

# Introducción

Wednesday, October 16, 2019 2:11 PM

## ► Traducción dirigida por Sintaxis

- Análisis Léxico → Expresiones Regulares
- Análisis Sintáctico → Gramáticas

NOTA

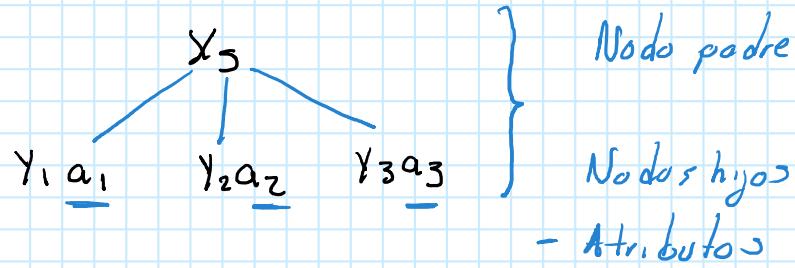
- Atributo
  - | Es cualquier propiedad de construcción
  - | Se guardan
    - | ① Tipo de variable
    - | ② Valor de la expresión
    - | ③ Dirección de una variable
    - | ④ Código Intermedio
    - | ⑤ Número de parámetros funciones
    - | ⑥ Dirección Temporal

- Definición Dirigida por Sintaxis
  - | Es una gramática libre de contexto
  - | A cada una se le asocia una o más ecuaciones de atributos
  - | o acciones semánticas
- Gramática con atributos
  - |  $a_1, a_2, \dots, a_n$  ;  $a_i \stackrel{\Delta}{=} \text{Atributos}$
  - |  $x \rightarrow y_1, y_2, \dots, y_n$  ;  $\leftarrow \text{Regla gramatical}$
  - |  $x \in N$     $y_i \in (N \cup \Sigma) \# i$  ;  $\begin{cases} \text{Terminal} \\ \text{como no terminales} \end{cases}$

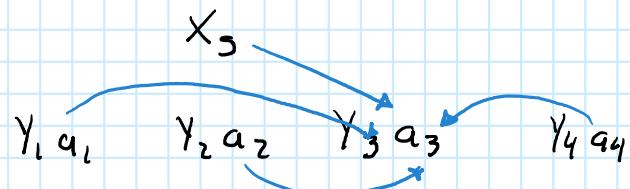
- Tipos de atributos
  - | ① Sintetizados
    - | Están asociados al encabezado producción ( $x$ )

// Grafo de dependencia para atributos sintetizados (Árbol sintáctico Anotado)

$$X_3 = f(Y_1 a_1, Y_2 a_2, \dots, Y_n a_n)$$

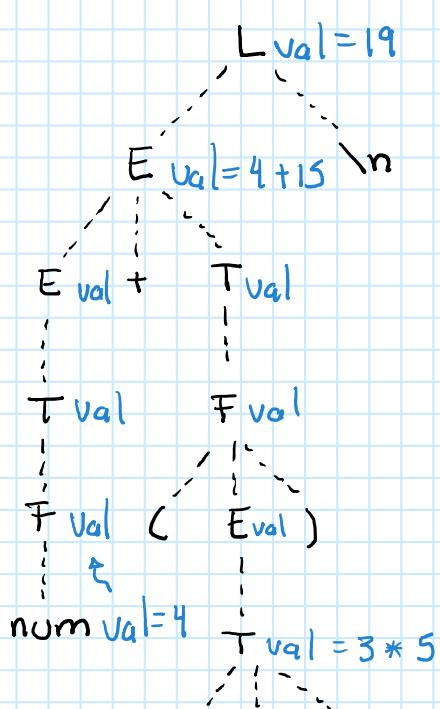


(2) Heredados | • Son aquellos asociados a cualquier  $Y_i$



~~Definición~~ Es una definición dirigida por sintaxis donde los atributos son sintetizados

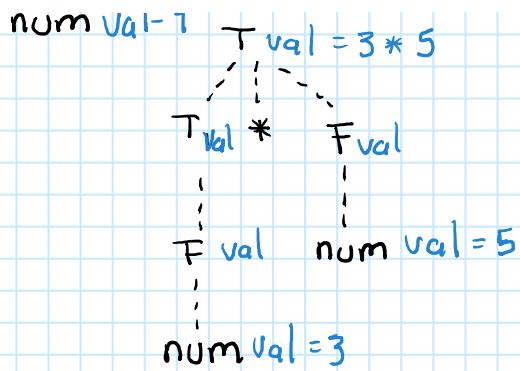
// Árbol Sintáctico | 4 + (3 \* 5)



El valor lo obtendremos del analizador Léxico (Lexer)

↑ Dependencia de atributos  
(Atributos sintetizados)

// Terminales y atributos



// Terminales y atributos

↳ No siempre tienen "valor"  
ejemplo "( )"

## Definición atribuida por Sintaxis L atribuida

DDs

- Puede tener atribuidos

- Sintetizados

- Heredados

① Los atributos heredados asociados con el encabezado (Del padre)

② Los atributos heredados asociados con los símbolos anteriores

"L" ≡ Left (De los hermanos izquierdos)

$y_1 \leftarrow y_2 \leftarrow y_3 \leftarrow \dots \leftarrow y_{i-1}$

③ No deben existir ciclos

Ejemplo :  $y_i \cdot a = y_i \cdot b$  ; No se  
 $y_i \cdot b = y_i \cdot a$  ; puede X

NOTA

$L \rightarrow E \setminus n$   $\leftarrow L\text{val} = E\text{val}$  // Sintetizado

$E \rightarrow T(E)$   $\leftarrow E\text{val} = T\text{val}$  // Heredado

- Heredado no está en el encabezado

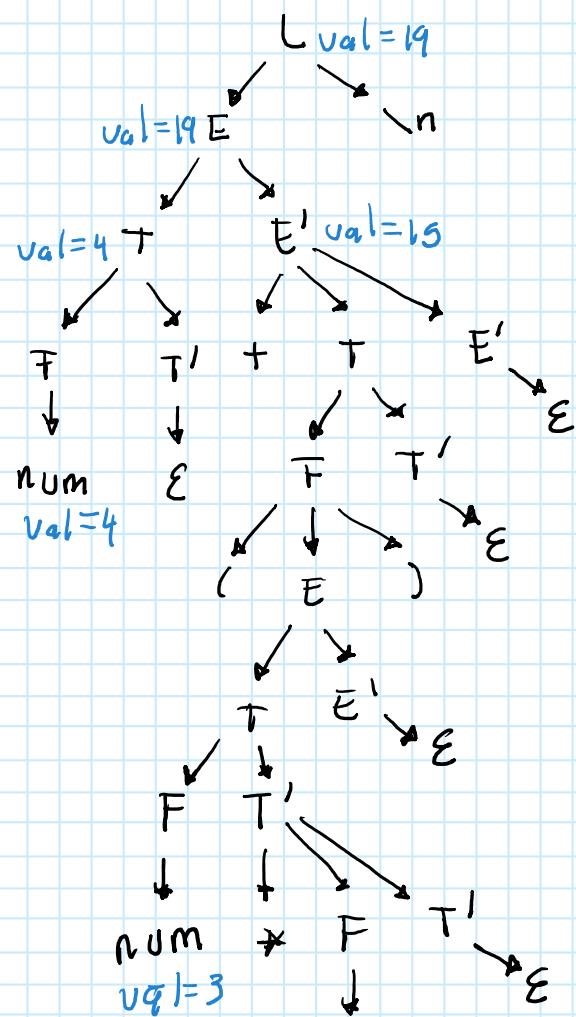
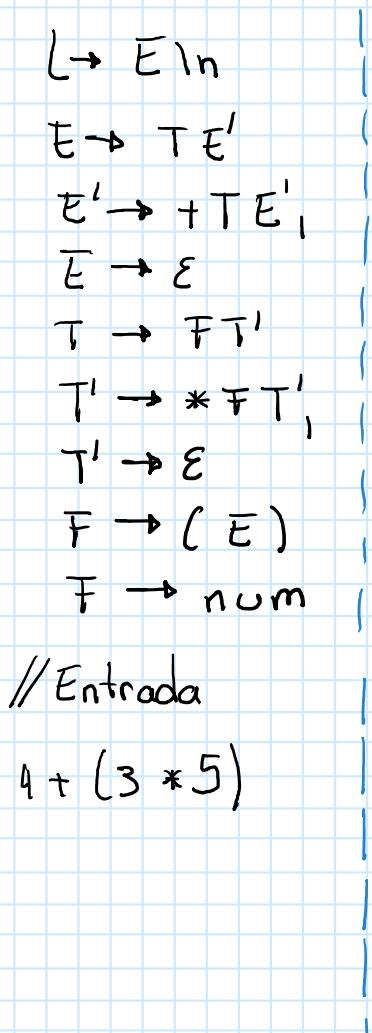
# Definición Dirigida por Sintaxis

