVE    .A, .R9
206         MOVE    [.R9], .R1

```

```

207             MOVE  #0, .R2
208             CMP   .R1, .R2
209             BZ    /Etiqu2
210
211     ; Inicio de sentencia
212
213     ; Inicio de asignacion
214
215             ; Valor de Oper1 en R1
216
217             MOVE  #0, .R1
218
219             ; Direccion de Res en R3
220
221             ADD   #135, .IX
222             MOVE  .A, .R3
223             MOVE  .R1, [.R3] ; Valor de Oper1(R1) a Res(direccion a donde
224
225             ; Valor de Oper1 en R1
226
227             ADD   #135, .IX
228             MOVE  .A, .R9
229             MOVE  [.R9], .R1
230
231             ; Direccion de Res en R3
232
233             ADD   #66, .IX
234             MOVE  .A, .R3
235             MOVE  .R1, [.R3] ; Valor de Oper1(R1) a Res(direccion a donde
236
237     ; Fin de asignacion
238
239
240     ; Fin de sentencia
241
242 Etiqu2:      NOP
243
244     ; ---- Fin de if simple
245
246
247     ; Fin del cuerpo
248
249
250     ; Inicio de actualización
251
252     ; Inicio de --id sin asignacion
253             ADD   #1, .IX
254             MOVE  .A, .R9
255             MOVE  [.R9], .R1

```

```

256             MOVE #1, .R2
257             ADD #1, .IX
258             MOVE .A, .R3
259             SUB .R1, .R2
260             MOVE .A, [.R3]
261
262             ; Fin de --id sin asignacion
263
264
265             ; Fin de actualizacion
266
267             BR /Etiqu3
268 Etiqu4:      NOP
269
270             ; ---- Fin de for
271
272
273             ;Valor a devolver en .R8
274
275             ADD #1, .IX
276             MOVE .A, .R9
277             MOVE [.R9], .R8
278             BR [.IX]
279             BR [.IX]
280
281             ; ----- Fin de funcion
282
283
284             ; ----- Fin de codigo de las funciones-----
285
286
287             ; Inicio de código del main
288 main:      NOP
289
290             ; Valor de Oper1 en R1
291
292             MOVE #0, .R1
293
294             ; Direccion de Res en R3
295
296             ADD #0, .IY
297             MOVE .A, .R3
298             MOVE .R1, [.R3] ; Valor de Oper1(R1) a Res(direccion a donde
299             MOVE #cad1_, .R1
300             ADD #1, .IY
301             MOVE .A, .R3
302
303             ; Inicio bucle de copia de cadena
304 copia1:   NOP

```

```

305             MOVE  [.R1], .R9
306             MOVE  .R9, [.R3]
307             ADD   #1, .R1
308             MOVE  .A, .R1
309             ADD   #1, .R3
310             MOVE  .A, .R3
311             CMP   #0, .R9
312             BNZ   /copia1
313
314     ; Fin bucle de copia de cadena
315
316     ; Inicio de asignacion
317
318             ; Valor de Oper1 en R1
319
320             MOVE   #5, .R1
321
322             ; Direccion de Res en R3
323
324             ADD    #65, .IY
325             MOVE   .A, .R3
326             MOVE   .R1, [.R3] ; Valor de Oper1(R1) a Res(direccion a donde
327
328             ; Valor de Oper1 en R1
329
330             ADD    #65, .IY
331             MOVE   .A, .R9
332             MOVE   [.R9], .R1
333
334             ; Direccion de Res en R3
335
336             ADD    #0, .IY
337             MOVE   .A, .R3
338             MOVE   .R1, [.R3] ; Valor de Oper1(R1) a Res(direccion a donde
339
340     ; Fin de asignacion
341
342
343     ; Inicio de asignacion
344             MOVE   #cad2_hola, .R1
345             ADD    #66, .IY
346             MOVE   .A, .R3
347
348     ; Inicio bucle de copia de cadena
349 copia2:      NOP
350             MOVE   [.R1], .R9
351             MOVE   .R9, [.R3]
352             ADD    #1, .R1
353             MOVE   .A, .R1

```



```

354             ADD #1, .R3
355             MOVE .A, .R3
356             CMP #0, .R9
357             BNZ /copia2
358
359             ; Fin bucle de copia de cadena
360             ADD #66, .IY
361             MOVE .A, .R1
362             ADD #1, .IY
363             MOVE .A, .R3
364
365             ; Inicio bucle de copia de cadena
366 copia3:      NOP
367             MOVE [.R1], .R9
368             MOVE .R9, [.R3]
369             ADD #1, .R1
370             MOVE .A, .R1
371             ADD #1, .R3
372             MOVE .A, .R3
373             CMP #0, .R9
374             BNZ /copia3
375
376             ; Fin bucle de copia de cadena
377
378             ; Fin de asignacion
379
380
381             ; Inicio de llamada a alert
382             MOVE #cad3_Elul, .R1
383             ADD #131, .IY
384             MOVE .A, .R3
385
386             ; Inicio bucle de copia de cadena
387 copia4:      NOP
388             MOVE [.R1], .R9
389             MOVE .R9, [.R3]
390             ADD #1, .R1
391             MOVE .A, .R1
392             ADD #1, .R3
393             MOVE .A, .R3
394             CMP #0, .R9
395             BNZ /copia4
396
397             ; Fin bucle de copia de cadena
398             ADD #131, .IY
399             MOVE .A, .R9
400             WRSTR [.R9]
401
402             ; Fin de llamada a alert

```

```

403
404
405 ; Inicio de asignacion
406
407 ; ---- Inicio de llamada a funcion
408
409 ; Inicio de paso de parámetros
410         ADD #0, .IY
411         MOVE .A, .R1
412         ADD #260, .IX
413         ADD #1, .A ; .A contiene la dirección del parametro alojado e
414         MOVE [.R1], [.A]
415         ADD #1, .IY
416         MOVE .A, .R1
417         ADD #260, .IX
418         ADD #2, .A ; .A contiene la dirección del parametro alojado e
419         MOVE .A, .R3
420
421 ; Inicio bucle de copia de cadena
422 copia5:  NOP
423         MOVE [.R1], .R9
424         MOVE .R9, [.R3]
425         ADD #1, .R1
426         MOVE .A, .R1
427         ADD #1, .R3
428         MOVE .A, .R3
429         CMP #0, .R9
430         BNZ /copia5
431
432 ; Fin bucle de copia de cadena
433
434 ; Fin de paso de parámetros
435
436
437 ; Secuencia de llamada
438         ADD #tamRAFunMain, .IX
439         MOVE #dirRet0_Fun0_f, [.A]
440         ADD #tamRAFunMain, .IX
441         MOVE .A, .IX
442         BR /EtiqFun0_f
443
444 ; Secuencia de retorno
445 dirRet0_Fun0_f:  NOP
446         SUB .IX, #tamRAFunMain
447         MOVE .A, .IX
448         ADD #195, .IY
449         MOVE .A, .R3
450         MOVE .R8, [.R3]
451

```

