3ª Guía Compiladores

Nombre: Escalona Zuñiga Juan Carlos
Grupo: 5CM2

Falso o Verdadero (F/V)

1.-En lenguaje C los **parámetros formales** no tienen nombre_____ (F)

2.-En lenguaje C los **parámetros formales** son como **variables locales** que ya fueron inicializadas en el momento de la llamada a la función o procedimiento (V)

3.-En lenguaje C las **variables locales** (no estáticas) se crean cuando se entra a una función y se destruyen cuando se sale de la función (V)

4.-En hoc los **parámetros formales** no tienen nombre_____ (F)

5.-No es posible definir **funciones recursivas** en hoc_____ (F)

6.-En hoc no hay **variables locales**_____ (F)

7.-Es imposible que la pila de hoc se desborde (Stack Overflow)____ (F)

8.-En hoc cuando una función termina su ejecución se saca su **marco** de la pila (V)

9.-En hoc los **parámetros reales** son listas de **expresiones** _____ (V)

10.-En hoc las llamadas a función no son **expresiones**_____ (F)

11.-En hoc las llamadas a procedimiento son **enunciados**_____ (V)

12.-En hoc el código que ejecuta la **maquina virtual de pila** esta en prefijo (considere como se ejecuta una operacion de suma) (F)

13.-En hoc los **parámetros reales** se meten a la pila (V)

14.-En el **Análisis sintáctico ascendente** el árbol de análisis sintáctico la construcción se inicia en la raíz y avanza hacia las hojas (F)

15.-En el **Análisis sintáctico descendente** se construye el árbol de análisis sintáctico de la cadena de desde las hojas y avanza hacia las raíz (F)

1.-Cuantas pilas hay en HOC6? B
a) 1 b) 2 c) 3 d) 4

2.-En HOC6 que se usa como pila de llamadas? B
a) prog b) stack c) frame d) fp

3,-Cual es el apuntador que apunta a la pila de llamadas? D
a) stack b) stackp c) frame d) fp

4.-Cual es el apuntador que apunta a la 1a instruccion de una funcion builtin? B
a) ptr b) defn c) progbase d) progp

5.-Cual es el apuntador que apunta a la 1a instruccion del cuerpo de una funcion o precedimiento definido por el usuario? A
a) ptr b) defn c) progbase d) progp

6.-Cual es el tipo de retorno de una funcion definida por el usuario en HOC6? A
a) double b) int c) Datum d) Symbol

7.-Que instruccion saca un marco de funcion de la pila de llamadas? A
a) --stackp b) --fp c) ++stackp d) ++fp
8.-Cual es el apuntador que apunta al ultimo de los parametros? D
a) stackp b) progbase c) fp d) argn

Problema.-Que codigo genera hoc5 para el codigo de abajo (dibuje el mapa de memoria)

```
i=0
while(1) print I

push 0
store i

L1:
   push 1
   jz L2
   load i
   print
   jmp L1
L2:
   halt
```

tabla de analisis sintactico LR

Edo	acción						ir_a		
	id	+	*	(	)	$	E	T	F
0	d5			d4			1	2	3
1		d6				acep			
2		r2	d7		r2	r2			
3		r4	r4		r4	r4			
4	d5			d4			8	2	3
5		r6	r6		r6	r6			
6	d5			d4				9	3
7	d5			d4					10
8		d6			d11				
9		r1	d7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

Gramatica

(1) $E \rightarrow E + T$

(2) $E \rightarrow T$

(3) $T \rightarrow T * F$

(4) $T \rightarrow F$

(5) $F \rightarrow (E)$

(6) $F \rightarrow id$

Problema.--Analice la siguientes cadenas:
id * id
id * (id + id)
(id + id) * (id + id)

id * (id + id * (id + id))

1. Cadena: id * id

Pila	Entrada	Acción
0	id * id $	d5 (desplazar a estado 5)
0 id 5	* id $	r6 (reducir por E → id)
0 E 3	* id $	r4 (reducir por J → E)
0 J 2	* id $	d7 (desplazar a estado 7)
0 J 2 * 7	id $	d5 (desplazar a estado 5)
0 J 2 * 7 id 5	$	r6 (reducir por E → id)
0 J 2 * 7 E 10	$	r3 (reducir por J → J * E)
0 J 2	$	r2 (reducir por E → J)
0 E 1	$	**acep** (cadena aceptada)