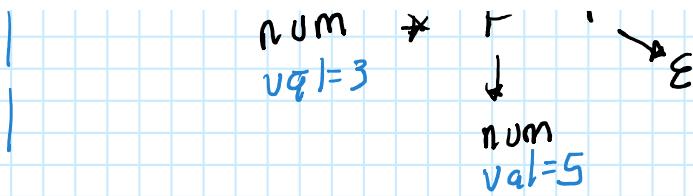
Ejemplo (Definición Dirigida por sintaxis L-atribuida)

PRODUCCIÓN	REGLAS SEMÁNTICAS
$L \rightarrow E \setminus n$	$L.val = E.val$
$E \rightarrow T E'$	$E.her = T.val$
$E' \rightarrow + T E'_1$	$E.val = E.sin$ $E'_1.her = E.her + T.val$
$E' \rightarrow \epsilon$	$E.sin = E.her$
$T \rightarrow F T'$	$T.her = F.val$
$T' \rightarrow * F T'_1$	$T.val = T.sin$ $T'_1.her = T.her \times  F.val$
$T' \rightarrow \epsilon$	$T.sin = T_1.sin$
$F \rightarrow ( E )$	$F.val = E.val$
$F \rightarrow num$	$F.val = num.level$

Adrian Ulises Mercado Martínez

Traducción Dirigida por Sintaxis

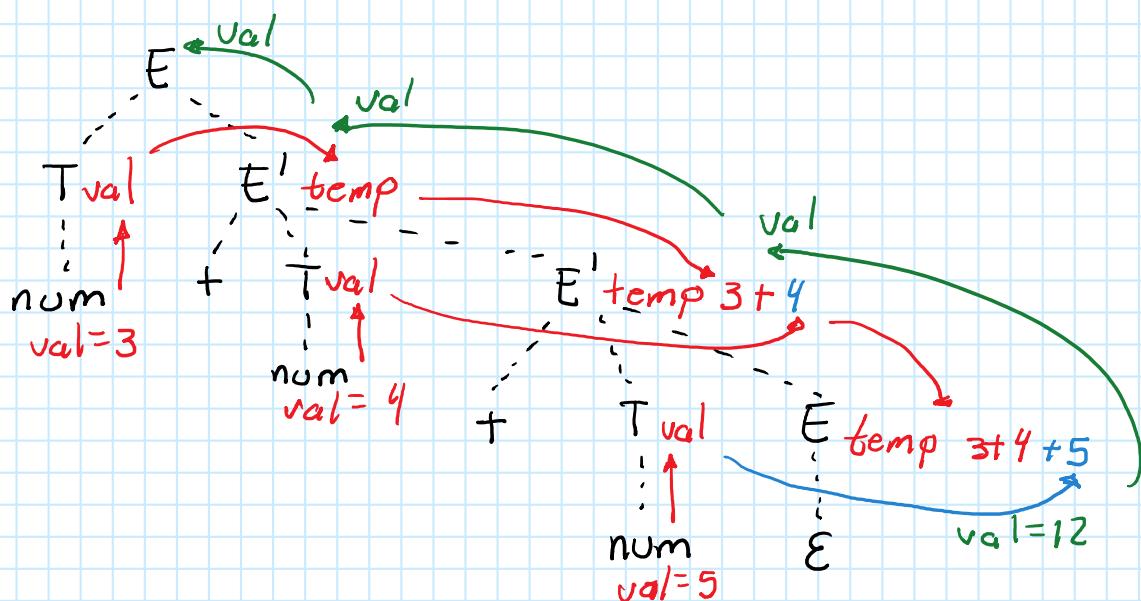




## // Gramática Reducida

$E \rightarrow T\bar{E}'$	$\mid E' . \text{temp} = T . \text{val}$
	$\mid E' . val = \bar{E}' val$
$\bar{E}' \rightarrow + T\bar{E}_1'$	$\mid \bar{E}_1' . \text{temp} = \bar{E}' . \text{temp} + T . \text{val}$
	$\mid \bar{E}_1' . val = \bar{E}_1' . val$
$E' \rightarrow E$	$\mid E' . val = E' . \text{temp}$
$T \rightarrow (E)$	$\mid T . val = E' . val$
$T \rightarrow \text{num}$	$\mid T . val = \text{num} . val$

## // Cadena de entrada 3 + 4 \* 5



// Recorrido Postfijo para obtener el resultado directamente

# Esquema de Traducción

Monday, October 21, 2019 2:39 PM

- Dirigido por sintaxis que da el orden de ejecución de las reglas semánticas

## ► Definición → Esquema

- ① Si el atributo es sintetizado, la acción semántica que evoluciona se calcula al final de la producción.
- ② Si el atributo es heredado, la acción que lo calcula se debe calcular antes del símbolo al que pertenece

$$E \rightarrow T \{ E'.temp = T.val \} E' \{ E.val = E'.val \}$$

$$E' \rightarrow + T \{ E'_1.temp = E.temp + T.val \} E'_1 \{ E'_1.val = E'.val \}$$

$$E' \rightarrow | \{ E'_1.val = E'.temp \}$$

$$T \rightarrow (E) \{ T.val = E.val \}$$

| num { T.val = num.val } // Es un intérprete

## ► Código de 3 direcciones

- Cada instrucción debe tener máximo 3 direcciones
- Una dirección ; ① Variable  
puede ser ; ② Temporal  
; ③ Constante

## ► Ejercicio

$D \rightarrow TL;$	$L.tipo = T.tipo$
$T \rightarrow int$	$T.tipo = int$
$T \rightarrow float$	$T.tipo = float$
$L \rightarrow L_1, id$	$L_1.tipo = L.tipo$
$L \rightarrow id$	$agregarSímbolo(id.val, L.tipo)$ $Id.tipo = L.tipo$
	$agregarSímbolo(id.val, L.tipo)$ <span style="float: right;">Agregan a Tabla Símbolos</span>

