Memoria
Final
-
Práctica de Traductores de Lenguajes

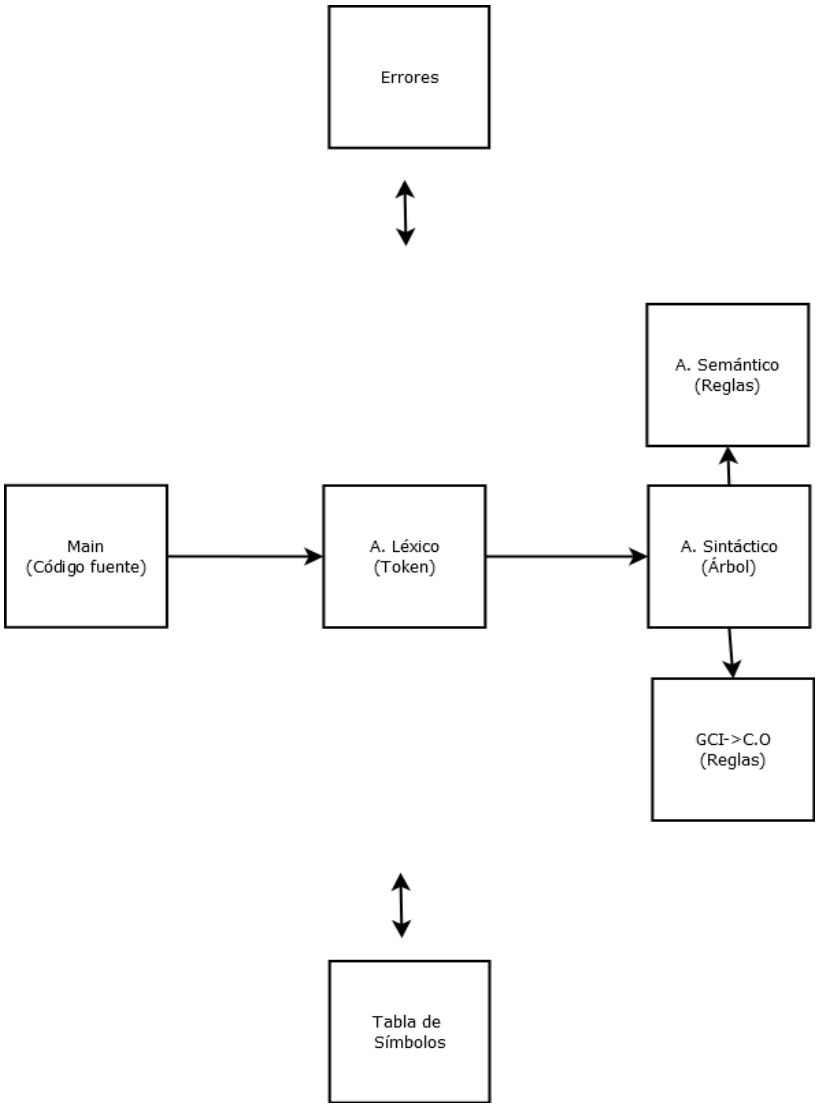
Alex Calvo Vergara(b190207)
Inês De Almeida Pissarra (e220907)

Grupo 2-jun-12:15

Parte específica:

- Operadores: +, <, &&
- Comentarios: Comentario de bloque (/* */)
- Cadenas: comillas simples (' ')
- Sentencias: if, while
- Operadores especiales: Asignación con resto (%=)

Descripción del diseño del Traductor



En el diseño del GCI y CO, hemos incorporado dichas funcionalidades al propio A. Sintáctico en la clase Analizadores/AnalizadorSintactico.java.
Tras aplicar cada regla correspondiente se genera un cuarteto y el código objeto correspondiente.

Para diferenciar la parte de la ejecución del código principal y de las funciones hemos utilizado dos ficheros de textos distintos, cada vez que entramos en la declaración de una función pasamos al fichero de funciones (Ficheros/ensambladorFun.txt) para generar ahi el C.O.
Finalmente con la función `imprimirCO_final`, añadimos el código principal de ensamblador y añadimos el código correspondiente de Ficheros/ensambladorMain.txt y Ficheros/ensambladorFun.txt.

Para la gestión de las cadenas diferenciamos las temporales que apuntan a la dirección donde se encuentra las cadenas DATA, de las temporales y variables que no apuntan a ese lugar sino que tienen el valor de las cadenas en esa misma dirección.

Los 5 casos de prueba se encuentan en la carpeta Ficheros

Para solventar el problema de los direccionamientos superiores a 127, sumamos con ADD el valor del desplazamiento más la dirección de IX o IY según corresponda. Ese resultado nos indica la dirección a donde queremos apuntar y trabajamos con ella sin problemas.
Ej. ADD #170, .IX

Generador de Código Intermedio

[1] P ->
{ B.return, P.return = false; B.valorRet, P.valorRet = false } (R56) BP
{ POP(2) } (R1)

[2] P ->
{ F.return, P.return = false; B.valorRet, P.valorRet = false } (R57) FP
{ POP(2) } (R2)

[3] P -> eof
{ POP(1) } (R3)

[4] B ->
{ zonaDeclaracion: true } (R50)
let T id ;
{ if getTipoDecl(id.posTS) != null then
B.tipo = error(No se puede redeclarar variables);
else
B.tipo = ok;
anadirTipoYDespl(id.posTS, T.Tipo);
zonaDeclaracion: false; POP(4) } (R4)