```

452 ; ---- Fin de llamada a funcion
453
454
455 ; Valor de Oper1 en R1
456
457 ADD #195, .IY
458 MOVE .A, .R9
459 MOVE [.R9], .R1
460
461 ; Direccion de Res en R3
462
463 ADD #0, .IY
464 MOVE .A, .R3
465 MOVE .R1, [.R3] ; Valor de Oper1(R1) a Res(direccion a donde
466
467 ; Fin de asignacion
468
469
470 ; Inicio de llamada a alert
471 MOVE #cad4_Elul, .R1
472 ADD #196, .IY
473 MOVE .A, .R3
474
475 ; Inicio bucle de copia de cadena
476 copia6: NOP
477 MOVE [.R1], .R9
478 MOVE .R9, [.R3]
479 ADD #1, .R1
480 MOVE .A, .R1
481 ADD #1, .R3
482 MOVE .A, .R3
483 CMP #0, .R9
484 BNZ /copia6
485
486 ; Fin bucle de copia de cadena
487 ADD #196, .IY
488 MOVE .A, .R9
489 WRSTR [.R9]
490
491 ; Fin de llamada a alert
492
493
494 ; Inicio de llamada a alert
495 ADD #0, .IY
496 MOVE .A, .R9
497 MOVE [.R9], .R9
498 WRINT .R9
499
500 ; Fin de llamada a alert

```

```

501
502             HALT
503
504     ; Fin de código del main
505
506 tamRAFun0_f:      EQU 136
507 tamRAFunMain:     EQU 260
508 beginED:          RES 260
509 cad0_n:           DATA "\n"
510 cad1_:            DATA ""
511 cad2_hola:        DATA "hola\n"
512 cad3_Elul:        DATA "El ultimo valor de numero deberia ser -1\n"
513 cad4_Elul:        DATA "El ultimo valor de numero es:\n"
514 beginStack:       NOP
515                  END

```

10.2 Prueba 2 - test de operaciones entrada y salida

Contenido de la prueba:

```
1 let number numero;
2 let string cadena;
3 function string pide_cadena(){
4     let string c;
5     alert("Introduce la cadena:\n");
6     input(c);
7     return c;
8 }
9 function number pide_numero(){
10    let number n;
11    alert("Introduce el numero:\n");
12    input(n);
13    return n;
14 }
15 cadena = pide_cadena();
16 alert("La cadena leida fue:\n");
17 alert(cadena);
18 alert("\n");
19 numero = pide_numero();
20 alert("El numero leido fue:\n");
21 alert(numero);
```

Contenido del CI:

```
1 ; ----- Codigo de las funciones -----
2
3 ; ----- Inicio de funcion
4 [:, #EtiquFun0_pide, , ]
5 [=Cad, "", , c]
6
7 ; Inicio de llamada a alert
8 [=Cad, "Introduce la cadena:\n", , ~Temp0]
9 [alertCad, ~Temp0, , ]
10 ; Fin de llamada a alert
11
12
13 ; Inicio de llamada a input
14 [inputCad, , , c]
15 ; Fin de llamada a input
16
17 [returnCad, c, , ]
18
19 [returnVoid, , , ]
20 ; ----- Fin de funcion
21
```

```

22
23 ; ----- Inicio de funcion
24 [:, #EtiqFun1_pide, , ]
25 [=EL, 0, , n]
26
27 ; Inicio de llamada a alert
28 [=Cad, "Introduce el numero:\n", , ~Temp1]
29 [alertCad, ~Temp1, , ]
30 ; Fin de llamada a alert
31
32
33 ; Inicio de llamada a input
34 [inputEnt, , , n]
35 ; Fin de llamada a input
36
37 [returnEL, n, , ]
38
39 [returnVoid, , , ]
40 ; ----- Fin de funcion
41
42 ; ----- Fin de codigo de las funciones-----
43
44 [=EL, 0, , numero]
45 [=Cad, "", , cadena]
46
47 ; Inicio de asignacion
48
49 ; ---- Inicio de llamada a funcion
50 [callValueCad, #EtiqFun0_pide, , ~Temp2]
51 ; ---- Fin de llamada a funcion
52
53 [=Cad, ~Temp2, , cadena]
54 ; Fin de asignacion
55
56
57 ; Inicio de llamada a alert
58 [=Cad, "La cadena leida fue:\n", , ~Temp3]
59 [alertCad, ~Temp3, , ]
60 ; Fin de llamada a alert
61
62
63 ; Inicio de llamada a alert
64 [alertCad, cadena, , ]
65 ; Fin de llamada a alert
66
67
68 ; Inicio de llamada a alert
69 [=Cad, "\n", , ~Temp4]
70 [alertCad, ~Temp4, , ]

```

```

71 ; Fin de llamada a alert
72
73
74 ; Inicio de asignacion
75
76 ; ---- Inicio de llamada a funcion
77 [callValueEL, #EtiqFun1_pide, , ~Temp5]
78 ; ---- Fin de llamada a funcion
79
80 [=EL, ~Temp5, , numero]
81 ; Fin de asignacion
82
83
84 ; Inicio de llamada a alert
85 [=Cad, "El numero leido fue:\n", , ~Temp6]
86 [alertCad, ~Temp6, , ]
87 ; Fin de llamada a alert
88
89
90 ; Inicio de llamada a alert
91 [alertEnt, numero, , ]
92 ; Fin de llamada a alert

```

Contenido del CO:

```

1          ORG 0
2          MOVE #beginED, .IY
3          MOVE #beginStack, .IX
4          BR /main
5
6          ; ----- Codigo de las funciones -----
7
8          ; ----- Inicio de funcion
9 EtiqFun0_pide:  NOP
10             MOVE #cad0_, .R1
11             ADD #1, .IX
12             MOVE .A, .R3
13
14             ; Inicio bucle de copia de cadena
15 copia0:      NOP
16             MOVE [.R1], .R9
17             MOVE .R9, [.R3]
18             ADD #1, .R1
19             MOVE .A, .R1
20             ADD #1, .R3
21             MOVE .A, .R3
22             CMP #0, .R9
23             BNZ /copia0
24

```

```

25      ; Fin bucle de copia de cadena
26
27      ; Inicio de llamada a alert
28          MOVE #cad1_Intr, .R1
29          ADD #65, .IX
30          MOVE .A, .R3
31
32      ; Inicio bucle de copia de cadena
33 copia1:    NOP
34          MOVE [.R1], .R9
35          MOVE .R9, [.R3]
36          ADD #1, .R1
37          MOVE .A, .R1
38          ADD #1, .R3
39          MOVE .A, .R3
40          CMP #0, .R9
41          BNZ /copia1
42
43      ; Fin bucle de copia de cadena
44          ADD #65, .IX
45          MOVE .A, .R9
46          WRSTR [.R9]
47
48      ; Fin de llamada a alert
49
50
51      ; Inicio de llamada a input
52          ADD #1, .IX
53          MOVE .A, .R9
54          INSTR [.R9]
55
56      ; Fin de llamada a input
57
58
59          ;Direccion de la cadena a devolver en .R8
60
61          ADD #1, .IX
62          MOVE .A, .R8
63          BR [.IX]
64          BR [.IX]
65
66      ; ----- Fin de funcion
67
68
69      ; ----- Inicio de funcion
70 EtiqFun1_pide:    NOP
71
72          ; Valor de Oper1 en R1
73

```