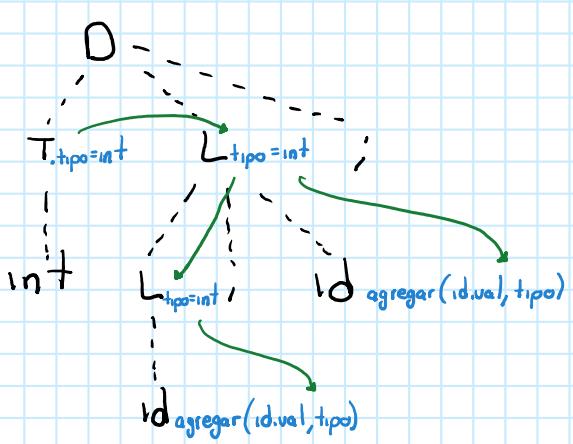
// Dirigido por sintaxis o L.Atribuida

int x, y;

► Ejercicio

$D \rightarrow T \{ L.tipo = T.tipo \} L;$   
 $T \rightarrow int \{ T.tipo = int \} | float \{ T.tipo = int \}$   
 $L \rightarrow \{ L_1.tipo = L.tipo \} L_1, id \{ agregarSímbolo(id.val, L.tipo) \}$   
 $| id \{ agregarSímbolo(id.val, L.tipo) \}$

// Derivando Cadena



// En Bison no se pueden los atributos heredados  
 $\therefore$  Crear una variable global

// Modificación

$$t_{\text{ipo}} = T \cdot t_{\text{ipo}}$$

$$T \cdot t_{\text{ipo}} = \text{int}$$

$$T \cdot t_{\text{ipo}} = \text{float}$$

agregarSímbolo(id.val, tipo)

agregarSímbolo(id.val, tipo)

$$D \rightarrow T \{ t_{\text{ipo}} = T \cdot t_{\text{ipo}} \} L;$$

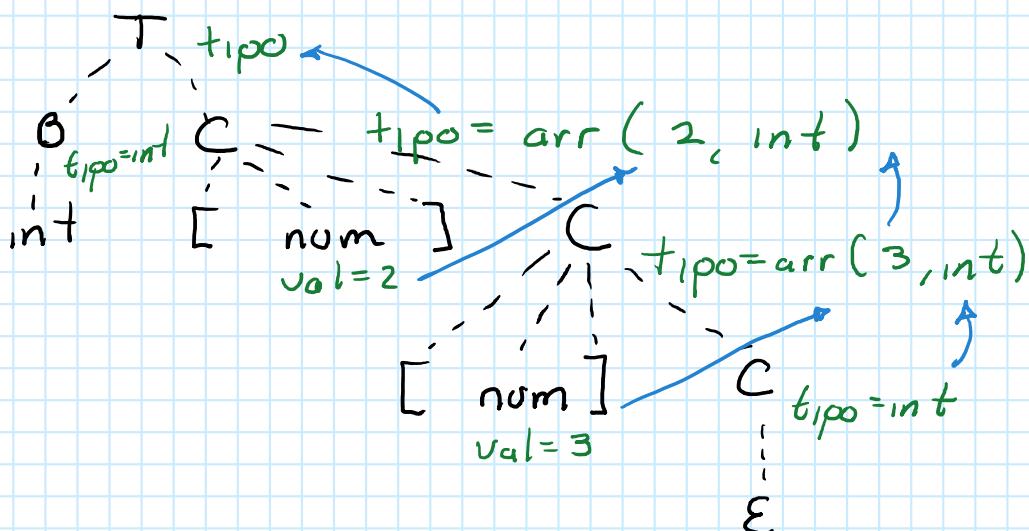
$$T \rightarrow \text{int} \{ T \cdot t_{\text{ipo}} = \text{int} \} | \text{float}$$

$$L \rightarrow L_1, \text{id} \{ \text{AregarSímbolo(id.val, tipo)} \}$$

| id AgregarSímbolo(id.val, tipo)

## Árbol del caso con variables globales

int [2][3] | base = int



# Tabla de tipos

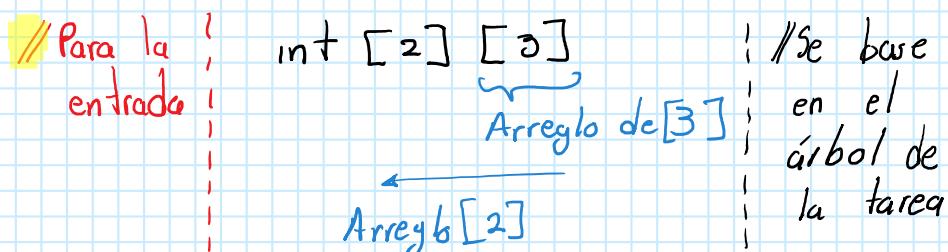
Monday, October 28, 2019 1:27 PM

- Inicialmente guarda los tipos nativos por el lenguaje pero posteriormente los creados por el usuario.

## TABLA DE TIPOS

ÍD	TIPO	TAMAÑO	#ELEMENTOS	TipoBase
0	int	4 Bytes	—	—
1	float	4 Bytes	—	—
2	arreglo	12 Bytes	3	0
3	array	24 Bytes	2	2

// long double ocupa 10



**NOTA:** Recuerda que los compuestos no basaremos en el recorrido del árbol de abajo hacia arriba.

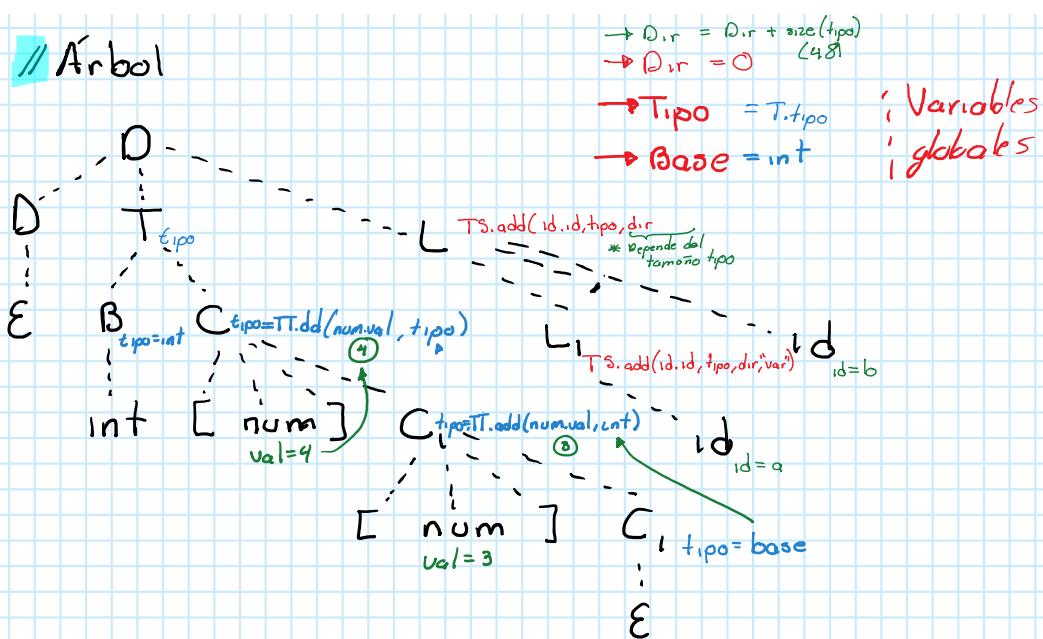
## Gramática ②

D → DTL		C → [num] C,
D → E		L → L, id
T → BC		L → .id
T → struct { D }		// Entrada
B → int		int [4][3] a, b;
B → float		