---

2. Cadena: id * (id + id)

Pila	Entrada	Acción
0	id * (id + id) $	d5 (desplazar a estado 5)
0 id 5	* (id + id) $	r6 (reducir por E → id)
0 E 3	* (id + id) $	r4 (reducir por J → E)
0 J 2	* (id + id) $	d7 (desplazar a estado 7)
0 J 2 * 7	(id + id) $	d4 (desplazar a estado 4)
0 J 2 * 7 (4	id + id) $	d5 (desplazar a estado 5)
0 J 2 * 7 (4 id 5	+ id) $	r6 (reducir por E → id)
0 J 2 * 7 (4 E 8	+ id) $	d6 (desplazar a estado 6)
0 J 2 * 7 (4 E 8 + 6	id) $	d5 (desplazar a estado 5)
0 J 2 * 7 (4 E 8 + 6 id 5	) $	r6 (reducir por E → id)
0 J 2 * 7 (4 E 8 + 6 E 9	) $	r1 (reducir por E → E + E)
0 J 2 * 7 (4 E 8	) $	d11 (desplazar a estado 11)
0 J 2 * 7 (4 E 8) 11	$	r5 (reducir por E → (E))
0 J 2 * 7 E 10	$	r3 (reducir por J → J * E)
0 J 2	$	r2 (reducir por E → J)
0 E 1	$	**acep** (cadena aceptada)

---

3. Cadena: (id + id) * (id + id)

Pila	Entrada	Acción
0	(id + id) * (id + id) $	d4 (desplazar a estado 4)
0 (4	id + id) * (id + id) $	d5 (desplazar a estado 5)
0 (4 id 5	+ id) * (id + id) $	r6 (reducir por E → id)
0 (4 E 8	+ id) * (id + id) $	d6 (desplazar a estado 6)
0 (4 E 8 + 6	id) * (id + id) $	d5 (desplazar a estado 5)

Pila	Entrada	Acción
0 (4 E 8 + 6 id 5	) * (id + id) $	r6 (reducir por E → id)
0 (4 E 8 + 6 E 9	) * (id + id) $	r1 (reducir por E → E + E)
0 (4 E 8	) * (id + id) $	d11 (desplazar a estado 11)
0 (4 E 8) 11	* (id + id) $	r5 (reducir por E → (E))
0 E 1	* (id + id) $	d6 (desplazar a estado 6)
0 E 1 * 6	(id + id) $	d4 (desplazar a estado 4)
0 E 1 * 6 (4	id + id) $	d5 (desplazar a estado 5)
0 E 1 * 6 (4 id 5	+ id) $	r6 (reducir por E → id)
0 E 1 * 6 (4 E 8	+ id) $	d6 (desplazar a estado 6)
0 E 1 * 6 (4 E 8 + 6	id) $	d5 (desplazar a estado 5)
0 E 1 * 6 (4 E 8 + 6 id 5	) $	r6 (reducir por E → id)
0 E 1 * 6 (4 E 8 + 6 E 9	) $	r1 (reducir por E → E + E)
0 E 1 * 6 (4 E 8	) $	d11 (desplazar a estado 11)
0 E 1 * 6 (4 E 8) 11	$	r5 (reducir por E → (E))
0 E 1 * 6 E 9	$	r1 (reducir por E → E * E)
0 E 1	$	**acep** (cadena aceptada)