```

74             MOVE  #0, .R1
75
76             ; Direccion de Res en R3
77
78             ADD   #1, .IX
79             MOVE  .A, .R3
80             MOVE  .R1, [.R3] ; Valor de Oper1(R1) a Res(direccion a donde
81
82             ; Inicio de llamada a alert
83             MOVE  #cad2_Intr, .R1
84             ADD   #2, .IX
85             MOVE  .A, .R3
86
87             ; Inicio bucle de copia de cadena
88 copia2:      NOP
89             MOVE  [.R1], .R9
90             MOVE  .R9, [.R3]
91             ADD   #1, .R1
92             MOVE  .A, .R1
93             ADD   #1, .R3
94             MOVE  .A, .R3
95             CMP   #0, .R9
96             BNZ   /copia2
97
98             ; Fin bucle de copia de cadena
99             ADD   #2, .IX
100            MOVE  .A, .R9
101            WRSTR  [.R9]
102
103            ; Fin de llamada a alert
104
105
106            ; Inicio de llamada a input
107            ADD   #1, .IX
108            MOVE  .A, .R9
109            ININT  [.R9]
110
111            ; Fin de llamada a input
112
113
114            ;Valor a devolver en .R8
115
116            ADD   #1, .IX
117            MOVE  .A, .R9
118            MOVE  [.R9], .R8
119            BR    [.IX]
120            BR    [.IX]
121
122            ; ----- Fin de funcion

```

```

123
124
125     ; ----- Fin de codigo de las funciones-----
126
127
128     ; Inicio de código del main
129 main:      NOP
130
131           ; Valor de Oper1 en R1
132
133           MOVE  #0, .R1
134
135           ; Direccion de Res en R3
136
137           ADD   #0, .IY
138           MOVE  .A, .R3
139           MOVE  .R1, [.R3] ; Valor de Oper1(R1) a Res(direccion a donde
140           MOVE  #cad3_, .R1
141           ADD   #1, .IY
142           MOVE  .A, .R3
143
144           ; Inicio bucle de copia de cadena
145 copia3:    NOP
146           MOVE  [.R1], .R9
147           MOVE  .R9, [.R3]
148           ADD   #1, .R1
149           MOVE  .A, .R1
150           ADD   #1, .R3
151           MOVE  .A, .R3
152           CMP   #0, .R9
153           BNZ   /copia3
154
155           ; Fin bucle de copia de cadena
156
157           ; Inicio de asignacion
158
159           ; ---- Inicio de llamada a funcion
160
161           ; Secuencia de llamada
162           ADD   #tamRAFunMain, .IX
163           MOVE  #dirRet0_Fun0_pide, [.A]
164           ADD   #tamRAFunMain, .IX
165           MOVE  .A, .IX
166           BR    /EtiqFun0_pide
167
168           ; Secuencia de retorno
169 dirRet0_Fun0_pide:  NOP
170                   SUB  .IX, #tamRAFunMain
171                   MOVE .A, .IX

```

```

172             ADD #65, .IY
173             MOVE .A, .R3
174
175             ; Inicio bucle de copia de cadena
176 copia4:      NOP
177             MOVE [.R8], .R9
178             MOVE .R9, [.R3]
179             ADD #1, .R8
180             MOVE .A, .R8
181             ADD #1, .R3
182             MOVE .A, .R3
183             CMP #0, .R9
184             BNZ /copia4
185
186             ; Fin bucle de copia de cadena
187
188             ; ---- Fin de llamada a funcion
189
190             ADD #65, .IY
191             MOVE .A, .R1
192             ADD #1, .IY
193             MOVE .A, .R3
194
195             ; Inicio bucle de copia de cadena
196 copia5:      NOP
197             MOVE [.R1], .R9
198             MOVE .R9, [.R3]
199             ADD #1, .R1
200             MOVE .A, .R1
201             ADD #1, .R3
202             MOVE .A, .R3
203             CMP #0, .R9
204             BNZ /copia5
205
206             ; Fin bucle de copia de cadena
207
208             ; Fin de asignacion
209
210
211             ; Inicio de llamada a alert
212             MOVE #cad4_Laca, .R1
213             ADD #129, .IY
214             MOVE .A, .R3
215
216             ; Inicio bucle de copia de cadena
217 copia6:      NOP
218             MOVE [.R1], .R9
219             MOVE .R9, [.R3]
220             ADD #1, .R1

```

```

221             MOVE .A, .R1
222             ADD #1, .R3
223             MOVE .A, .R3
224             CMP #0, .R9
225             BNZ /copia6
226
227 ; Fin bucle de copia de cadena
228             ADD #129, .IY
229             MOVE .A, .R9
230             WRSTR [.R9]
231
232 ; Fin de llamada a alert
233
234
235 ; Inicio de llamada a alert
236             ADD #1, .IY
237             MOVE .A, .R9
238             WRSTR [.R9]
239
240 ; Fin de llamada a alert
241
242
243 ; Inicio de llamada a alert
244             MOVE #cad5_n, .R1
245             ADD #193, .IY
246             MOVE .A, .R3
247
248 ; Inicio bucle de copia de cadena
249 copia7:      NOP
250             MOVE [.R1], .R9
251             MOVE .R9, [.R3]
252             ADD #1, .R1
253             MOVE .A, .R1
254             ADD #1, .R3
255             MOVE .A, .R3
256             CMP #0, .R9
257             BNZ /copia7
258
259 ; Fin bucle de copia de cadena
260             ADD #193, .IY
261             MOVE .A, .R9
262             WRSTR [.R9]
263
264 ; Fin de llamada a alert
265
266
267 ; Inicio de asignacion
268
269 ; ---- Inicio de llamada a funcion

```

```

270
271     ; Secuencia de llamada
272         ADD #tamRAFunMain, .IX
273         MOVE #dirRet1_Fun1_pide, [.A]
274         ADD #tamRAFunMain, .IX
275         MOVE .A, .IX
276         BR /EtiqFun1_pide
277
278     ; Secuencia de retorno
279 dirRet1_Fun1_pide: NOP
280         SUB .IX, #tamRAFunMain
281         MOVE .A, .IX
282         ADD #257, .IY
283         MOVE .A, .R3
284         MOVE .R8, [.R3]
285
286     ; ---- Fin de llamada a funcion
287
288
289     ; Valor de Oper1 en R1
290
291         ADD #257, .IY
292         MOVE .A, .R9
293         MOVE [.R9], .R1
294
295     ; Direccion de Res en R3
296
297         ADD #0, .IY
298         MOVE .A, .R3
299         MOVE .R1, [.R3] ; Valor de Oper1(R1) a Res(direccion a donde
300
301     ; Fin de asignacion
302
303
304     ; Inicio de llamada a alert
305         MOVE #cad6_Elnu, .R1
306         ADD #258, .IY
307         MOVE .A, .R3
308
309     ; Inicio bucle de copia de cadena
310 copia8: NOP
311         MOVE [.R1], .R9
312         MOVE .R9, [.R3]
313         ADD #1, .R1
314         MOVE .A, .R1
315         ADD #1, .R3
316         MOVE .A, .R3
317         CMP #0, .R9
318         BNZ /copia8

```