## // Árbol

$$\begin{aligned} \rightarrow D_{ir} &= D_{ir} + \text{size(tipo)} \\ \rightarrow D_{ir} &= 0 \quad (48) \\ \rightarrow T_{in} &= T_{in} \dots \text{! Variables} \end{aligned}$$

## // Árbol



### NOTAS

• TT  $\triangleq$  Tabla de tipos  $\rightarrow$  Verificar el (tamaño > 0)

• TS  $\triangleq$  Tabla de símbolo  $\rightarrow$  Verificar que el identificador no esté en la TS (Func ADD)

### \* NOTA 2

Ej 48 porque int = 4  $\rightarrow$   $4 * [3] = 12$

$$12 * [4] \rightarrow \underline{48}$$

## ► Errores Semánticos

$L \rightarrow L_1, id \quad \text{if } TS.add(id.id, tipo, dir, "var") \neq 1$

$$\quad \quad \quad dir = dir + TT.getSize(tipo)$$

else

error (...);

$L \rightarrow id \quad \text{if } TS.add(id.id, tipo, dir, "var") \neq 1$

$$\quad \quad \quad dir = dir + TT.getSize(tipo)$$

else

error (...);

if num.tipo = int and num.val > 0

$$\quad \quad \quad C.tipo = TT.add(num.val, C1.tipo)$$

else

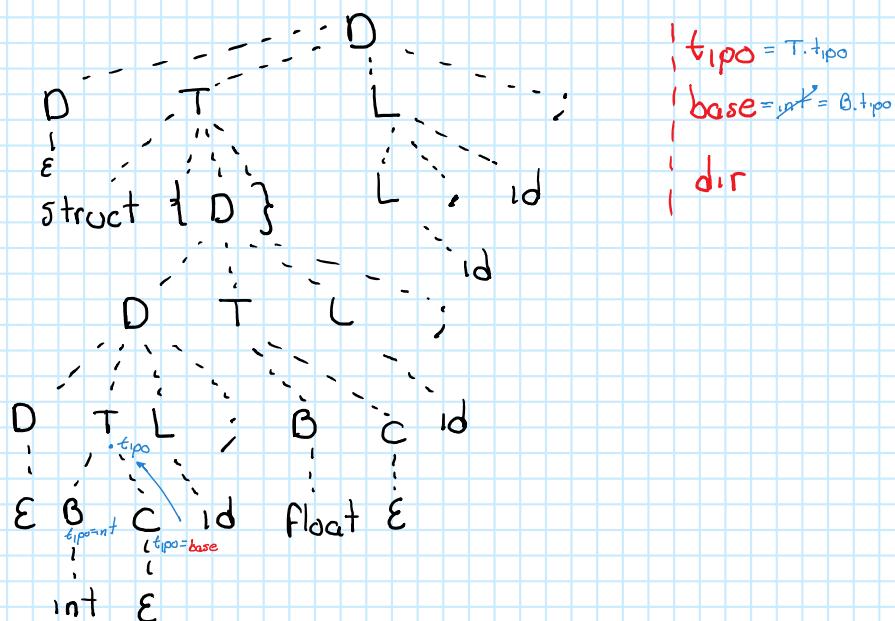
error(...);

### Ejercicio ③

Entrada:

```
struct {
    int a;
    float f;
} a, b;
```

// Árbol



stack

TABLA SIMBOLOS

Pos	ID	DIR	TIPO	VAR
0	a	0	0	var
1	f	4	1	var



TABLA DE TIPOS

TIPO	#ELEMENTOS	TipoBase
int	4	
float	4	
struct	8	

Tabla Símbolos del TOP stack

NOTA: Están "a" y "f" de fuera y dentro de la estructura varian respecto al "ALCANCE" de la variable en el código

estructura varian respecto al "ALCANCE"  
de la variable en el código

- Cada vez que se vean "{" se tendrá que hacer push de una tabla de tipos y simbolos en una pila(stack)
- Cuando se ve "}" se hace pop

## // Parte semántica

$D \rightarrow D \sqcup L$	$t_{\text{tipo}} = T, t_{\text{tipo}}$
$D \rightarrow \epsilon$	$\text{base} = B, t_{\text{tipo}}$
$T \rightarrow BC$	$T, t_{\text{tipo}} = C, t_{\text{tipo}}$
$T \rightarrow \text{struct } \{ \text{ID} \}$	$\text{PTS.push(new TS());}$ $\text{PTT.push(new TT());}$ $\text{PilaDir.push(dir);}$ $\text{dir = 0;}$ $\text{ts = PTS.pop();}$ $\text{tt = PTT.pop();}$ $\text{ts.setTT(tt);}$ $\text{T.tipo = topPTT.add(ts);}$ $\text{dir = PilaDir.pop();}$

$B \rightarrow \text{int}$	$B, t_{\text{tipo}} = 0$
$B \rightarrow \text{float}$	$B, t_{\text{tipo}} = 1$
$C \rightarrow [\text{num}] C,$	$\text{if num.tipo = 0}$ $\text{if num.var > 0}$ $\quad C, t_{\text{tipo}} = \text{topTT.add(num.val, c.tipo)}$ $\text{else}$ $\quad \text{error(" "});$ $\text{else}$ $\quad \text{error(" "});$

$C \rightarrow \epsilon$	$C, t_{\text{tipo}} = \text{base}$
$L \rightarrow L_1, \text{id}$	$\text{if topPTS.add(id.val, tipo, dir, "var") \neq 1}$ $\quad \text{dir = dir + topPTS.getSize(tipo)}$

$L \rightarrow L_1, id$

```
| if top PTS.add(idval, tipo, dir, "var") ≠ 1  
|   dir = dir + top PTG.getSize(tipo)  
| else  
|   error();
```