[5] B -> if (E)
{ S.return = B.return; S.valorRet = B.valorRet;
S.despues = nuevaetiq();
emite('if', E.lugar, ':=', 0, 'goto', S.despues);} (R58)
S
{ if E.tipo != logico then
B.tipo = error(La condicion del IF no es de tipo logica);
else if S.tipo != ok then
B.tipo = error;
else
B.tipo = ok;
emite(S.despues, ':');
POP(5) } (R5)

[6] B ->
{ S.return = B.return, S.valorRet = B.valorRet } (R59) S
{ B.tipo = S.tipo; POP(1) } (R6)

[7] B -> while (
 { **B.inicio = nuevaetiq();**
  B.despues = nuevaetiq();
  emite(B.inicio, ':') } (R66)
 E){
 { C.return = B.return; C.valorRet = B.valorRet;
  emite('if', E.lugar, '=', 0, 'goto', B.despues) } (R60)
C}
 { if E.tipo != logico then
        B.tipo = error(La condicion del WHILE no es de tipo logica);
  else if C.tipo != ok then
        B.tipo = error
  else
        B.tipo = ok
 emite('goto', B.inicio);
 emite(B.despues, ':');
 POP(7) } (R7)

[8] T -> int
 { T.tipo = entero; POP(1) } (R8)

[9] T -> string
 { T.tipo = cadena; POP(1) } (R9)

[10] T -> boolean
 { T.tipo = logico; POP(1) } (R10)

[11] S -> id
 { N.posTS = id.posTS } (R64) N;
 { if N.tipo == error then
        S.tipo = error
 else if getTipoID(id.posTS) != N.tipo then
        S.tipo = error(No se puede operar con valores de distinto tipo);
 else
        S.tipo = ok
 POP(3) } (R11)

[12] S -> print (E) ;
 { if E.tipo == cadena || E.tipo == entero then
        S.tipo = ok
        emite('print', E.lugar);
  else
        S.tipo = error(Print solo puede recibir valores de tipo cadena o entero);
  POP(5) } (R12)

```
[13] S -> input ( id ) ;
 { if getTipoID(id.posTS) == cadena || getTipoID(id.posTS) == entero then
        S.tipo = ok
        emite('input', buscarLugarTS(id.pos));
    else
        S.tipo = error(Input solo puede recibir variables de tipo cadena o entero);
   POP(5) } (R13)


[14] S -> return X ;
 { if S.return == true then
        if S.valorRet == X.tipo then
                S.tipo = ok
                if S.valorRet != vacio then
                        emite('return', X.lugar);
                else
                        emite('return');
        else
                S.tipo = error(El valor de retorno es de tipo ${X.tipo} cuando se esperaba un
valor de tipo ${S.valorRet});
   else
        S.tipo = error(La sentencia return solo puede utilizarse dentro de funciones);
POP(3) } (R14)


[15] N -> = E
 { if getTipoID(N.posTS) == funcion then
        N.tipo = error(El operador '=' no puede recibir funciones como asignacion);
   else
        if E.tipo == funcion then
                N.tipo = E.valorRet
        else
                N.tipo = E.tipo
        emite(buscarLugarTS(N.posTS), ':=', E.lugar);
   POP(2) } (R15)

[16] N -> ( L )
 { if getTipoID(N.posTS) == funcion then
        if compararNumParams(N.posTS, L.tipo) then
                if compararParams(N.posTS, L.tipo) then
                        N.tipo = funcion
                        for i:=1 to L.long
                                emite('param', L.param[i]);
                        emite('call', buscarEtiqTS(N.posTS));
                else
                        N.tipo = error(La funcion ${getCabecera(N.posTS)} no es aplicable
para la cabecera (${L.tipo}));
        else
                N.tipo = error(La funcion ${getCabecera(N.posTS)} no es aplicable para la
cabecera (${L.tipo}));
else
        N.tipo = error(La variable ${getNombreID(N.posTS)} no puede ser utilizada como
una funcion);

POP(3) } (R16)
```

```
[17] N -> %= E
 { if getTipoID(N.posTS) == funcion then
        N.tipo = error(El operador '%=' no puede recibir funciones como asignacion);
   else
        if E.tipo == funcion then
              N.tipo = E.valorRet
        else
              N.tipo = E.tipo
        emite(buscarLugarTS(N.posTS), ':=', buscarLugarTS(N.posTS), '%=', E.lugar);
   POP(2) } (R17)


[18] X -> E
 { X.tipo = E.tipo;
   X.lugar = E.lugar;
   POP(1) } (R18)

[19] X -> lambda
 { X.tipo = vacio } (R19)

[20] C1 ->
 { B.return, C2.return = C1.return; B.valorRet, C2.valorRet = C1.valorRet } (61)
B C2
 { if B.tipo == ok && (C2.tipo == vacio || C2.tipo == ok) then
        C1.tipo = ok
   else
        C1.tipo = error
   POP(2) } (R20)

[21] C -> lambda
 { C.tipo = vacio } (R21)

[22] L -> E Q
 { if E.tipo == funcion && E.valorRet == vacio then
        L.tipo = error(No se permiten funciones de tipo void como parametro)
   else if E.tipo = error || Q.tipo = error then
        L.tipo = error
   else
        if Q.tipo == vacio then
              if E.tipo == funcion then
                    L.tipo = E.valorRet
                    L.lugar = E.valorRet
              else
                    L.tipo = E.tipo
                    L.lugar = E.lugar
              L.param = E.lugar
              L.long = 1
        else
              if E.tipo == funcion then
                    L.tipo = E.valorRet x Q.tipo
              else
                    L.tipo = E.tipo x Q.tipo
              L.param = E.lugar ⊕ Q.param
              L.long = 1 + Q.long
POP(2) } (R22)
```