```

319
320     ; Fin bucle de copia de cadena
321         ADD #258, .IY
322         MOVE .A, .R9
323         WRSTR [.R9]
324
325     ; Fin de llamada a alert
326
327
328     ; Inicio de llamada a alert
329         ADD #0, .IY
330         MOVE .A, .R9
331         MOVE [.R9], .R9
332         WRINT .R9
333
334     ; Fin de llamada a alert
335
336         HALT
337
338     ; Fin de código del main
339
340 tamRAFun0_pide:    EQU 129
341 tamRAFun1_pide:    EQU 66
342 tamRAFunMain:      EQU 322
343 beginED:           RES 322
344 cad0_:             DATA ""
345 cad1_Intr:         DATA "Introduce la cadena:\n"
346 cad2_Intr:         DATA "Introduce el numero:\n"
347 cad3_:             DATA ""
348 cad4_Laca:         DATA "La cadena leida fue:\n"
349 cad5_n:            DATA "\n"
350 cad6_Elnu:         DATA "El numero leido fue:\n"
351 beginStack:        NOP
352                     END

```

10.3 Prueba 3 - test genérico

Contenido de la prueba:

```
1 let number numero;
2 numero = 4;
3 alert("El valor de number es:\n");
4 alert(numero);
5 numero = numero - global;
6 alert("\nAhora numero sigue igual debido a la global\n");
7 alert(numero);
```

Contenido del CI:

```
1 ; ----- Codigo de las funciones -----
2
3 ; ----- Inicio de funcion
4 [:, #EtiqFun0_pide, , ]
5 [=Cad, "", , c]
6
7 ; Inicio de llamada a alert
8 [=Cad, "Introduce la cadena:\n", , ~Temp0]
9 [alertCad, ~Temp0, , ]
10 ; Fin de llamada a alert
11
12
13 ; Inicio de llamada a input
14 [inputCad, , , c]
15 ; Fin de llamada a input
16
17 [returnCad, c, , ]
18
19 [returnVoid, , , ]
20 ; ----- Fin de funcion
21
22
23 ; ----- Inicio de funcion
24 [:, #EtiqFun1_pide, , ]
25 [=EL, 0, , n]
26
27 ; Inicio de llamada a alert
28 [=Cad, "Introduce el numero:\n", , ~Temp1]
29 [alertCad, ~Temp1, , ]
30 ; Fin de llamada a alert
31
32
33 ; Inicio de llamada a input
34 [inputEnt, , , n]
35 ; Fin de llamada a input
```

```

36
37 [returnEL, n, , ]
38
39 [returnVoid, , , ]
40 ; ----- Fin de funcion
41
42 ; ----- Fin de codigo de las funciones-----
43
44 [=EL, 0, , numero]
45 [=Cad, "", , cadena]
46
47 ; Inicio de asignacion
48
49 ; ---- Inicio de llamada a funcion
50 [callValueCad, #EtiqFun0_pide, , ~Temp2]
51 ; ---- Fin de llamada a funcion
52
53 [=Cad, ~Temp2, , cadena]
54 ; Fin de asignacion
55
56
57 ; Inicio de llamada a alert
58 [=Cad, "La cadena leida fue:\n", , ~Temp3]
59 [alertCad, ~Temp3, , ]
60 ; Fin de llamada a alert
61
62
63 ; Inicio de llamada a alert
64 [alertCad, cadena, , ]
65 ; Fin de llamada a alert
66
67
68 ; Inicio de llamada a alert
69 [=Cad, "\n", , ~Temp4]
70 [alertCad, ~Temp4, , ]
71 ; Fin de llamada a alert
72
73
74 ; Inicio de asignacion
75
76 ; ---- Inicio de llamada a funcion
77 [callValueEL, #EtiqFun1_pide, , ~Temp5]
78 ; ---- Fin de llamada a funcion
79
80 [=EL, ~Temp5, , numero]
81 ; Fin de asignacion
82
83
84 ; Inicio de llamada a alert

```



```

85 [=Cad, "El numero leido fue:\n", , ~Temp6]
86 [alertCad, ~Temp6, , ]
87 ; Fin de llamada a alert
88
89
90 ; Inicio de llamada a alert
91 [alertEnt, numero, , ]
92 ; Fin de llamada a alert

```

Contenido del CO:

```

1          ORG 0
2          MOVE #beginED, .IY
3          MOVE #beginStack, .IX
4          BR /main
5 ; ----- Inicializacion variables globales no inicializadas -----
6          ; Valor de Oper1 en R1
7          MOVE #0, .R1
8          ; Direccion de Res en R3
9          ADD #66, .IY
10         MOVE .A, .R3
11         MOVE .R1, [.R3] ; Valor de Oper1(R1) a Res(direccion a donde
12         ; ----- Fin de inicializacion de variables globales no inici
13
14 ; -----Codigo de las funciones -----
15         ; ----- Fin de codigo de las funciones-----
16
17
18
19         ; Inicio de código del main
20 main:    NOP
21         ; Valor de Oper1 en R1
22         MOVE #0, .R1
23         ; Direccion de Res en R3
24         ADD #0, .IY
25         MOVE .A, .R3
26         MOVE .R1, [.R3] ; Valor de Oper1(R1) a Res(direccion a donde
27
28
29         ; Inicio de asignacion
30         ; Valor de Oper1 en R1
31         MOVE #4, .R1
32         ; Direccion de Res en R3
33         ADD #1, .IY
34         MOVE .A, .R3
35         MOVE .R1, [.R3] ; Valor de Oper1(R1) a Res(direccion a donde
36         ; Valor de Oper1 en R1
37         ADD #1, .IY
38         MOVE .A, .R9

```

```

39             MOVE  [.R9], .R1
40     ; Direccion de Res en R3
41             ADD   #0, .IY
42             MOVE  .A, .R3
43             MOVE  .R1, [.R3] ; Valor de Oper1(R1) a Res(direccion a donde
44     ; Fin de asignacion
45
46
47
48     ; Inicio de llamada a alert
49             MOVE  #cad0_Elva, .R1
50             ADD   #2, .IY
51             MOVE  .A, .R3
52
53
54     ; Inicio bucle de copia de cadena
55 copia0:     NOP
56             MOVE  [.R1], .R9
57             MOVE  .R9, [.R3]
58             ADD   #1, .R1
59             MOVE  .A, .R1
60             ADD   #1, .R3
61             MOVE  .A, .R3
62             CMP   #0, .R9
63             BNZ   /copia0
64     ; Fin bucle de copia de cadena
65
66             ADD   #2, .IY
67             MOVE  .A, .R9
68             WRSTR  [.R9]
69     ; Fin de llamada a alert
70
71
72
73     ; Inicio de llamada a alert
74             ADD   #0, .IY
75             MOVE  .A, .R9
76             MOVE  [.R9], .R9
77             WRINT  .R9
78     ; Fin de llamada a alert
79
80
81
82     ; Inicio de asignacion
83
84
85     ; Inicio de resta aritmetica
86             ADD   #0, .IY
87             MOVE  .A, .R9

```