$L \rightarrow id$

```
| //Lo mismo
```

## 2. TTY y TS

Wednesday, October 30, 2019

2:33 PM

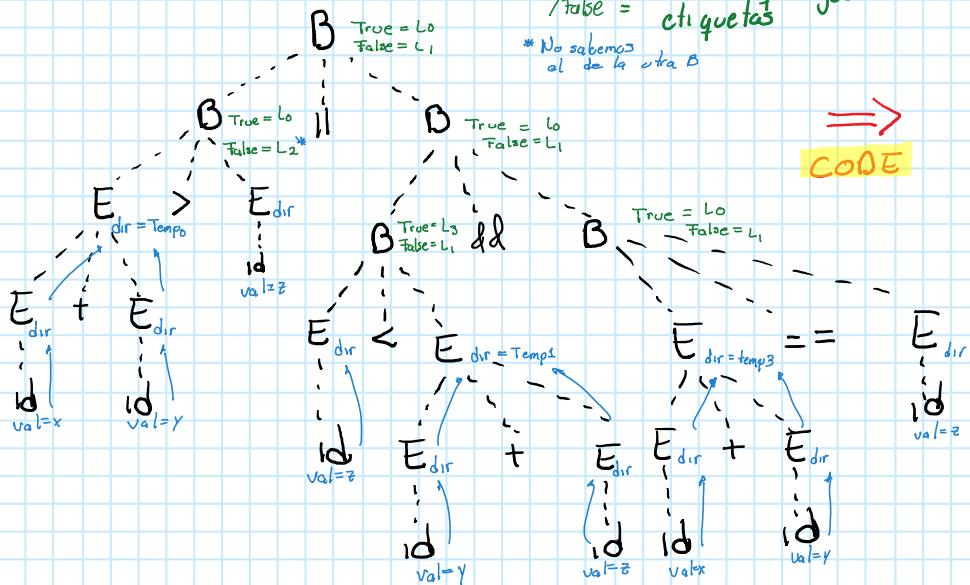
$$B \rightarrow B_1 \parallel B_2 \mid B_1 \& B_2 \mid !B_1 \mid E_1 \text{ oprel } E_2 \mid \text{true} \mid \text{false}$$

$$E \rightarrow E_1 + E_2 \mid \text{id} \mid \text{num}$$

CODE

$$\text{Cadena} \left\{ \begin{array}{l} x + y > z \parallel x < y + z \& x + z == w \end{array} \right.$$

True/False  $\triangleq$  Atributos que guardan etiquetas  
\* No sabemos al de la otra B



CODE

t0=x+y  
if t0>z goto L0  
goto L2 //else

L2: t1=y+z  
if x<t1 goto L3  
goto L1

L3: t2 = x + y  
if t2 == goto L0

L0: True code  
L1: Follow code

•  $B_1 \rightarrow B_1 \& B_2$

B. code

$B_1.\text{code}$

False

$B_1.\text{true}$

$B_2.\text{code}$

False

$B_1.\text{true} = \text{newLabel}()$

$B_1.\text{false} = B.\text{false}$

$B_2.\text{true} = B_1.\text{true}$

$B_2.\text{false} = B_1.\text{false}$

$B.\text{code} = B_1.\text{code}$

// Label( $B_1.\text{true}$ )

//  $B_2.\text{code}$

•  $B \rightarrow !B_1$

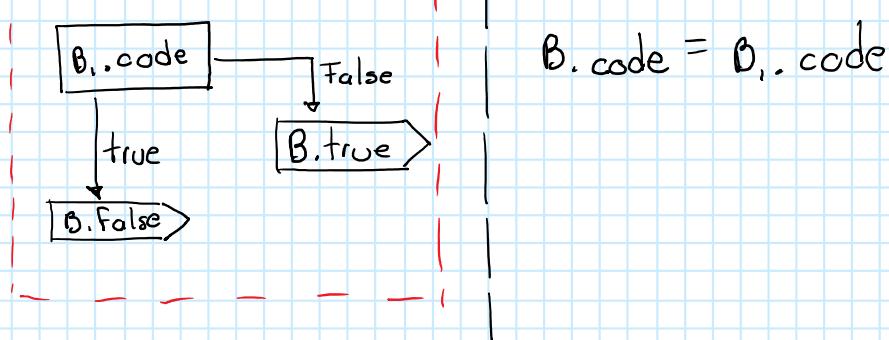
B. code

$B_1.\text{code}$

$B_1.\text{true} = B_1.\text{false}$

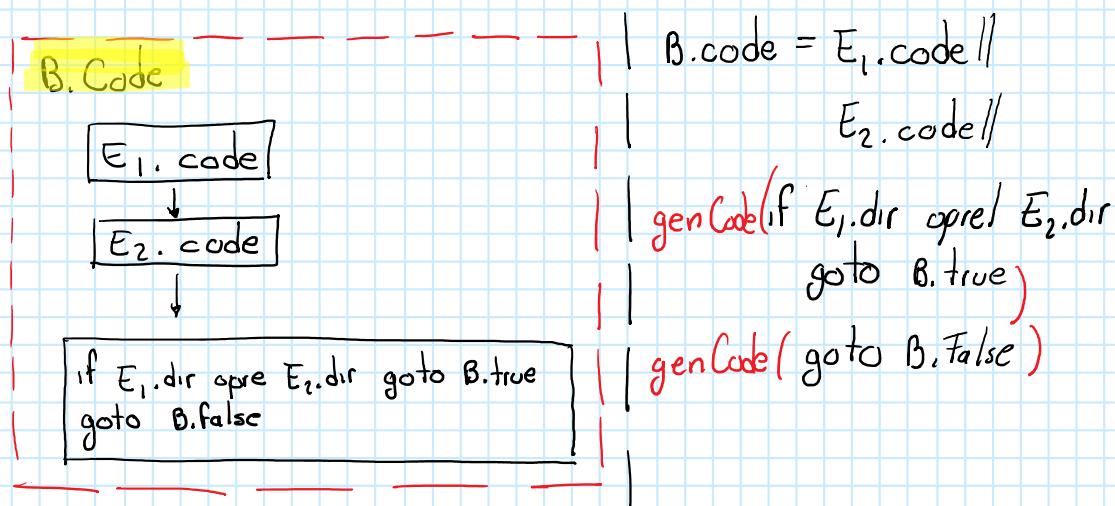
$B_1.\text{false} = B_1.\text{true}$

$B.\text{code} = B_1.\text{code}$



B.code = B<sub>1</sub>.code

- $B_1 \rightarrow E_1 \text{ ope } E_2$



B.code = E<sub>1</sub>.code ||

E<sub>2</sub>.code ||

genCode(if E<sub>1</sub>.dir opre E<sub>2</sub>.dir  
goto B.true)

genCode(goto B.false)

- $B \rightarrow \text{true} ; B.\text{code} = \text{genCode}(\text{goto } B.\text{true})$

- $B \rightarrow \text{false} ; B.\text{code} = \text{genCode}(\text{goto } B.\text{false})$

### Ejercicio

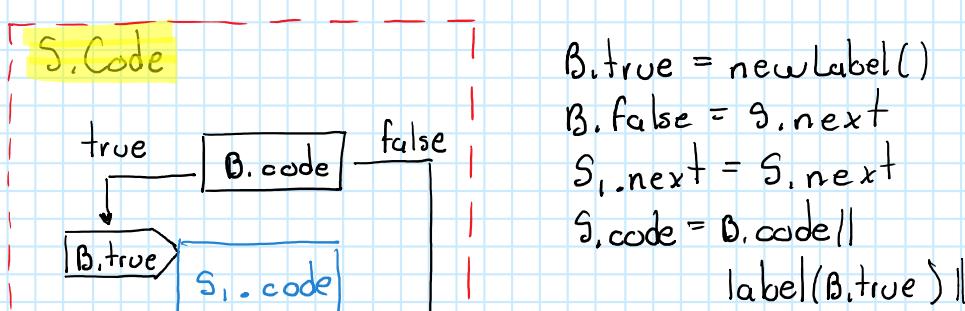
P → S

S → SS | ,f(B)S | ,f(B)S else S | while(B)S |

for(S;B;S)S | id = E ;