---

4. Cadena: id * (id + id * (id + id))

Pila	Entrada	Acción
0	id * (id + id * (id + id)) $	d5 (desplazar a estado 5)
0 id 5	* (id + id * (id + id)) $	r6 (reducir por E → id)
0 E 3	* (id + id * (id + id)) $	r4 (reducir por J → E)
0 J 2	* (id + id * (id + id)) $	d7 (desplazar a estado 7)
0 J 2 * 7	(id + id * (id + id)) $	d4 (desplazar a estado 4)
0 J 2 * 7 (4	id + id * (id + id)) $	d5 (desplazar a estado 5)
0 J 2 * 7 (4 id 5	+ id * (id + id)) $	r6 (reducir por E → id)
0 J 2 * 7 (4 E 8	+ id * (id + id)) $	d6 (desplazar a estado 6)
0 J 2 * 7 (4 E 8 + 6	id * (id + id)) $	d5 (desplazar a estado 5)
0 J 2 * 7 (4 E 8 + 6 id 5	* (id + id)) $	r6 (reducir por E → id)
0 J 2 * 7 (4 E 8 + 6 E 9	* (id + id)) $	r1 (reducir por E → E + E)
0 J 2 * 7 (4 E 8	* (id + id)) $	d7 (desplazar a estado 7)
0 J 2 * 7 (4 E 8 * 7	(id + id)) $	d4 (desplazar a estado 4)
0 J 2 * 7 (4 E 8 * 7 (4	id + id)) $	d5 (desplazar a estado 5)
0 J 2 * 7 (4 E 8 * 7 (4 id 5	+ id)) $	r6 (reducir por E → id)
0 J 2 * 7 (4 E 8 * 7 (4 E 8	+ id)) $	d6 (desplazar a estado 6)
0 J 2 * 7 (4 E 8 * 7 (4 E 8 + 6	id)) $	d5 (desplazar a estado 5)
0 J 2 * 7 (4 E 8 * 7 (4 E 8 + 6 id 5	)) $	r6 (reducir por E → id)
0 J 2 * 7 (4 E 8 * 7 (4 E 8 + 6 E 9	)) $	r1 (reducir por E → E + E)
0 J 2 * 7 (4 E 8 * 7 (4 E 8	)) $	d11 (desplazar a estado 11)

Pila	Entrada	Acción
0 J 2 * 7 (4 E 8 * 7 (4 E 8) 11	) $	r5 (reducir por E → (E))
0 J 2 * 7 (4 E 8 * 7 E 10	) $	r3 (reducir por J → J * E)
0 J 2 * 7 (4 J 2	) $	r2 (reducir por E → J)
0 J 2 * 7 (4 E 8	) $	d11 (desplazar a estado 11)
0 J 2 * 7 (4 E 8) 11	$	r5 (reducir por E → (E))
0 J 2 * 7 E 10	$	r3 (reducir por J → J * E)
0 J 2	$	r2 (reducir por E → J)
0 E 1	$	**acep** (cadena aceptada)

Muestre el contenido de la pila, la entrada y la acción a realizar

Problema.-Considere la siguiente gramática:

S → A	A → **ε**	A → **bb**A

-Analice la siguiente cadena: **bbbb**
Muestre el contenido de la pila, la entrada y la acción a realizar

tabla de analisis sintactico

	b	$
S	S → A	S → A
A	A → bbA	A → ε

Tabla de Análisis Sintáctico (SLR):

Estado	Acción (b)	Acción ($)	Ir a (S)	Ir a (A)
0	desplazar	-	-	1
1	-	aceptar	-	-
2	desplazar	-	-	3
3	reducir (A → bbA)	reducir (A → bbA)	-	-

Proceso de Análisis para bbbb

Pila	Entrada	Acción	Detalle
0	b b b b $	desplazar (b)	Se lee el primer `b`.
0 b 2	b b b $	desplazar (b)	Se lee el segundo `b`.
0 b 2 b 2	b b $	desplazar (b)	Se lee el tercer `b`.
0 b 2 b 2 b 2	b $	desplazar (b)	Se lee el cuarto `b`.
0 b 2 b 2 b 2 b 2	$	reducir (A → ε)	No hay más `b`, se reduce a `A → ε`.
0 b 2 b 2 b 2 A 3	$	reducir (A → bbA)	Se reduce `bbA` a `A`.
0 b 2 A 3	$	reducir (A → bbA)	Nueva reducción `bbA` a `A`.
0 A 1	$	aceptar	**Cadena aceptada**.