```

88             MOVE  [.R9], .R1
89             ADD   #66, .IY
90             MOVE  .A, .R9
91             MOVE  [.R9], .R2
92             ADD   #66, .IY
93             MOVE  .A, .R3
94             SUB   .R1, .R2
95             MOVE  .A, [.R3]
96 ; Fin de resta aritmetica
97
98 ; Valor de Oper1 en R1
99             ADD   #66, .IY
100            MOVE  .A, .R9
101            MOVE  [.R9], .R1
102 ; Direccion de Res en R3
103            ADD   #0, .IY
104            MOVE  .A, .R3
105            MOVE  .R1, [.R3] ; Valor de Oper1(R1) a Res(direccion a donde
106 ; Fin de asignacion
107
108
109
110 ; Inicio de llamada a alert
111            MOVE  #cad1_nAho, .R1
112            ADD   #67, .IY
113            MOVE  .A, .R3
114
115
116 ; Inicio bucle de copia de cadena
117 copia1:    NOP
118            MOVE  [.R1], .R9
119            MOVE  .R9, [.R3]
120            ADD   #1, .R1
121            MOVE  .A, .R1
122            ADD   #1, .R3
123            MOVE  .A, .R3
124            CMP   #0, .R9
125            BNZ   /copia1
126 ; Fin bucle de copia de cadena
127
128            ADD   #67, .IY
129            MOVE  .A, .R9
130            WRSTR  [.R9]
131 ; Fin de llamada a alert
132
133
134
135 ; Inicio de llamada a alert
136            ADD   #0, .IY

```

```

137             MOVE  .A, .R9
138             MOVE  [.R9], .R9
139             WRINT  .R9
140             ; Fin de llamada a alert
141
142             HALT
143             ; Fin de código del main
144
145 tamRAFunMain: EQU 131
146 beginED:      RES 131
147 cad0_Elva:    DATA "El valor de number es:\n"
148 cad1_nAho:    DATA "\nAhora numero deberia seguir igual debido a la global\n"
149 beginStack:   NOP
150             END

```

10.4 Prueba 4 - test recursivo

Contenido de la prueba:

```
1 let number x;  
2 x = 1;  
3  
4 function print_x(){  
5     alert(x);  
6     alert("\n");  
7  
8     x = x - 1;  
9     if(x == 0)  
10         print_x();  
11 }  
12  
13 print_x();
```

Contenido del CI:

```
1 ; ----- Codigo de las funciones -----  
2  
3 ; ----- Inicio de funcion  
4 [:, #EtiqFun0_prin, , ]  
5  
6 ; Inicio de llamada a alert  
7 [alertEnt, x, , ]  
8 ; Fin de llamada a alert  
9  
10  
11 ; Inicio de llamada a alert  
12 [=Cad, "\n", , ~Temp1]  
13 [alertCad, ~Temp1, , ]  
14 ; Fin de llamada a alert  
15  
16  
17 ; Inicio de asignacion  
18  
19 ; Inicio de resta aritmetica  
20 [=EL, 1, , ~Temp2]  
21 [=, x, ~Temp2, ~Temp3]  
22 ; Fin de resta aritmetica  
23  
24 [=EL, ~Temp3, , x]  
25 ; Fin de asignacion  
26  
27  
28 ; ---- Inicio de if simple  
29
```

```

30 ; Inicio de condicion
31
32 ; Inicio de operador de igualdad
33 [=EL, 0, , ~Temp4]
34 [if=goto, x, ~Temp4, #Etiqu0]
35 [=EL, 0, , ~Temp5]
36 [goto, , , #Etiqu1]
37 [:, #Etiqu0, , ]
38 [=EL, 1, , ~Temp5]
39 [:, #Etiqu1, , ]
40 ; Inicio de operador de igualdad
41
42 ; Fin de condicion
43
44 [if=goto, ~Temp5, 0, #Etiqu2]
45
46 ; Inicio de sentencia
47
48 ; ---- Inicio de llamada a funcion
49 [callVoid, #EtiquFun0_prin, , ]
50 ; ---- Fin de llamada a funcion
51
52 ; Fin de sentencia
53
54 [:, #Etiqu2, , ]
55 ; ---- Fin de if simple
56
57
58 [returnVoid, , , ]
59 ; ----- Fin de funcion
60
61 ; ----- Fin de codigo de las funciones-----
62
63 [=EL, 0, , x]
64
65 ; Inicio de asignacion
66 [=EL, 1, , ~Temp0]
67 [=EL, ~Temp0, , x]
68 ; Fin de asignacion
69
70
71 ; ---- Inicio de llamada a funcion
72 [callVoid, #EtiquFun0_prin, , ]
73 ; ---- Fin de llamada a funcion

```

Contenido del CO:

1	ORG 0
2	MOVE #beginED, .IY

```

3          MOVE #beginStack, .IX
4          BR  /main
5
6          ; ----- Codigo de las funciones -----
7
8          ; ----- Inicio de funcion
9 EtiqFun0_prin:      NOP
10
11         ; Inicio de llamada a alert
12             ADD #0, .IY
13             MOVE .A, .R9
14             MOVE [.R9], .R9
15             WRINT .R9
16
17         ; Fin de llamada a alert
18
19
20         ; Inicio de llamada a alert
21             MOVE #cad0_n, .R1
22             ADD #1, .IX
23             MOVE .A, .R3
24
25         ; Inicio bucle de copia de cadena
26 copia0:      NOP
27             MOVE [.R1], .R9
28             MOVE .R9, [.R3]
29             ADD #1, .R1
30             MOVE .A, .R1
31             ADD #1, .R3
32             MOVE .A, .R3
33             CMP #0, .R9
34             BNZ /copia0
35
36         ; Fin bucle de copia de cadena
37             ADD #1, .IX
38             MOVE .A, .R9
39             WRSTR [.R9]
40
41         ; Fin de llamada a alert
42
43
44         ; Inicio de asignacion
45
46         ; Inicio de resta aritmetica
47
48         ; Valor de Oper1 en R1
49
50             MOVE #1, .R1
51

```

```

52         ; Direccion de Res en R3
53
54         ADD #65, .IX
55         MOVE .A, .R3
56         MOVE .R1, [.R3] ; Valor de Oper1(R1) a Res(direccion a donde
57         ADD #0, .IY
58         MOVE .A, .R9
59         MOVE [.R9], .R1
60         ADD #65, .IX
61         MOVE .A, .R9
62         MOVE [.R9], .R2
63         ADD #66, .IX
64         MOVE .A, .R3
65         SUB .R1, .R2
66         MOVE .A, [.R3]
67
68     ; Fin de resta aritmetica
69
70
71     ; Valor de Oper1 en R1
72
73         ADD #66, .IX
74         MOVE .A, .R9
75         MOVE [.R9], .R1
76
77     ; Direccion de Res en R3
78
79         ADD #0, .IY
80         MOVE .A, .R3
81         MOVE .R1, [.R3] ; Valor de Oper1(R1) a Res(direccion a donde
82
83     ; Fin de asignacion
84
85
86     ; ---- Inicio de if simple
87
88     ; Inicio de condicion
89
90     ; Inicio de operador de igualdad
91
92     ; Valor de Oper1 en R1
93
94         MOVE #0, .R1
95
96     ; Direccion de Res en R3
97
98         ADD #67, .IX
99         MOVE .A, .R3
100        MOVE .R1, [.R3] ; Valor de Oper1(R1) a Res(direccion a donde

```