[23] L -> lambda
{ L.tipo = vacio } (R23)

[24] Q1 -> , E Q2
{ if E.tipo == funcion && E.valorRet == vacio then
Q1.tipo = error(No se permiten funciones de tipo void como parametro);
else if E.tipo == error || Q2.tipo == error then
Q1.tipo = error
else
if Q2.tipo == vacio then
if E.tipo == funcion then
Q1.tipo = E.valorRet
else
Q1.tipo = E.tipo
Q1.param = E.lugar
Q1.long = 1
else
if E.tipo == funcion then
Q1.tipo = E.valorRet x Q2.tipo
else
Q1.tipo = E.tipo x Q2.tipo
Q1.param = E.lugar ⊕ Q2.param
Q1.long = 1 + Q2.long
POP(3) } (R24)

[25] Q -> lambda
{ Q.tipo = vacio } (R25)

[26] F -> function
{ zonaDeclaracion: true } (R51) id H
{ TS = crearTabla() } (R52)
(A
{ zonaDeclaracion: false; anadirDatosFun(id.posTS, H.tipo, A.tipo) } (R53)){
{ C.return = true; C.valorRet = H.tipo;
emite(buscarEtiqTS(id.pos), ':') } (R62)
C
{ C.return = false; C.valorRet = false;
emite('return') } (R63)
}
{ borrarTabla(); POP(9) } (R26)

[27] H -> T
{ H.tipo = T.tipo; POP(1) } (R27)

[28] H -> lambda
{ H.tipo = vacio } (R28)

[29] A -> T id
{ anadirTipoYDespl(id.posTS, T.tipo) } (R54)
K
{ if K.tipo == vacio then
A.tipo = T.tipo
else
A.tipo = T.tipo x K.tipo POP(3) } (R29)

[30] A -> lambda
{ A.tipo = vacio } (R30)

[31] K -> , T id
{ anadirTipoYDespl(id.posTS, T.tipo) } (R55)
K2
{ if K2.tipo == vacio then
K.tipo = T.tipo
else
K.tipo = T.tipo x K2.tipo POP(4) } (R31)

[32] K -> lambda
{ K.tipo = vacio } (R32)

[33] E -> W E2
{ if E2.tipo == vacio then
E.tipo = W.tipo
E.valorRet = W.valorRet
E.lugar = W.lugar
else if W.tipo == logico || (W.tipo == funcion && W.valorRet == logico) then
if E2.tipo == error then
E.tipo == error
else
E.tipo = logico
E.valorRet = W.valorRet
E.lugar = nuevatemp();
emite(E.lugar, ':=', W.lugar, '&&', E2.lugar[1]);
for i:=2 to E2.elems
emite(E.lugar, ':=', E.lugar, '&&', E2.lugar[i]);
else
E.tipo = error(El operador '&&' solo puede recibir valores de tipo logico);
POP(2) } (R33)

[34] E2 -> lambda
{ E2.tipo = vacio }

[35] E2 -> && W E2(2)
{ if W.tipo == logico || (W.tipo == funcion && W.valorRet == logico) then
if E2(2).tipo == error then
E2.tipo = error
else
E2.tipo = logico
if E2(2).tipo = vacio then
E2.lugar = W.lugar
E2.elems = 1
else
E2.lugar = W.lugar ⊕ E2(2).lugar
E2.elems = 1 + E2(2).elems
else
E2.tipo = error(El operador '&&' solo puede recibir valores de tipo logico);
POP(3) } (R35)

[36] W -> R W2
{ if W2.tipo == vacio then
W.tipo = R.tipo
W.valorRet = R.valorRet
W.lugar = R.lugar
else if R.tipo == entero || (R.tipo == funcion && R.valorRet == entero)
if W2.tipo == error then
W.tipo = error
else
W.tipo = logico
W.valorRet = R.valorRet
W.lugar = nuevatemp();
nueva_etiq = nuevaetiq();
nueva_etiq2 = nuevaetiq();
emite('if', R.lugar, '<', W2.lugar[1], 'goto', nueva_etiq);
emite(W.lugar, ':=', 0);
emite('goto', nueva_etiq2);
emite(nueva_etiq, ':');
emite(W.lugar, ':=', 1);
emite(nueva_etiq2, ':');
for i:=2 to R2.elems
nueva_etiq = nuevaetiq();
nueva_etiq2 = nuevaetiq();
emite('if', W.lugar, '<', W2.lugar[i], 'goto', nueva_etiq);
emite(W.lugar, ':=', 0);
emite('goto', nueva_etiq2);
emite(nueva_etiq, ':');
emite(W.lugar, ':=', 1);
emite(nueva_etiq2, ':');
else
W.tipo = error(El operador '<' solo puede recibir valores de tipo entero);
POP(2) } (R36)