E → E oparit E | (E) | ,d | num

- $S \rightarrow ,f(B)S,$



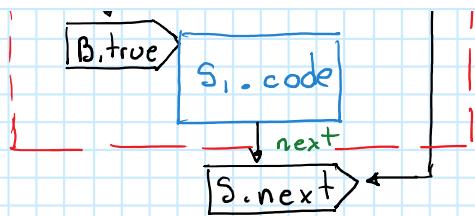
B.true = newLabel()

B.false = S.next

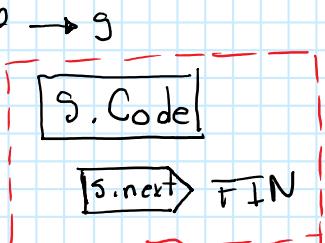
S<sub>1</sub>.next = S.next

S.code = B.code ||

label(B.true) ||

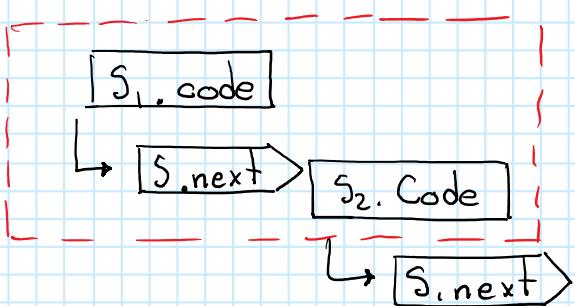


$S_1.code = B.code //$   
 $\text{label}(B.\text{true}) //$   
 $S_1.code$



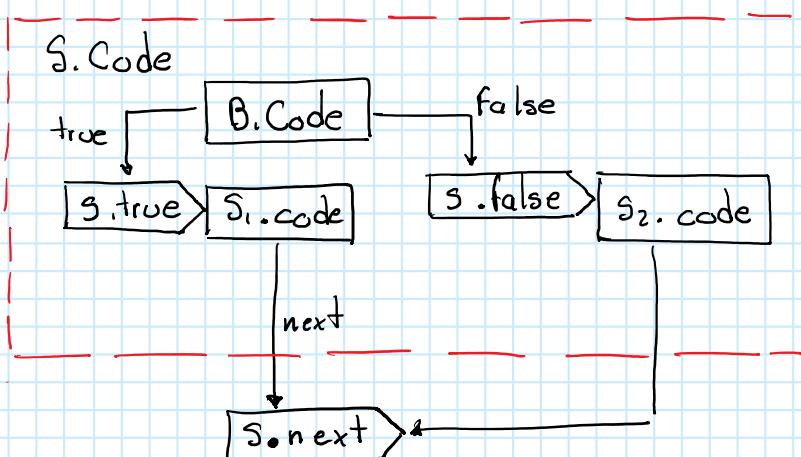
$S.next = \text{newLabel}()$   
 $P.code = S.code // \text{label}(S.next)$

•  $S \rightarrow SS ; \quad S \rightarrow S_1 S_2 \quad \times \text{ Esto mal}$



$S_1.next = \text{newLabel}()$   
 $S_2.next = \text{newLabel}()$   
 $S.code = S_1.code //$   
 $\text{label}(S.next) //$   
 $S_2.code$

•  $S \rightarrow \text{if } (B) S \text{ else } S$

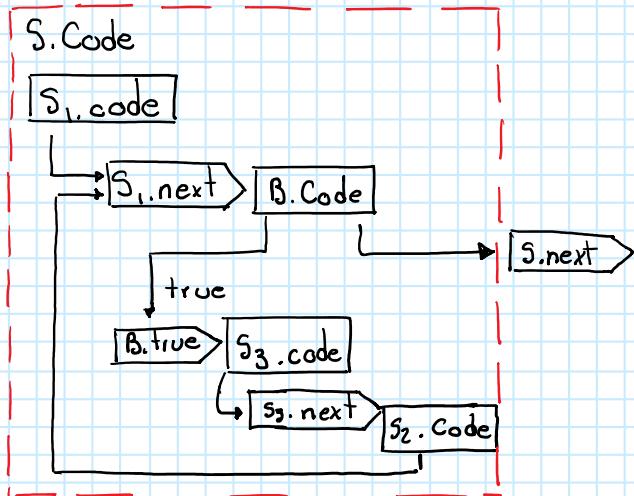


$B.\text{true} = \text{newLabel}()$   
 $B.\text{false} = \text{newLabel}()$   
 $S_1.next = S.next$   
 $S_2.next = S.next$   
 $S.code = B.code // \text{label}(B.\text{true}) //$   
 $S_1.code // \text{label}(B.\text{false}) // S_2.code$

// El código no  
puede ser etiqueta  
 $S_1.code = \underbrace{B.\text{true}}_{\text{Etiqueta}}$

$S_0.code = S_0.code \parallel \text{Label}(B.\text{true}) \parallel$   
 $S_1.code \parallel \text{Label}(B.\text{false}) \parallel S_2.code$

- $S \rightarrow \text{for}(S_1; B; S_2) S_3$

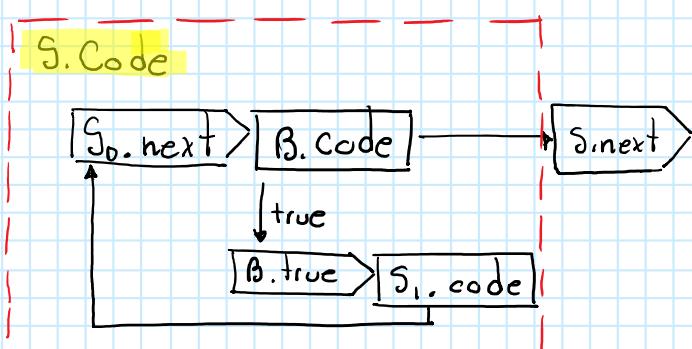


$S_1.next = \text{newLabel}()$   
 $S_3.next = \text{newLabel}()$

$B.\text{true} = \text{newLabel}()$   
 $B.\text{false} = S_1.next$

$S_0.code = S_1.code \parallel \text{Label}(S_1.next) \parallel B.code \parallel B.\text{true} \parallel$   
 $S_3.code \parallel \text{Label}(S_3.next) \parallel S_2.code$

- $S \rightarrow \text{while}(B) S$



$S_0.next = \text{newLabel}()$   
 $B.\text{true} = \text{newLabel}()$   
 $B.\text{false} = S_0.next$   
 $S_1.next = S_0.next$

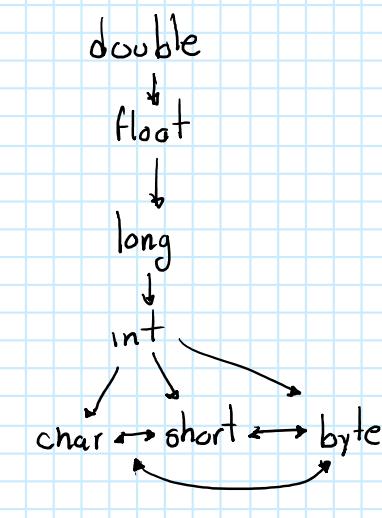
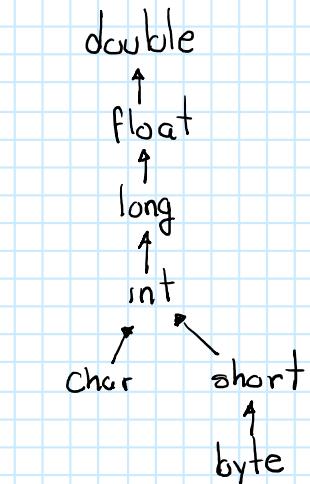
$S_0.code = \text{label}(S_0.next) \parallel B.code \parallel$   
 $\text{label}(B.\text{true}) \parallel S_1.code$