```

101          ADD #0, .IY
102          MOVE .A, .R9
103          MOVE [.R9], .R1
104          ADD #67, .IX
105          MOVE .A, .R9
106          MOVE [.R9], .R2
107          CMP .R1, .R2
108          BZ /Etiq0
109
110          ; Valor de Oper1 en R1
111
112          MOVE #0, .R1
113
114          ; Direccion de Res en R3
115
116          ADD #68, .IX
117          MOVE .A, .R3
118          MOVE .R1, [.R3] ; Valor de Oper1(R1) a Res(direccion a donde
119          BR /Etiq1
120 Etiq0:      NOP
121
122          ; Valor de Oper1 en R1
123
124          MOVE #1, .R1
125
126          ; Direccion de Res en R3
127
128          ADD #68, .IX
129          MOVE .A, .R3
130          MOVE .R1, [.R3] ; Valor de Oper1(R1) a Res(direccion a donde
131 Etiq1:      NOP
132
133          ; Inicio de operador de igualdad
134
135
136          ; Fin de condicion
137
138          ADD #68, .IX
139          MOVE .A, .R9
140          MOVE [.R9], .R1
141          MOVE #0, .R2
142          CMP .R1, .R2
143          BZ /Etiq2
144
145          ; Inicio de sentencia
146
147          ; ---- Inicio de llamada a funcion
148
149          ; Secuencia de llamada

```

```

150             ADD #tamRAFun0_prin, .IX
151             MOVE #dirRet0_Fun0_prin, [.A]
152             ADD #tamRAFun0_prin, .IX
153             MOVE .A, .IX
154             BR /EtiquFun0_prin
155
156             ; Secuencia de retorno
157 dirRet0_Fun0_prin: NOP
158                 SUB .IX, #tamRAFun0_prin
159                 MOVE .A, .IX
160
161             ; ---- Fin de llamada a funcion
162
163
164             ; Fin de sentencia
165
166 Etiqu2:        NOP
167
168             ; ---- Fin de if simple
169
170                 BR [.IX]
171
172             ; ----- Fin de funcion
173
174
175             ; ----- Fin de codigo de las funciones-----
176
177
178             ; Inicio de código del main
179 main:          NOP
180
181                 ; Valor de Oper1 en R1
182
183                 MOVE #0, .R1
184
185                 ; Direccion de Res en R3
186
187                 ADD #0, .IY
188                 MOVE .A, .R3
189                 MOVE .R1, [.R3] ; Valor de Oper1(R1) a Res(direccion a donde
190
191             ; Inicio de asignacion
192
193                 ; Valor de Oper1 en R1
194
195                 MOVE #1, .R1
196
197                 ; Direccion de Res en R3
198

```

```

199             ADD #1, .IY
200             MOVE .A, .R3
201             MOVE .R1, [.R3] ; Valor de Oper1(R1) a Res(direccion a donde
202
203             ; Valor de Oper1 en R1
204
205             ADD #1, .IY
206             MOVE .A, .R9
207             MOVE [.R9], .R1
208
209             ; Direccion de Res en R3
210
211             ADD #0, .IY
212             MOVE .A, .R3
213             MOVE .R1, [.R3] ; Valor de Oper1(R1) a Res(direccion a donde
214
215             ; Fin de asignacion
216
217
218             ; ---- Inicio de llamada a funcion
219
220             ; Secuencia de llamada
221             ADD #tamRAFunMain, .IX
222             MOVE #dirRet1_Fun0_prin, [.A]
223             ADD #tamRAFunMain, .IX
224             MOVE .A, .IX
225             BR /EtiqFun0_prin
226
227             ; Secuencia de retorno
228 dirRet1_Fun0_prin: NOP
229                     SUB .IX, #tamRAFunMain
230                     MOVE .A, .IX
231
232             ; ---- Fin de llamada a funcion
233
234             HALT
235
236             ; Fin de código del main
237
238 tamRAFun0_prin: EQU 69
239 tamRAFunMain: EQU 2
240 beginED: RES 2
241 cad0_n: DATA "\n"
242 beginStack: NOP
243 END

```

10.5 Prueba 5 - test llamadas a funciones

Contenido de la prueba:

```
1 function boolean f(number n, boolean b){
2     b = true;
3     return b;
4 }
5 function number h(string s){
6     alert("Me ha llegado esta cadena: \n");
7     alert(s);
8     alert("\n");
9     return 23-2;
10 }
11 function number dameNum(string s, boolean b){
12     if(f(21,b))
13         return h(s);
14     return 0;
15 }
16 let string in;
17 alert("Introduce una cadena para transmitir a una funcion anidada \n");
18 input(in);
19 alert("Transmitiendo la cadena... \n");
20 dameNum(in,false);
```

Contenido del CI:

```
1 ; ----- Codigo de las funciones -----
2
3 ; ----- Inicio de funcion
4 [:, #EtiqFun0_f, , ]
5
6 ; Inicio de asignacion
7 [=EL, 1, , ~Temp0]
8 [=EL, ~Temp0, , b]
9 ; Fin de asignacion
10
11 [returnEL, b, , ]
12
13 [returnVoid, , , ]
14 ; ----- Fin de funcion
15
16
17 ; ----- Inicio de funcion
18 [:, #EtiqFun1_h, , ]
19
20 ; Inicio de llamada a alert
21 [=Cad, "Me ha llegado esta cadena: \n", , ~Temp1]
22 [alertCad, ~Temp1, , ]
```

```

23 ; Fin de llamada a alert
24
25
26 ; Inicio de llamada a alert
27 [alertCad, s, , ]
28 ; Fin de llamada a alert
29
30
31 ; Inicio de llamada a alert
32 [=Cad, "\n", , ~Temp2]
33 [alertCad, ~Temp2, , ]
34 ; Fin de llamada a alert
35
36
37 ; Inicio de resta aritmetica
38 [=EL, 23, , ~Temp3]
39 [=EL, 2, , ~Temp4]
40 [=-, ~Temp3, ~Temp4, ~Temp5]
41 ; Fin de resta aritmetica
42
43 [returnEL, ~Temp5, , ]
44
45 [returnVoid, , , ]
46 ; ----- Fin de funcion
47
48
49 ; ----- Inicio de funcion
50 [:, #EtiqFun2_dame, , ]
51
52 ; ---- Inicio de if simple
53
54 ; Inicio de condicion
55
56 ; ---- Inicio de llamada a funcion
57
58 ; Inicio de asignacion de literales en temporales
59 [=EL, 21, , ~Temp6]
60 ; Fin de asignacion de literales en temporales
61
62
63 ; Inicio de paso de parámetros
64 [paramEL, ~Temp6, , ]
65 [paramEL, b, , ]
66 ; Fin de paso de parámetros
67
68 [callValueEL, #EtiqFun0_f, , ~Temp7]
69 ; ---- Fin de llamada a funcion
70
71 ; Fin de condicion

```

```

72
73 [if=goto, ~Temp7, 0, #Etiq0]
74
75 ; Inicio de sentencia
76
77 ; ---- Inicio de llamada a funcion
78
79 ; Inicio de paso de parámetros
80 [paramCad, s, , ]
81 ; Fin de paso de parámetros
82
83 [callValueEL, #EtiqFun1_h, , ~Temp8]
84 ; ---- Fin de llamada a funcion
85
86 [returnEL, ~Temp8, , ]
87 ; Fin de sentencia
88
89 [:, #Etiq0, , ]
90 ; ---- Fin de if simple
91
92 [=EL, 0, , ~Temp9]
93 [returnEL, ~Temp9, , ]
94
95 [returnVoid, , , ]
96 ; ----- Fin de funcion
97
98 ; ----- Fin de codigo de las funciones-----
99
100 [=Cad, "", , in]
101
102 ; Inicio de llamada a alert
103 [=Cad, "Introduce una cadena para transmitir a una funcion anidada \n", , ~Temp10]
104 [alertCad, ~Temp10, , ]
105 ; Fin de llamada a alert
106
107
108 ; Inicio de llamada a input
109 [inputCad, , , in]
110 ; Fin de llamada a input
111
112
113 ; Inicio de llamada a alert
114 [=Cad, "Transmitiendo la cadena... \n", , ~Temp11]
115 [alertCad, ~Temp11, , ]
116 ; Fin de llamada a alert
117
118
119 ; ---- Inicio de llamada a funcion
120

```