Problema.-Considere la siguiente gramática:

1) A → **x**A	2) A → **y**A	3) A → **y**

-Analice la siguiente cadena: xyy
Muestre el contenido de la pila, la entrada y la acción a realizar

tabla de analisis sintactico

	x	y	$
A	A → xA	A → yA A → y	

Tabla de Análisis Sintáctico (SLR):

Estado	Acción (x)	Acción (y)	Acción ($)	Ir a (A)
0	desplazar	desplazar	-	1
1	desplazar	desplazar	aceptar	-

Proceso de Análisis para xyy

Pila	Entrada	Acción	Detalle
0	x y y $	desplazar (x)	Se lee el primer `x`.
0 x 2	y y $	desplazar (y)	Se lee el primer `y`.
0 x 2 y 3	y $	reducir (A → y)	Se reduce `y` a `A`.
0 x 2 A 4	y $	desplazar (y)	Se lee el segundo `y`.
0 x 2 A 4 y 3	$	reducir (A → y)	Se reduce `y` a `A`.
0 x 2 A 4 A 5	$	reducir (A → yA)	Se reduce `yA` a `A`.
0 A 1	$	aceptar	**Cadena aceptada**.

Problema.-Considere la siguiente gramática:

1) A → A **m**	2) A → **n**

-Analice la siguiente cadena: nmmm
Muestre el contenido de la pila, la entrada y la acción a realizar

tabla de analisis sintactico

	accion			Ir_a
	m	n	$	A
0		d1		2
1	r2		r2	
2	d3		ace	
3	r1		r1	

Tabla de Análisis Sintáctico (SLR):

Estado	Acción (m)	Acción (n)	Acción ($)	Ir a (A)
0	-	d1	-	2
1	r2	-	r2	-
2	d3	-	aceptar	-
3	r1	-	r1	-

Proceso de Análisis para nmmm

Pila	Entrada	Acción	Detalle
0	n m m m $	desplazar (n)	Se lee el primer símbolo `n`.
0 n 1	m m m $	reducir (A → n)	Se reduce `n` a `A`.
0 A 2	m m m $	desplazar (m)	Se lee el primer `m`.
0 A 2 m 3	m m $	reducir (A → A m)	Se reduce `A m` a `A`.
0 A 2	m m $	desplazar (m)	Se lee el segundo `m`.
0 A 2 m 3	m $	reducir (A → A m)	Se reduce `A m` a `A`.
0 A 2	m $	desplazar (m)	Se lee el tercer `m`.
0 A 2 m 3	$	reducir (A → A m)	Se reduce `A m` a `A`.
0 A 2	$	aceptar	**Cadena aceptada**.

Problema.-Considere la siguiente gramática:

1) C → AB	2) A → **a**	3) B → **a**

-Analice la siguiente cadena: **aa**
Muestre el contenido de la pila, la entrada y la acción a realizar

tabla de analisis sintactico

	accion		Ir_a		
	a	$	A	B	C
0	d1		3		2
1	r2				
2		ac			
3	d4			5	
4		r3			
5		r1			

Tabla de Análisis Sintáctico (SLR):

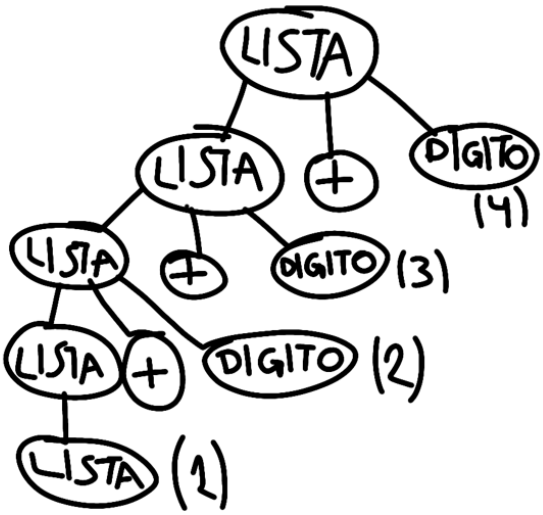
Estado	Acción (a)	Acción ($)	Ir a (A)	Ir a (B)	Ir a (C)
0	d1	-	3	-	2
1	r2	r2	-	-	-
2	-	aceptar	-	-	-
3	d4	-	-	5	-
4	-	r3	-	-	-
5	-	r1	-	-	-