[37] W2 -> lambda { W2.tipo = vacio }

[38] W2 -> < R W2(2)
{ if R.tipo == entero || (R.tipo == funcion && R.valorRet == entero) then
if W2(2).tipo == error then
W2.tipo = error
else
W2.tipo = entero
if W2(2).tipo = vacio then
W2.lugar = R.lugar
W2.elems = 1
else
W2.lugar = R.lugar ⊕ W2(2).lugar
W2.elems = 1 + W2(2).elems
else
W2.tipo = error(El operador '<' solo puede recibir valores de tipo entero);
POP(3)} (R38)

[39] R -> V R2
{ if R2.tipo == vacio then
R.tipo = V.tipo
R.valorRet = V.valorRet
R.lugar = V.lugar
else if V.tipo == entero || (V.tipo == funcion && V.valorRet == entero) then
if R2.tipo == error then
R.tipo = error
else
R.tipo = entero
R.valorRet = V.valorRet
R.lugar = nuevatemp();
emite(R.lugar, ':=', V.lugar, '+', R2.lugar[1]);
for i:=2 to R2.elems
emite(R.lugar, ':=', R.lugar, '+', R2.lugar[i]);
else
R.tipo = error(El operador '+' solo puede recibir valores de tipo entero):
POP(2) } (R39)

[40] R2 -> lambda
{ R2.tipo = vacio } (R40)

[41] R2 -> + V R2(2)
{ if V.tipo == entero || (V.tipo == funcion && V.valorRet == entero)
if R2(2).tipo == error then
R2.tipo = error
else
R2.tipo = entero
if R2(2).tipo = vacio then
R2.lugar = V.lugar
R2.elems = 1
else
R2.lugar = V.lugar ⊕ R2(2).lugar
R2.elems = 1 + R2(2).elems
else
R2.tipo = error(El operador '+' solo puede recibir valores de tipo entero);
POP(3) } (R41)

[42] V -> id
{ Z.posTS = id.posTS } (R65)
Z
{ if getTipoID(id.posTS) == funcion then
if Z.tipo == vacio then
V.tipo = error(La funcion ${getCabecera(id.posTS)} no puede ser utilizada como una variable);
else
V.tipo = getTipoID(id.posTS)
V.valorRet = valorRet(id.posTS)
V.lugar = Z.lugar

else
if Z.tipo != vacio then
V.tipo = error(La variable ${getNombreID(id.posTS)} no puede ser utilizada como una funcion);
else
V.tipo = getTipoID(id.posTS)
V.valorRet = false
V.lugar = buscarLugarTS(id.pos)
POP(2); } (R42)

[43] V -> (E)
{ V.tipo = E.tipo; V.valorRet = false; POP(3);
V.lugar = E.lugar } (R43)

[44] Z -> (L)
{ if getTipoID(Z.posTS) == funcion then
if compararNumParams(Z.posTS, L.tipo)
if compararParams(Z.posTS, L.tipo) then
Z.tipo = funcion
Z.lugar = nuevatemp();
for i:=1 to L.long
emite('param', L.param[i]);
emite(Z.lugar, ':=', 'call', buscarEtqTS(id.pos));
else
Z.tipo = error(La funcion ${getCabecera(Z.posTS)} no es aplicable para la cabecera (${L.tipo}));
else
Z.tipo = error(La funcion ${getCabecera(Z.posTS)} no es aplicable para la cabecera (${L.tipo}));
else
Z.tipo = error
POP(3) } (R44)

[45] Z -> lambda
{ Z.tipo = vacio } (R45)
[46] V -> entero
{ V.tipo = entero; V.valorRet = false; POP(1);
E.lugar = nuevatemp();
emite(E.lugar, ':=', entero.valor); } (R46)

[47] V -> cadena
{ V.tipo = cadena; V.valorRet = false; POP(1);
E.lugar = nuevatemp();
emite(E.lugar, ':=c', cadena.valor); } (R47)

[48] V -> true
{ V.tipo = logico; V.valorRet = false; POP(1);
V.lugar = nuevatemp();
emite(V.lugar, ':=', 1); } (R48)

[49] V -> false
{ V.tipo = logico; V.valorRet = false; POP(1);
V.lugar = nuevatemp();
emite(V.lugar, ':=', 0); } (R49)

Generador de Código Final

(goto , , , etiq)	BR /etiq
(:, etiq, ,)	etiq:
(if=, op1, op2, etiq)	CMP op1, op2 BZ /etiq
(if<, op1, op2, etiq)	CMP op1, op2 BN /etiq

(+, op1, op2, res)	MOVE op1, .R0
	ADD .R0, op2
	MOVE .A, res

(&&, op1, op2, res)	MOVE op1, .R0
	AND .R0, op2
	MOVE .A, res

(%, op1, op2, res)	MOVE op1, .R0
	MOD .R0, op2
	MOVE .A, res

(:=, valor, , res) valor entero o lógico	MOVE #valor, res

(:=c, valor, , res)
valor cadena
	MOVE res, .R6
	MOVE #valor, .R7
	etiqBucle: CMP [.R7], #0
	BZ /etiqFin
	MOVE [.R7], [.R6]
	INC .R6
	INC .R7
	BR /etiqBucle
	etiqFin: MOVE [.R7], [.R6]

(=, op1, , res) op1 entero o lógico	MOVE op1, .R0 MOVE .R0, res

(=, op1, , res)
op1 cadena
	MOVE res, .R6
	MOVE op1, .R7
	etiqBucle: CMP [.R7], #0
	BZ /etiqFin
	MOVE [.R7], [.R6]
	INC .R6
	INC .R7
	BR /etiqBucle
	etiqFin: MOVE [.R7], [.R6]