- $S \rightarrow id = E;$

```

if top PTS.get(id) ≠ -1 then
  if top PTS.getType(id) = E.type then
    S.code = genCode(, id = E.dir)
  else
    error("Incompatibilidad tipos")
  else
    error("Variable no declarada")
  
```

## Ampliación y reducción de tipos



`tipo max(tipo t1, tipo t2)`

`dir ampliar(tipo t1, tipo t2)`

$E \rightarrow E_1 + E_2$

$E.tipo = \max(E_1.tipo, E_2.tipo)$

$E.dir = \text{newTemp()}$

$\alpha_1 = \text{ampliar}(E_1.tipo, E.tipo)$

$\alpha_2 = \text{ampliar}(E_2.tipo, E.tipo)$

$E.code = E_1.code || E_2.code || \text{genCode}(E.dir = \alpha_1 + \alpha_2)$

max y ampliar

↳ Pseudocode (Libro)

# Direccionamiento Arreglos

miércoles, 6 de noviembre de 2019 13:15

$$E \rightarrow E_1 + E_2$$

$$E.\text{tipo} = \max(E_1.\text{tipo}, E_2.\text{tipo}) \quad // El más grande entre los valores$$

$$E.\text{dir} = \text{newTemp}()$$

$$\alpha_1 = \text{ampliar}(E_1.\text{tipo}, E.\text{tipo}, E.\text{dir})$$

$$\alpha_2 = \text{ampliar}(E_2.\text{tipo}, E.\text{tipo}, E_2.\text{dir})$$

$$E.\text{code} = E_1.\text{code} || E_2.\text{code} || \text{genCode}(E.\text{dir} = \alpha_1 + \alpha_2)$$

```
dirAmpliar(tipo t1, tipo t2, dir d){  
    if (t1 == t2){  
        return d  
    }else if (t1 == int && t2 == float){  
        t = newTemp()  
        genCode(t = (float)d)  
        return t  
    }else if (...) {  
        ...  
    }else {  
        error()  
    }  
}
```

## ► Direccionamiento de elementos de un arreglo

- $a[i]; \text{base} + i * \text{size}$
- $a[i][j]; \text{base} + i * \text{tamaño}_i + j * \text{tamaño}_j$

## // Gramática

$$S \rightarrow \text{id} = E \mid L = E$$

$$L \rightarrow \text{id}[E] \mid L [E]$$

$$E \rightarrow E \oplus E \mid \text{id} \mid L$$

Puede ser cualquier operador

$$S \rightarrow \text{id} = E \mid \text{if } (\text{top PTS.get(id)} \neq -1) \text{ }\alpha = \text{reducir}(\text{top PTS.getTipo(id)}, E.\text{tipo}, E.\text{dir}) \text{ }\text{genCode}(\text{id} = \alpha) \mid \text{else }$$

```

    | genCode( .id = α )
    | else
    |   error("Variable no declarada")
  S → L = E | α = reducir(L.tipo, E.tipo, E.dir)
  | genCode( L.id [L.dir] = α ) ← a[i] = t₀
  | q[i] = 4 + 3

```

CODE |  $t_0 = 4 + 3$   
 3 DIR |  $t_1 = i * 4 \leftarrow \text{base} + (i * \text{size})$   
 |  $a[t_1] = t_0$

S → i.d [E] // Validar que el tipo de E sea entero

NOTA • La parte semántica se da  
 antes y después de la compilación  
 ∵ No sabemos valor de "E"  
 $\rightarrow a[-1] = 3;$

```

  | if (topPTS.get(.id) ≠ 1) {
  |   if (E.tipo = int) {
  |     L.id = .id
  |     L.tipo = topPTT.getTipoBase(id.tipo)
  |     L.dir = newTemp() // Nueva dirección
  |     t₁ = i * 4 → genCode( L.dir = "E.dir * topPTT.getTipoBase(L.tipo) )
  |   } else
  |     error();
  }
```

// Ejemplo

int [3][2] a;      Más a la derecha = Más al fondo  
 ↓ ↓ ↓      o base de la tabla

TABLA DE TIPOS

ID	TIPOS	SIZE	#ELEMENTS	TIPO BASE
0	int	4	—	—
1	float	4	—	—
2	array	8	2	0
3	array	24	3	2

$a[i][j] = 4$  |  $t_0 = i * 8$  3.tipoBase(2.size) ↑  
 |  $t_1 = j * 4$  0.tipoBase(0.size) ↑  
 |  $t_2 = t_0 + t_1$   
 |  $a[t_2] = 4$

```

L → L1[ε] | if ( E.tipo = int ) {
|   L.dir = L1.dir
|   L.tipo = topPPT.getTipoBase(L1.tipo)
|   t = newTempo()
|   L.dir = newTempo()
t1 = j * 4 → genCode( t = "E.dir" * topPPT.getSize(L.tipo) )
t2 = t0 + t1           genCode( L.dir = L1.dir + t )

```

**NOTA:** sintetizados porque pertenecen al encabezado de la producción

```

E → id | if ( topTS.getId(id) ≠ -1 ) {
|   E.dir = id
|   E.top = topTS.getTipo(id)
| } else
|   error ("Id no declarado");
}

```

```

E → L | // En code de alto nivel; a = b[i]
|   E.dir = newTempo()
|   E.tipo = L.tipo
|   genCode( E.dir = "L1.dir["L.dir"]" )

```

- Ejemplo

a = b[i][j]	t <sub>0</sub> = i * 8
	t <sub>1</sub> = j * 4
	t <sub>2</sub> = t <sub>0</sub> + t <sub>1</sub>
	t <sub>3</sub> = b[t <sub>2</sub> ]
	a = t <sub>3</sub>

# Funciones

miércoles, 6 de noviembre de 2019 14:32

D → define T id(F) { S }

F → T id | F , T id

S → return E; | ss | id=E;

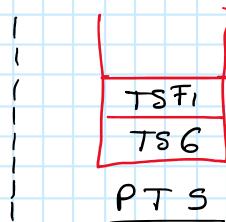
E → id(A)

A → E | A, E

T → int | float

- Tabla de Símbolos Global → Checar que no existe el identificador de la función

- Posteriormente se crea una tabla de símbolos asociada a la función



## // Ejemplo

```
define suma(int a, int b){  
    int c;  
    c = a+b;  
}
```

POS ID DIR VAR ARGS

0 a 0 param —

1 b 4 param —

2 c 8 var —

0 suma — Función

int, int  
0,0 ← Lista de sus argumentos

- Validar que S tenga una expresión de retorno además de verificar que el tipo del retorno sea el mismo que el de la función

↳ Tenemos que reducir ya que no podemos cambiar el tipo → Casting

- F → id // Se crea la lista de los argumentos

- S → return E // Se puede crear una lista donde se almacenen todos los tipos de los retornos

```
if (...)      se almacenen todos los tipos de  
    return 1;  los retornos  
else         return 0;
```