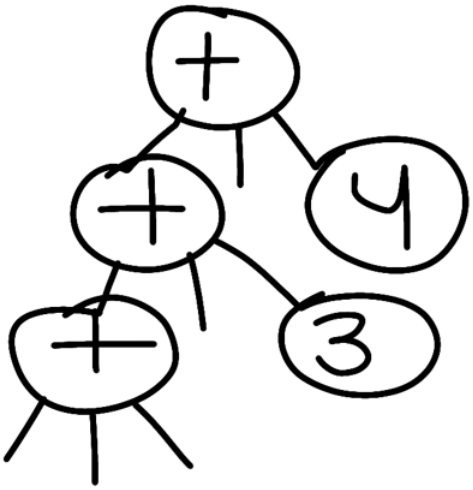
Proceso de Análisis para aa

Pila	Entrada	Acción	Detalle
0	a a $	desplazar (a)	Se lee el primer `a`.
0 a 1	a $	reducir (A → a)	Se reduce `a` a `A`.
0 A 3	a $	desplazar (a)	Se lee el segundo `a`.
0 A 3 a 4	$	reducir (B → a)	Se reduce `a` a `B`.
0 A 3 B 5	$	reducir (C → A B)	Se reduce `A B` a `C`.
0 C 2	$	aceptar	**Cadena aceptada**.

PROBLEMAS
Dada la gramatica
LISTA -> LISTA + DIGITO | LISTA – DIGITO | DIGITO
DIGITO -> 0 | 1 |2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

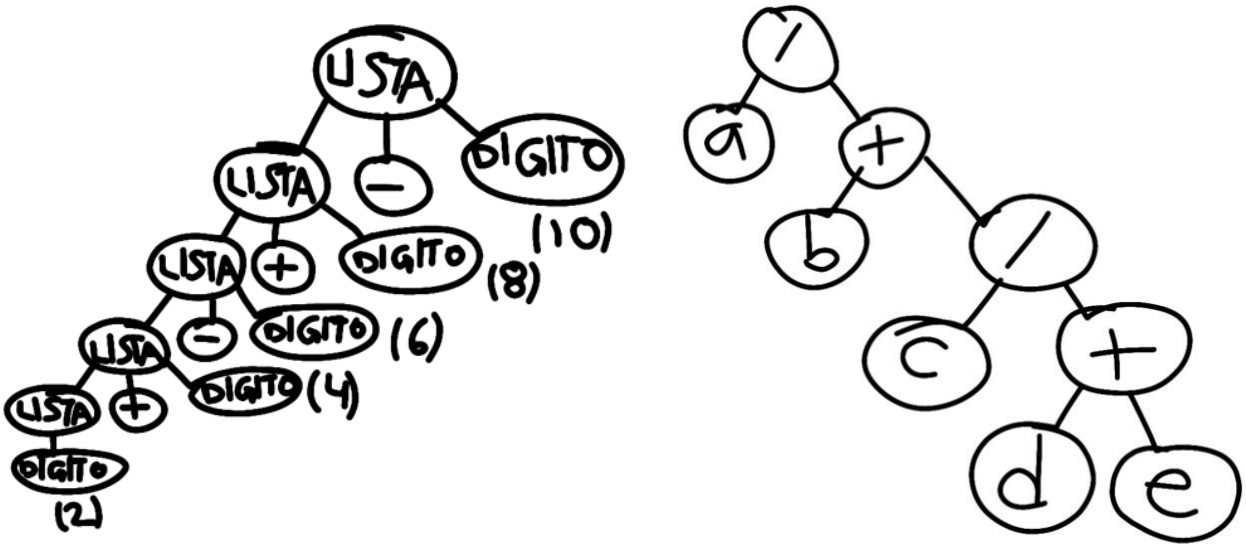
1.-Dibuje el **arbol de analisis sintactico** , el **arbol sintactico** y el **grafo de dependencias** para las siguientes cadenas:
1+2+3+4 y 2+4-6+8-10