(print , , , op) op entero	WRINT op
(print , , , op) op cadena	WRSTR op
(input, , , op) op entero	ININT op
(input, , , op) op cadena	INSTR op

(param, desp, , op)
op entero o lógico
	ADD #TAM_RA_llamador, .IX
	ADD #desp_param, .A
	MOVE op, [.A]

(param, desp, , op)
op cadena
	ADD #TAM_RA_llamador, .IX
	ADD #desp_param, .A
	MOVE .A, .R6
	MOVE op, .R7
	etiqBucle: CMP [.R7], #0
	BZ /etiqFin
	MOVE [.R7], [.R6]
	INC .R6
	INC .R7
	BR /etiqBucle
	etiqFin: MOVE [.R7], [.R6]

(call, etiq , ,)	ADD #TAM_RA_llamador, .IX
	MOVE #dir_ret, [.A]
	ADD #TAM_RA_llamador, .IX
	MOVE .A, .IX

	BR /etiq dir_ret: SUB .IX, #TAM_RA_llamador MOVE .A, .IX

(call, etiq, , res)	MOVE #dir_ret, #TAM_RA_llamador[.IX]
	ADD #TAM_RA_llamador, .IX
	MOVE .A, .IX
	BR /etiq
	dir_ret: SUB #TAM_RA_etiq, #despVD
	ADD .A, .IX
	MOVE [.A], .R9
	SUB .IX, #TAM_RA_llamador
	MOVE .A, .IX
	MOVE .R9, res

(return, , , op)	SUB #TAM_RA_funcion, #desp
	(desp = 1 para enteros o lógicos, 65 para cadenas)
	ADD .A, .IX
	MOVE op, [.A];
	BR [.IX]

(return, , ,)	BR [.IX]

Diseño de los Registros de Activación

Estado de la máquina
Parámetros
Variables Locales
Datos Temporales
Valor de Retorno

El tamaño de los registros de activación era calculado al final con el emite('return') utilizando un mapa `Map<String, Integer> tamRA_fun = new HashMap<String, Integer>();`
Al final de la ejecución se declaran los TAM_RA de cada función.
En el caso de crearse un RA a partir de main no se mueve el puntero .IX ya que está ya apuntando a la posición correspondiente, mientras se usa .IY para el RA de la ejecución principal con los datos globales.

ANEXO:

CASOS DE PRUEBA

Caso de prueba 1:

```
/* Programa de ejemplo 1 */

let boolean pasa;
let int num;

function hola () {
   print('Hola Mundo!');
   print(num);
}

pasa = true;
num = 10;
while (num < 20 && pasa) {
   num %= 2;
   if (num < 3)
   pasa = false;
   num = num + 1;
}
hola();
```

- Cuartetos:

```
(:, Ethola, , )
(:=c, "Hola Mundo!", , t1)
(print, , , t1)
(print, , , num)
(return, , , )
(:=, 1, , t2)
(=, t2, , pasa)
(:=, 10, , t3)
(=, t3, , num)
(:, etiq1, , )
(:=, 20, , t4)
(if<, num, t4, etiq3)
(:=, 0, , t5)
(goto, , , etiq4)
(:, etiq3, , )
(:=, 1, , t5)
(:, etiq4, , )
(&&, t5, pasa, t6)
(if=, t6, 0, etiq2)
(:=, 2, , t7)
(%, num, t7, num)
(:=, 3, , t8)
(if<, num, t8, etiq5)
```

```
(:=, 0, , t9)
(goto, , , etiq6)
(:, etiq5, , )
(:=, 1, , t9)
(:, etiq6, , )
(if=, t9, 0, etiq7)
(:=, 0, , t10)
(=, t10, , pasa)
(:, etiq7, , )
(:=, 1, , t11)
(+, num, t11, t12)
(=, t12, , num)
(goto, , , etiq1)
(:, etiq2, , )
(call, Ethola, , )
```