```

121 ; Inicio de asignacion de literales en temporales
122 [=EL, 0, , ~Temp12]
123 ; Fin de asignacion de literales en temporales
124
125
126 ; Inicio de paso de parámetros
127 [paramCad, in, , ]
128 [paramEL, ~Temp12, , ]
129 ; Fin de paso de parámetros
130
131 [callValueEL, #EtiqFun2_dame, , ~Temp13]
132 ; ---- Fin de llamada a funcion

```

Contenido del CO:

```

1          ORG 0
2          MOVE #beginED, .IY
3          MOVE #beginStack, .IX
4          BR /main
5 ; ----- Codigo de las funciones -----
6
7
8 ; ----- Inicio de funcion
9 EtiqFun0_f:      NOP
10
11
12 ; Inicio de asignacion
13 ; Valor de Oper1 en R1
14          MOVE #1, .R1
15 ; Direccion de Res en R3
16          ADD #3, .IX
17          MOVE .A, .R3
18          MOVE .R1, [.R3] ; Valor de Oper1(R1) a Res(direccion a donde
19 ; Valor de Oper1 en R1
20          ADD #3, .IX
21          MOVE .A, .R9
22          MOVE [.R9], .R1
23 ; Direccion de Res en R3
24          ADD #2, .IX
25          MOVE .A, .R3
26          MOVE .R1, [.R3] ; Valor de Oper1(R1) a Res(direccion a donde
27 ; Fin de asignacion
28
29 ;Valor a devolver en .R8
30          ADD #2, .IX
31          MOVE .A, .R9
32          MOVE [.R9], .R8
33          BR [.IX]
34

```

```

35     ; ----- Fin de funcion
36
37
38
39     ; ----- Inicio de funcion
40 EtiqFun1_h:      NOP
41
42
43     ; Inicio de llamada a alert
44             MOVE #cad0_Meha, .R1
45             ADD #65, .IX
46             MOVE .A, .R3
47
48
49     ; Inicio bucle de copia de cadena
50 copia0:      NOP
51             MOVE [.R1], .R9
52             MOVE .R9, [.R3]
53             ADD #1, .R1
54             MOVE .A, .R1
55             ADD #1, .R3
56             MOVE .A, .R3
57             CMP #0, .R9
58             BNZ /copia0
59     ; Fin bucle de copia de cadena
60
61             ADD #65, .IX
62             MOVE .A, .R9
63             WRSTR [.R9]
64     ; Fin de llamada a alert
65
66
67
68     ; Inicio de llamada a alert
69             ADD #1, .IX
70             MOVE .A, .R9
71             WRSTR [.R9]
72     ; Fin de llamada a alert
73
74
75
76     ; Inicio de llamada a alert
77             MOVE #cad1_n, .R1
78             ADD #129, .IX
79             MOVE .A, .R3
80
81
82     ; Inicio bucle de copia de cadena
83 copia1:      NOP

```



```

84         MOVE  [.R1], .R9
85         MOVE  .R9, [.R3]
86         ADD   #1, .R1
87         MOVE  .A, .R1
88         ADD   #1, .R3
89         MOVE  .A, .R3
90         CMP   #0, .R9
91         BNZ   /copia1
92 ; Fin bucle de copia de cadena
93
94         ADD   #129, .IX
95         MOVE  .A, .R9
96         WRSTR [.R9]
97 ; Fin de llamada a alert
98
99
100
101 ; Inicio de resta aritmetica
102 ; Valor de Oper1 en R1
103         MOVE  #23, .R1
104 ; Direccion de Res en R3
105         ADD   #193, .IX
106         MOVE  .A, .R3
107         MOVE  .R1, [.R3] ; Valor de Oper1(R1) a Res(direccion a donde
108 ; Valor de Oper1 en R1
109         MOVE  #2, .R1
110 ; Direccion de Res en R3
111         ADD   #194, .IX
112         MOVE  .A, .R3
113         MOVE  .R1, [.R3] ; Valor de Oper1(R1) a Res(direccion a donde
114         ADD   #193, .IX
115         MOVE  .A, .R9
116         MOVE  [.R9], .R1
117         ADD   #194, .IX
118         MOVE  .A, .R9
119         MOVE  [.R9], .R2
120         ADD   #195, .IX
121         MOVE  .A, .R3
122         SUB   .R1, .R2
123         MOVE  .A, [.R3]
124 ; Fin de resta aritmetica
125
126 ;Valor a devolver en .R8
127         ADD   #195, .IX
128         MOVE  .A, .R9
129         MOVE  [.R9], .R8
130         BR    [.IX]
131
132 ; ----- Fin de funcion

```

```

133
134
135
136 ; ----- Inicio de funcion
137 EtiqFun2_dame:      NOP
138
139
140 ; ---- Inicio de if simple
141
142
143 ; Inicio de condicion
144
145
146 ; ---- Inicio de llamada a funcion
147
148
149 ; Inicio de asignacion de literales en temporales
150 ; Valor de Oper1 en R1
151             MOVE #21, .R1
152 ; Direccion de Res en R3
153             ADD #66, .IX
154             MOVE .A, .R3
155             MOVE .R1, [.R3] ; Valor de Oper1(R1) a Res(direccion a donde
156 ; Fin de asignacion de literales en temporales
157
158
159
160 ; Inicio de paso de parámetros
161             ADD #66, .IX
162             MOVE .A, .R1
163             ADD #70, .IX
164             ADD #1, .A ; .A contiene la dirección del parametro alojado e
165             MOVE [.R1], [.A]
166             ADD #65, .IX
167             MOVE .A, .R1
168             ADD #70, .IX
169             ADD #2, .A ; .A contiene la dirección del parametro alojado e
170             MOVE [.R1], [.A]
171 ; Fin de paso de parámetros
172
173
174 ; Secuencia de llamada
175             ADD #tamRAFun2_dame, .IX
176             MOVE #dirRetO_Fun0_f, [.A]
177             ADD #tamRAFun2_dame, .IX
178             MOVE .A, .IX
179             BR /EtqFun0_f
180
181 ; Secuencia de retorno

```