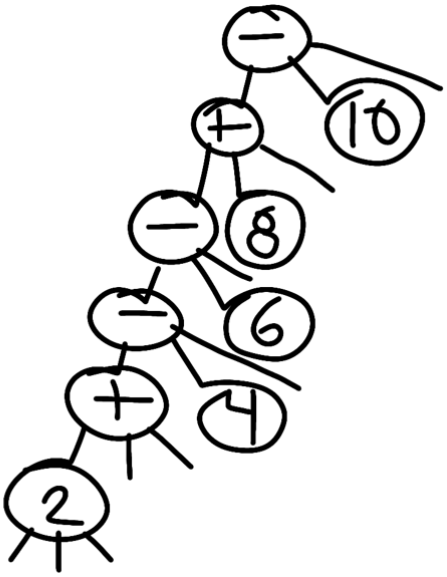
Grafo de Dependencias

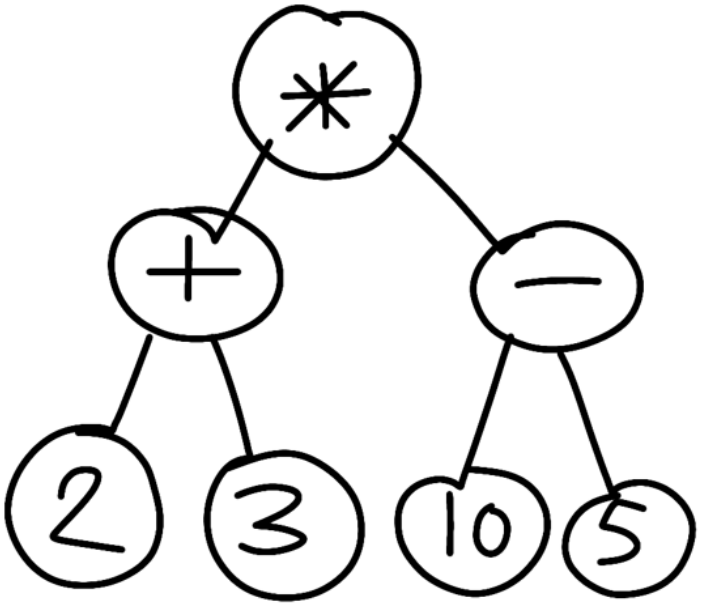
graph TD
 1 --> +
 2 --> +
 + --> +2
 3 --> +2
 +2 --> +3
 4 -->



Grafo de dependencias
 2 --> +
 4 --> +
 + --> -
 6 --> -
 - --> +
 8 --> +
 + --> -
 10 --> -

2.-Dibuje el **arbol sintactico** para las siguientes cadenas:
(2+3) * (10 – 5) y a/(b+c/(d+e))





3.-Esciba las expresiones de tipo para :
a) un entero → int
b) un apuntador a entero → int *
c) un arreglo de apuntadores a char → char *[]
d) struct agregado { char x; int y; double z};

4.-Supongase que los nombres de tipos enlace y nodo se definen como en la seccion 6.3 del libro del dragon. Cuales de las siguientes expresiones de tipos son estructuralmente equivalentes?
enlace
pointer nodo
pointer enlace
pointer(record ((infoXinteger)X(siguienteXpointer(nodo)))

enlace y pointer(record...).
pointer enlace y pointer(record...) (si enlace es el record).

PROBLEMAS
Genere el codigo de 3 direcciones de
1.- a * - (b+c) y a:= (b + c) * (e + f) ;

t1 = b + c
t2 = -t1
t3 = a * t2

t1 = b + c
t2 = e + f
t3 = t1 * t2
a = t3

2.-(a * b + h) - j * k + 1

t1 = a * b
t2 = t1 + h
t3 = j * k
t4 = t2 - t3
t5 = t4 + 1

3.- **a > b + h or b == d**

t1 = b + h
t2 = a > t1
t3 = b == d
t4 = t2 or t3

Problema.- Traduzca a flujo de control (obtenga el codigo de tres direcciones) las siguientes expresiones booleanas

I) a < b **or** c < d
if a < b goto L_true
if c < d goto L_true
if e < f goto L_true
goto L_false
L_true:
  // ...
L_false:
  // ...

II) a < b **or** c < d **or** e < f
if a < b goto L_true
if c < d goto L_true
if e < f goto L_true
if g < h goto L_true
goto L_false
L_true:
  // ...
L_false:
  // ...