- Código Final:

```
ORG 0
MOVE #PILA, .IX
MOVE .IX, .SP
MOVE #DE, .IY
BR /main
Ethola:
ADD .IX, #1
MOVE #cad1, [.A]
ADD .IX, #1
MOVE [.A], .R1
WRSTR [.R1]
ADD .IY, #1
MOVE .A, .R1
WRINT [.R1]
BR [.IX]
main: NOP
ADD .IY, #2
MOVE #1, [.A]
ADD .IY, #2
MOVE [.A], .R0
ADD .IY, #0
MOVE .R0, [.A]
ADD .IY, #3
MOVE #10, [.A]
ADD .IY, #3
MOVE [.A], .R0
ADD .IY, #1
MOVE .R0, [.A]
etiq1:
ADD .IY, #4
MOVE #20, [.A]
ADD .IY, #1
MOVE [.A], .R0
ADD .IY, #4
```

```
CMP .R0, [.A]
BN /etiq3
ADD .IY, #5
MOVE #0, [.A]
BR /etiq4
etiq3:
ADD .IY, #5
MOVE #1, [.A]
etiq4:
ADD .IY, #6
MOVE .A, .R1
ADD .IY, #5
MOVE [.A], .R0
ADD .IY, #0
AND [.A], .R0
MOVE .A, [.R1]
ADD .IY, #6
CMP [.A], #0
BZ /etiq2
ADD .IY, #7
MOVE #2, [.A]
ADD .IY, #1
MOVE [.A], .R0
ADD .IY, #7
MOD .R0, [.A]
MOVE .A, .R1
ADD .IY, #1
MOVE .R1, [.A]
ADD .IY, #8
MOVE #3, [.A]
ADD .IY, #1
MOVE [.A], .R0
ADD .IY, #8
CMP .R0, [.A]
BN /etiq5
ADD .IY, #9
MOVE #0, [.A]
BR /etiq6
etiq5:
ADD .IY, #9
MOVE #1, [.A]
etiq6:
ADD .IY, #9
CMP [.A], #0
BZ /etiq7
ADD .IY, #10
MOVE #0, [.A]
ADD .IY, #10
MOVE [.A], .R0
ADD .IY, #0
MOVE .R0, [.A]
etiq7:
```

```
ADD .IY, #11
MOVE #1, [.A]
ADD .IY, #1
MOVE [.A], .R0
ADD .IY, #11
ADD [.A], .R0
MOVE .A, .R1
ADD .IY, #12
MOVE .R1, [.A]
ADD .IY, #12
MOVE [.A], .R0
ADD .IY, #1
MOVE .R0, [.A]
BR /etiq1
etiq2:
MOVE #dir_ret1, #0[.IX]
BR /Ethola
dir_ret1:
HALT
TAM_RA_main: EQU 13
TAM_RA_Ethola: EQU 66
cad1:
DATA "Hola Mundo!"
DE: RES 13
PILA: NOP
END
```

Caso de prueba 2:

```
/* Programa de ejemplo 2 */

let string nombre;

function saludo () {
   input(nombre);
   print('Hola ');
   print(nombre);
}

let int a;
input(a);
while(a < 10) {
   print('Numero muy bajo');
   input(a);
}

if(a < 11)
   print('Bien hecho!');

saludo();
```

- Cuartetos:

```
(:, Etsaludo, , )
(input, , , nombre)
(:=c, "Hola ", , t1)
(print, , , t1)
(print, , , nombre)
(return, , , )
(input, , , a)
(:, etiq1, , )
(:=, 10, , t2)
(if<, a, t2, etiq3)
(:=, 0, , t3)
(goto, , , etiq4)
(:, etiq3, , )
(:=, 1, , t3)
(:, etiq4, , )
(if=, t3, 0, etiq2)
(:=c, "Numero muy bajo", , t4)
(print, , , t4)
(input, , , a)
(goto, , , etiq1)
(:, etiq2, , )
(:=, 11, , t5)
(if<, a, t5, etiq5)
(:=, 0, , t6)
(goto, , , etiq6)
(:, etiq5, , )
(:=, 1, , t6)
(:, etiq6, , )
(if=, t6, 0, etiq7)
(:=c, "Bien hecho!", , t7)
(print, , , t7)
(:, etiq7, , )
(call, Etsaludo, , )
```

- Código Final:

```
ORG 0
MOVE #PILA, .IX
MOVE .IX, .SP
MOVE #DE, .IY
BR /main
Etsaludo:
ADD .IY, #0
MOVE .A, .R1
INSTR [.R1]
ADD .IX, #1
MOVE #cad1, [.A]
ADD .IX, #1
```

```
MOVE [.A], .R1
WRSTR [.R1]
ADD .IY, #0
MOVE .A, .R1
WRSTR [.R1]
BR [.IX]
main: NOP
ADD .IY, #65
MOVE .A, .R1
ININT[.R1]
etiq1:
ADD .IY, #66
MOVE #10, [.A]
ADD .IY, #65
MOVE [.A], .R0
ADD .IY, #66
CMP .R0, [.A]
BN /etiq3
ADD .IY, #67
MOVE #0, [.A]
BR /etiq4
etiq3:
ADD .IY, #67
MOVE #1, [.A]
etiq4:
ADD .IY, #67
CMP [.A], #0
BZ /etiq2
ADD .IY, #68
MOVE #cad2, [.A]
ADD .IY, #68
MOVE [.A], .R1
WRSTR [.R1]
ADD .IY, #65
MOVE .A, .R1
ININT[.R1]
BR /etiq1
etiq2:
ADD .IY, #133
MOVE #11, [.A]
ADD .IY, #65
MOVE [.A], .R0
ADD .IY, #133
CMP .R0, [.A]
BN /etiq5
ADD .IY, #134
MOVE #0, [.A]
BR /etiq6
etiq5:
ADD .IY, #134
MOVE #1, [.A]
etiq6:
```

```
ADD .IY, #134
CMP [.A], #0
BZ /etiq7
ADD .IY, #135
MOVE #cad3, [.A]
ADD .IY, #135
MOVE [.A], .R1
WRSTR [.R1]
etiq7:
MOVE #dir_ret1, #0[.IX]
BR /Etsaludo
dir_ret1:
HALT
TAM_RA_main: EQU 200
TAM_RA_Etsaludo: EQU 66
cad2:
DATA "Numero muy bajo"
cad3:
DATA "Bien hecho!"
cad1:
DATA "Hola "
DE: RES 200
PILA: NOP
END
```