```

182 dirRet0_Fun0_f:      NOP
183                      SUB  .IX, #tamRAFun2_dame
184                      MOVE  .A, .IX
185                      ADD   #67, .IX
186                      MOVE  .A, .R3
187                      MOVE  .R8, [.R3]
188      ; ---- Fin de llamada a funcion
189
190      ; Fin de condicion
191
192                      ADD   #67, .IX
193                      MOVE  .A, .R9
194                      MOVE  [.R9], .R1
195                      MOVE  #0, .R2
196                      CMP   .R1, .R2
197                      BZ    /Etiqu0
198
199
200      ; Inicio de sentencia
201
202
203      ; ---- Inicio de llamada a funcion
204
205
206      ; Inicio de paso de parámetros
207                      ADD   #1, .IX
208                      MOVE  .A, .R1
209                      ADD   #70, .IX
210                      ADD   #1, .A ; .A contiene la dirección del parametro alojado e
211                      MOVE  .A, .R3
212
213
214      ; Inicio bucle de copia de cadena
215 copia2:      NOP
216                      MOVE  [.R1], .R9
217                      MOVE  .R9, [.R3]
218                      ADD   #1, .R1
219                      MOVE  .A, .R1
220                      ADD   #1, .R3
221                      MOVE  .A, .R3
222                      CMP   #0, .R9
223                      BNZ   /copia2
224      ; Fin bucle de copia de cadena
225
226      ; Fin de paso de parámetros
227
228
229      ; Secuencia de llamada
230                      ADD   #tamRAFun2_dame, .IX

```

```

231             MOVE  #dirRet1_Fun1_h, [.A]
232             ADD   #tamRAFun2_dame, .IX
233             MOVE  .A, .IX
234             BR    /EtiquFun1_h
235
236     ; Secuencia de retorno
237 dirRet1_Fun1_h:  NOP
238                 SUB   .IX, #tamRAFun2_dame
239                 MOVE  .A, .IX
240                 ADD   #68, .IX
241                 MOVE  .A, .R3
242                 MOVE  .R8, [.R3]
243     ; ---- Fin de llamada a funcion
244
245     ;Valor a devolver en .R8
246                 ADD   #68, .IX
247                 MOVE  .A, .R9
248                 MOVE  [.R9], .R8
249                 BR    [.IX]
250     ; Fin de sentencia
251
252 Etiqu0:          NOP
253     ; ---- Fin de if simple
254
255     ; Valor de Oper1 en R1
256                 MOVE  #0, .R1
257     ; Direccion de Res en R3
258                 ADD   #69, .IX
259                 MOVE  .A, .R3
260                 MOVE  .R1, [.R3] ; Valor de Oper1(R1) a Res(direccion a donde
261     ;Valor a devolver en .R8
262                 ADD   #69, .IX
263                 MOVE  .A, .R9
264                 MOVE  [.R9], .R8
265                 BR    [.IX]
266
267     ; ----- Fin de funcion
268
269     ; ----- Fin de codigo de las funciones-----
270
271
272
273     ; Inicio de código del main
274 main:           NOP
275                 MOVE  #cad2_, .R1
276                 ADD   #0, .IY
277                 MOVE  .A, .R3
278
279

```

```

280      ; Inicio bucle de copia de cadena
281 copia3:      NOP
282              MOVE  [.R1], .R9
283              MOVE  .R9, [.R3]
284              ADD   #1, .R1
285              MOVE  .A, .R1
286              ADD   #1, .R3
287              MOVE  .A, .R3
288              CMP   #0, .R9
289              BNZ   /copia3
290      ; Fin bucle de copia de cadena
291
292
293
294      ; Inicio de llamada a alert
295              MOVE  #cad3_Intr, .R1
296              ADD   #64, .IY
297              MOVE  .A, .R3
298
299
300      ; Inicio bucle de copia de cadena
301 copia4:      NOP
302              MOVE  [.R1], .R9
303              MOVE  .R9, [.R3]
304              ADD   #1, .R1
305              MOVE  .A, .R1
306              ADD   #1, .R3
307              MOVE  .A, .R3
308              CMP   #0, .R9
309              BNZ   /copia4
310      ; Fin bucle de copia de cadena
311
312              ADD   #64, .IY
313              MOVE  .A, .R9
314              WRSTR  [.R9]
315      ; Fin de llamada a alert
316
317
318
319      ; Inicio de llamada a input
320              ADD   #0, .IY
321              MOVE  .A, .R9
322              INSTR  [.R9]
323      ; Fin de llamada a input
324
325
326
327      ; Inicio de llamada a alert
328              MOVE  #cad4_Tran, .R1

```

```

329             ADD #128, .IY
330             MOVE .A, .R3
331
332
333             ; Inicio bucle de copia de cadena
334 copia5:      NOP
335             MOVE [.R1], .R9
336             MOVE .R9, [.R3]
337             ADD #1, .R1
338             MOVE .A, .R1
339             ADD #1, .R3
340             MOVE .A, .R3
341             CMP #0, .R9
342             BNZ /copia5
343             ; Fin bucle de copia de cadena
344
345             ADD #128, .IY
346             MOVE .A, .R9
347             WRSTR [.R9]
348             ; Fin de llamada a alert
349
350
351
352             ; ---- Inicio de llamada a funcion
353
354
355             ; Inicio de asignacion de literales en temporales
356             ; Valor de Oper1 en R1
357             MOVE #0, .R1
358             ; Direccion de Res en R3
359             ADD #192, .IY
360             MOVE .A, .R3
361             MOVE .R1, [.R3] ; Valor de Oper1(R1) a Res(direccion a donde e
362             ; Fin de asignacion de literales en temporales
363
364
365
366             ; Inicio de paso de parámetros
367             ADD #0, .IY
368             MOVE .A, .R1
369             ADD #194, .IX
370             ADD #1, .A ; .A contiene la dirección del parametro alojado e
371             MOVE .A, .R3
372
373
374             ; Inicio bucle de copia de cadena
375 copia6:      NOP
376             MOVE [.R1], .R9
377             MOVE .R9, [.R3]

```

```

378             ADD #1, .R1
379             MOVE .A, .R1
380             ADD #1, .R3
381             MOVE .A, .R3
382             CMP #0, .R9
383             BNZ /copia6
384 ; Fin bucle de copia de cadena
385
386             ADD #192, .IY
387             MOVE .A, .R1
388             ADD #194, .IX
389             ADD #65, .A ; .A contiene la dirección del parametro alojado
390             MOVE [.R1], [.A]
391 ; Fin de paso de parámetros
392
393
394 ; Secuencia de llamada
395             ADD #tamRAFunMain, .IX
396             MOVE #dirRet2_Fun2_dame, [.A]
397             ADD #tamRAFunMain, .IX
398             MOVE .A, .IX
399             BR /EtiqFun2_dame
400
401 ; Secuencia de retorno
402 dirRet2_Fun2_dame: NOP
403                 SUB .IX, #tamRAFunMain
404                 MOVE .A, .IX
405                 ADD #193, .IY
406                 MOVE .A, .R3
407                 MOVE .R8, [.R3]
408 ; ---- Fin de llamada a funcion
409
410 ; Fin de código del main
411
412             HALT
413 tamRAFun0_f: EQU 4
414 tamRAFun1_h: EQU 196
415 tamRAFun2_dame: EQU 70
416 tamRAFunMain: EQU 194
417 beginED: RES 194
418 cad0_Meha: DATA "Me ha llegado esta cadena: \n"
419 cad1_n: DATA "\n"
420 cad2_: DATA ""
421 cad3_Intr: DATA "Introduce una cadena para transmitir a una funcion anid
422 cad4_Trans: DATA "Transmitiendo la cadena... \n"
423 beginStack: NOP
424             END

```

11 | Referencias

1. *Documentación librería SLY*
<https://sly.readthedocs.io/en/latest/>
2. *Generador de tabla LR(1)*
<http://jsmachines.sourceforge.net/machines/lr1.html>