III) a < b **or** c < d **or** e < f **or** g < h
if a < b goto L_true
if c < d goto L_true
if e < f goto L_true
if g < h goto L_true
goto L_false
L_true:
  // ...
L_false:

```
   // ...
```

IV) a < b **and** c < d

```
if a >= b goto L_false
if c >= d goto L_false
goto L_true
L_true:
   // ...
L_false:
   // ...
```

V) a < b **and** c < d **and** e < f

```
f a >= b goto L_false
if c >= d goto L_false
if e >= f goto L_false
goto L_true
L_true:
   // ...
L_false:
   // ...
```

VI) a < b **and** c < d **and** e < f **and** g < h

```
if a >= b goto L_false
if c >= d goto L_false
if e >= f goto L_false
if g >= h goto L_false
goto L_true
L_true:
   // ...
L_false:
   // ...
```

VIII) a < b **or** c < d **and** e < f

```
if a < b goto L_true
if c >= d goto L_false
if e >= f goto L_false
goto L_true
L_true:
   // ...
L_false:
   // ...
```

IX) a < b **and** c < d **or** e < f

```
if a < b goto L_check_cd
goto L_check_ef
L_check_cd:
   if c < d goto L_true
   goto L_check_ef
L_check_ef:
   if e < f goto L_true
   goto L_false
```

L_true:
  // …
L_false:
  // …

.-Genere el codigo de 3 direcciones de :

a := 0
while (a <= 5) {
  a :=a + 1
}
	for (i = 0; i < 5; i=i+1){
	  A;
	}

a = 0

L1:

  t1 = a <= 5
  if t1 == 0 goto L2 // Si a > 5, salir del bucle
  // Cuerpo del while (vacío o instrucciones específicas)
  goto L1 // Repetir bucle
L2:

i = 0

L3:
  t2 = i < 5
  if t2 == 0 goto L4 // Si i >= 5, salir del bucle
  // Cuerpo del for (vacío o instrucciones específicas)
  i = i + 1 // Incremento
  goto L3 // Repetir bucle
L4:

6.-Genere el codigo de 3 direcciones de :

a := 2 * x + 10;
while (a <= p + 2) {
  a := p[4+i*2] ;
}
a := a + 1;
	for (i = 0; i < 5; i=i+1){
	  a=p[2*i];
	}

t1 = 2 * x

i = 0

```
a = t1 + 10

// while (a <= p + 2)
L1:
   t2 = p + 2
   t3 = a <= t2
   if t3 == 0 goto L2

   // a := p[4 + i * 2]
   t4 = i * 2
   t5 = 4 + t4
   t6 = p + t5
   a = *t6

   goto L1
```

```
L3:
   t7 = i < 5
   if t7 == 0 goto L4

   // a = p[2 * i]
   t8 = 2 * i
   t9 = p + t8
   a = *t9

   i = i + 1
   goto L3
L4:
```

Generacion Codigo Objeto

7.-Si una instrucción de asignación de la forma x=y+z . Se traduce a:	Como se traducen

mov y, R0
add z, R0
mov R0 , x
	a = b + c;
	d = a + c;
	a = a + 1;

```
mov b, R0
add c, R0
mov R0, a

mov a, R0
add c, R0
mov R0, d

mov a, R0
add 1, R0
mov R0, a
```

8.-Si una instrucción de asignación de la forma a=a+1 . Se traduce a:	Como se traducen

Mov a , R0
add #1, R0
mov R0, a
	a = c + 2
	d = a + 3

```
mov c, R0
add #2, R0
```

```
mov R0, a

mov a, R0
add #3, R0
mov R0, d
```

9.-Para el siguiente codigo genere el codigo de 3 direcciones y divida el codigo generado en bloques basicos

```
w = 0;
x = x + y;
y=0;
if( x > z ){
   y = x;
   x++;
} else {
   y = z;
   z++;
}
w = x + z;
```

BLOQUE 1

```
w = 0
t1 = x + y
x = t1
y = 0
```

BLOQUE 2

```
t2 = x > z
if t2 == 0 goto 10
```

BLOQUE 3

```
y = x
x = x + 1
goto 12
```

BLOQUE 4

```
y = z
z = z + 1
```

BLOQUE 5

```
t3 = x + z
w = t3
```