Caso de prueba 3:

```
/* Programa de ejemplo 3 */

let int numero;
let string palabra;
let boolean cierto;

numero = 0;
cierto = true;
while (cierto && numero < 10) {
   numero = numero + 1;
   print('Iteracion producida');
}

/* Funcion 1 */
function modificar () {
   let int n; input(n);
   if (n < 0)
   cierto = false;
}

/* Funcion 2 */
function esCierto boolean () {
   return true;
}
```

```
/* Funcion 3 */
function noEsCierto boolean () {
    return false;
}
```

- Cuartetos:

```
(:=, 0, , t1)
(=, t1, , numero)
(:=, 1, , t2)
(=, t2, , cierto)
(:, etiq1, , )
(:=, 10, , t3)
(if<, numero, t3, etiq3)
(:=, 0, , t4)
(goto, , , etiq4)
(:, etiq3, , )
(:=, 1, , t4)
(:, etiq4, , )
(&&, cierto, t4, t5)
(if=, t5, 0, etiq2)
(:=, 1, , t6)
(+, numero, t6, t7)
(=, t7, , numero)
(:=c, "Iteracion producida", , t8)
(print, , , t8)
(goto, , , etiq1)
(:, etiq2, , )
(:, Etmodificar, , )
(input, , , n)
(:=, 0, , t9)
(if<, n, t9, etiq5)
(:=, 0, , t10)
(goto, , , etiq6)
(:, etiq5, , )
(:=, 1, , t10)
(:, etiq6, , )
(if=, t10, 0, etiq7)
(:=, 0, , t11)
(=, t11, , cierto)
(:, etiq7, , )
(return, , , )
(:, EtesCierto, , )
(:=, 1, , t12)
(return, t12, , )
(:, EtnoEsCierto, , )
(:=, 0, , t13)
(return, t13, , )
```

- Código Final:

```
ORG 0
MOVE #PILA, .IX
MOVE .IX, .SP
MOVE #DE, .IY
BR /main
Etmodificar:
ADD .IX, #1
MOVE .A, .R1
ININT[.R1]
ADD .IX, #2
MOVE #0, [.A]
ADD .IX, #1
MOVE [.A], .R0
ADD .IX, #2
CMP .R0, [.A]
BN /etiq5
ADD .IX, #3
MOVE #0, [.A]
BR /etiq6
etiq5:
ADD .IX, #3
MOVE #1, [.A]
etiq6:
ADD .IX, #3
CMP [.A], #0
BZ /etiq7
ADD .IX, #4
MOVE #0, [.A]
ADD .IX, #4
MOVE [.A], .R0
ADD .IY, #66
MOVE .R0, [.A]
etiq7:
BR [.IX]
EtesCierto:
ADD .IX, #1
MOVE #1, [.A]
SUB #TAM_RA_EtesCierto, #1
ADD .A, .IX
MOVE .A, .R9
ADD #1, .IX
MOVE [.A], [.R9]
BR [.IX]
EtnoEsCierto:
ADD .IX, #1
MOVE #0, [.A]
SUB #TAM_RA_EtnoEsCierto, #1
ADD .A, .IX
MOVE .A, .R9
ADD #1, .IX
MOVE [.A], [.R9]
```

```
BR [.IX]
main: NOP
ADD .IY, #67
MOVE #0, [.A]
ADD .IY, #67
MOVE [.A], .R0
ADD .IY, #0
MOVE .R0, [.A]
ADD .IY, #68
MOVE #1, [.A]
ADD .IY, #68
MOVE [.A], .R0
ADD .IY, #66
MOVE .R0, [.A]
etiq1:
ADD .IY, #69
MOVE #10, [.A]
ADD .IY, #0
MOVE [.A], .R0
ADD .IY, #69
CMP .R0, [.A]
BN /etiq3
ADD .IY, #70
MOVE #0, [.A]
BR /etiq4
etiq3:
ADD .IY, #70
MOVE #1, [.A]
etiq4:
ADD .IY, #71
MOVE .A, .R1
ADD .IY, #66
MOVE [.A], .R0
ADD .IY, #70
AND [.A], .R0
MOVE .A, [.R1]
ADD .IY, #71
CMP [.A], #0
BZ /etiq2
ADD .IY, #72
MOVE #1, [.A]
ADD .IY, #0
MOVE [.A], .R0
ADD .IY, #72
ADD [.A], .R0
MOVE .A, .R1
ADD .IY, #73
MOVE .R1, [.A]
ADD .IY, #73
MOVE [.A], .R0
ADD .IY, #0
MOVE .R0, [.A]
```

```
ADD .IY, #74
MOVE #cad1, [.A]
ADD .IY, #74
MOVE [.A], .R1
WRSTR [.R1]
BR /etiq1
etiq2:
HALT
TAM_RA_main: EQU 139
TAM_RA_Etmodificar: EQU 5
TAM_RA_EtesCierto: EQU 3
TAM_RA_EtnoEsCierto: EQU 3
cad1:
DATA "Iteracion producida"
DE: RES 139
PILA: NOP
END
```

Caso de prueba 4:

```
/* Programa de ejemplo 4 */

let string nombre;
function saludo (string x) {
   print(x);
}
nombre='Alex';
saludo(nombre);
let string nombre2;
nombre2=nombre;
print(nombre2);
```

- Cuartetos:

```
(:, Etsaludo, , )
(print, , , x)
(return, , , )
(:=c, "Alex", , t1)
(=, t1, , nombre)
(param, , , nombre)
(call, Etsaludo, , )
(=, nombre, , nombre2)
(print, , , nombre2)
```

- Código Final:

```
ORG 0
MOVE #PILA, .IX
MOVE .IX, .SP
```

```
MOVE #DE, .IY
BR /main
Etsaludo:
ADD .IX, #1
MOVE .A, .R1
WRSTR [.R1]
BR [.IX]
main: NOP
ADD .IY, #65
MOVE #cad1, [.A]
ADD .IY, #65
MOVE [.A], .R7
ADD .IY, #0
MOVE .A, .R6
etiq1:
CMP [.R7], #0
BZ /etiq2
MOVE [.R7], [.R6]
INC .R6
INC .R7
BR /etiq1
etiq2:
MOVE [.R7], [.R6]
ADD #0, .IX
ADD #1, .A
MOVE .A, .R0
ADD #0, .IY
MOVE .A, .R7
MOVE .R0, .R6
etiq3:
CMP [.R7], #0
BZ /etiq4
MOVE [.R7], [.R6]
INC .R6
INC .R7
BR /etiq3
etiq4:
MOVE [.R7], [.R6]
MOVE #dir_ret1, #0[.IX]
BR /Etsaludo
dir_ret1:
ADD .IY, #0
MOVE .A, .R7
ADD .IY, #130
MOVE .A, .R6
etiq5:
CMP [.R7], #0
BZ /etiq6
MOVE [.R7], [.R6]
INC .R6
INC .R7
BR /etiq5
```

```
etiq6:
MOVE [.R7], [.R6]
ADD .IY, #130
MOVE .A, .R1
WRSTR [.R1]
HALT
TAM_RA_main: EQU 195
TAM_RA_Etsaludo: EQU 66
cad1:
DATA "Alex"
DE: RES 195
PILA: NOP
END
```

Caso de prueba 5:

```
/* Programa de ejemplo 5 */

let boolean stop;
stop = false;
function Sumar int(int a, int b) {
   return a+b;
}

let int sumando;
sumando = Sumar(4, 5);
if(sumando < 10)
stop = true;

if(stop)
print('Resultado menor a 10');
```

- Cuartetos:

```
(:=, 0, , t1)
(=, t1, , stop)
(:, EtSumar, , )
(+, a, b, t2)
(return, t2, , )
(:=, 4, , t3)
(:=, 5, , t4)
(param, , , t3)
(param, , , t4)
(call, EtSumar, , t5)
(=, t5, , sumando)
(:=, 10, , t6)
(if<, sumando, t6, etiq1)
(:=, 0, , t7)
(goto, , , etiq2)
(:, etiq1, , )
(:=, 1, , t7)
(:, etiq2, , )
```

```
(if=, t7, 0, etiq3)
(:=, 1, , t8)
(=, t8, , stop)
(:, etiq3, , )
(if=, stop, 0, etiq4)
(:=c, "Resultado menor a 10", , t9)
(print, , , t9)
(:, etiq4, , )
```

- Código Final:

```
ORG 0
MOVE #PILA, .IX
MOVE .IX, .SP
MOVE #DE, .IY
BR /main
EtSumar:
ADD .IX, #1
MOVE [.A], .R0
ADD .IX, #2
ADD [.A], .R0
MOVE .A, .R1
ADD .IX, #3
MOVE .R1, [.A]
SUB #TAM_RA_EtSumar, #1
ADD .A, .IX
MOVE .A, .R9
ADD #3, .IX
MOVE [.A], [.R9]
BR [.IX]
main: NOP
ADD .IY, #1
MOVE #0, [.A]
ADD .IY, #1
MOVE [.A], .R0
ADD .IY, #0
MOVE .R0, [.A]
ADD .IY, #3
MOVE #4, [.A]
ADD .IY, #4
MOVE #5, [.A]
ADD #0, .IX
ADD #1, .A
MOVE .A, .R0
ADD #3, .IY
MOVE [.A], [.R0]
ADD #0, .IX
ADD #2, .A
MOVE .A, .R0
ADD #4, .IY
MOVE [.A], [.R0]
MOVE #dir_ret1, #0[.IX]
BR /EtSumar
```

```
dir_ret1:
SUB #TAM_RA_EtSumar, #1
ADD .A, .IX
MOVE .A, .R9
ADD #5, .IY
MOVE [.R9], [.A]
ADD .IY, #5
MOVE [.A], .R0
ADD .IY, #2
MOVE .R0, [.A]
ADD .IY, #6
MOVE #10, [.A]
ADD .IY, #2
MOVE [.A], .R0
ADD .IY, #6
CMP .R0, [.A]
BN /etiq1
ADD .IY, #7
MOVE #0, [.A]
BR /etiq2
etiq1:
ADD .IY, #7
MOVE #1, [.A]
etiq2:
ADD .IY, #7
CMP [.A], #0
BZ /etiq3
ADD .IY, #8
MOVE #1, [.A]
ADD .IY, #8
MOVE [.A], .R0
ADD .IY, #0
MOVE .R0, [.A]
etiq3:
ADD .IY, #0
CMP [.A], #0
BZ /etiq4
ADD .IY, #9
MOVE #cad1, [.A]
ADD .IY, #9
MOVE [.A], .R1
WRSTR [.R1]
etiq4:
HALT
TAM_RA_main: EQU 74
TAM_RA_EtSumar: EQU 5
cad1:
DATA "Resultado menor a 10"
DE: RES 74
PILA: NOP
END
```