- $E \rightarrow \text{id}(A)$  // Para llamar a una función

- ① Hay que buscar en TS Global
  - ② Revisar que sea una función
  - ③ Validar que la cantidad de parámetros sea la misma

→ Crear lista de los parámetros

Lista de A con T5 global

(Comparar)

$$A \rightarrow E | A, \bar{E}$$

# Libro Alfa y Omega

← Dragón

## Switch y Recursividad

Tareg

$s \rightarrow \text{return } E_i \mid ss \mid id = E;$

$$E \rightarrow id(A)$$

$$A \rightarrow E | A, E$$

$T \rightarrow \text{int} | \text{float}$

$S \rightarrow \text{return } E$   
 $ss | id = E$

```

    | , f ( get IDTG(, d) = 1 )
    |   if (get T, tipoTG(T, , d) = - 1 )
    |     if (S, tipo = T, tipo = 1 )
    |       T, lista + a = T, lista + a
    |       TS, add(, d, T)
    |
    | T, param = crearListaPC()
    | T, param.add(T, tipo, , d)
    |   if ( S, listaReturn == Null )
    |     S, listaReturn = crearListaRC()
    |     S, listaReturn(E)
    |     S, tipo = E, tipo

```

$E \rightarrow id$  (A)

```
|  
| s.tipo = E.tipo  
| ,f getIDTS(id) = 1  
|   id = E.id  
| else  
|   error();  
| if getIdTS(id) = -1  
|   if getVarTg(id) = Función  
|     A.lista = crearlistaP()  
|     E.lista = A.lista  
|   else  
|     error();  
| else  
|   error();
```

$A \rightarrow E | AE$

```
A.lista.add(E)
```

$T \rightarrow int | float$

```
T.tipo = int  
T.tipo = float
```

# TAC

Lunes, 11 de noviembre de 2019

13:29

## Ejemplo

```
int x,y;
```

```
define int suma (int a, int b) { define int main() {
```

```
    int c;
    c = a + b;
    return c;
}
```

```
    int a,b,c;
    a = 3 + 4 * 2;
    b = a * 8 / (4 + a);
    c = suma(a,b);
    return 0;
}
```

Pos Id Tipo Dir Var Args

TS6

0	x	0	0	var	=
1	y	0	4	var	=
2	suma	0	-	Funcióñ	0,0
3	main	0	-	Funcióñ	-

Id Tpo Tam #Elem TpoBase

0	int	4	=	=
1	float	4	=	=

Nota | PilaDir = 8 //Forma global

TSuma	0	a	0	0	param	-
	1	b	0	4	param	-
	2	c	0	8	var	-

0	int	4	=	=
1	float	4	=	=

\*Lista de parámetros que guarda los tipos

TT → TTsumma

TSmain	0	a	0	0	Var	=
	1	b	0	4	Var	=
	2	c	0	8	Var	=

0	int	4	=	=
1	float	4	=	=

Pila Tabla de Símbolos

Pila de Tabla de Tipos

// Código de 3 direcciones → Con base a la etiquetas

\* Preguntar si están en tabla y tienen el mismo tipo

suma : t<sub>0</sub> = a + b

$c = t_0$

return c

main:

$t_0 = 4 * 2$

$t_1 = 3 + t_0$

$a = t_1$

$t_2 = a * 8$

$t_3 = 4 * 9$

$t_4 = t_2 / t_3$

$b = t_4$

param q

param b

$t_5 = \text{call suma}(2) // \text{or 2 parámetros}$

$c = t_5$

return 0;

① Sacar tabla de tipos Suma y asignar el apuntador en la Tabla de Símbolos

② 0,0 se coloca hasta que se hace el pop de Tsuma

// Con las estructuras cambia un poco

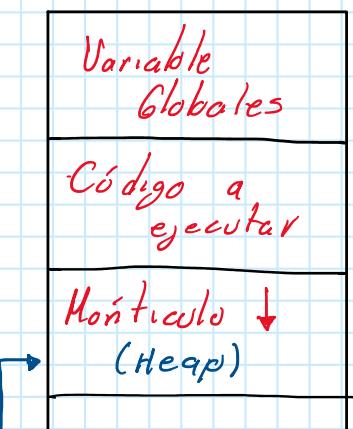
③ — se elimina la tabla de main y se indica que no tuvo argumentos.

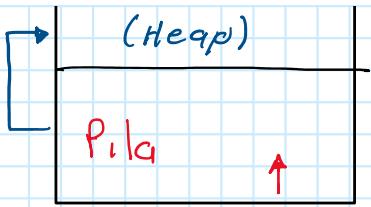
## Assembly Code

- Registro de activación
  - Parámetros Actuales
  - Valor Devuelto
  - Enlace de control
  - Enlace de acceso
  - Estado de la máquina
  - Variables de locales
  - Variables temporales

## Visualización

### Memoria





- Pila | Memoria estática donde se guardan registros de funciones

- Montículo | Es la parte de la memoria dinámica (Aquí se llama a "Malloc" o "Calloc")

- Estado de la máquina | • El contador de programa, si los valores en registros se deben llevar a la memoria
  - Las banderas de correo.

- Enlace de acceso | Sirve para localizar datos que necesita el procedimiento al que se llamó pero que se encuentra en otro registro de activación

- Enlace de control | Apunta el registro de activación del proceso que hizo la llamada

- Valor devuelto | En esa dirección de memoria se coloca el valor retornado por la función.

- Parámetros Actuales | Almacena el valor de los parámetros para que la función realice cálculos.

### ► Registro de Activación

// Cuestiones tomadas

-2 Dir de retorno

-1 Valor de retorno

// Para función Suma

-5

-4

← La de retorno sólo un byte  
← -4(SP)

$\lambda$	Dir de retorno	-5	50 ..
-1	Valor de retorno	-4	* -4(SP)
0	Parámetros	0	* 0(SP)
n	Variables locales	4	* 4(SP)
:			
m	Variables temporales	8	* 8(SP)

Suma:

```

LD R1, *0(SP)
LD R2, *4(SP)
ADD R3, R1, R2
ST R3, *8(SP)
ST R3, *-4(SP)
BR *-5(SP)

```

main:

```

LD R1, #4
LD R2, #2
MUL R3, R1, R2
LD R4, #3
ADD R5, R3, R4 // R5 =
ST R5, *0(SP)
LD R6, #8
MUL R7, R5, R6
ADD R8, R1, R5
DIV R9, R7, R8
ST R9, *4(SP)
SUB SP, SP, #17
{ ST *-5(SP), Dir Ret
  ST *0(SP), R5
  ST *4(SP), R9
}
  | Tamaño de registro de
  | main
  | • 12 de main
  | • 5 de dirección de
  | forma negativa
(Dirección) BR Suma
Valor → LD R10, *-1(SP)
retornado ADD SP, SP, 17

```

<i>Valor retornado</i>	LD	R10, * - 4(SP)		forma negativa
	ADD	SP, SP, 17		
	ST	R10, * 8(SP)		
	LD	R11, #0		
	ST	R11, * - 4(SP)		
	HALT			

# TAC

viernes, 15 de noviembre de 2019

13:17

$L_0: \text{if } c \text{ goto } L_1$

$$x = y + 1$$

$$y = 2 * z$$

$\text{if } d \text{ goto } L_2$

$$x = y + z$$

$L_2: z = 1$

goto  $L_0$

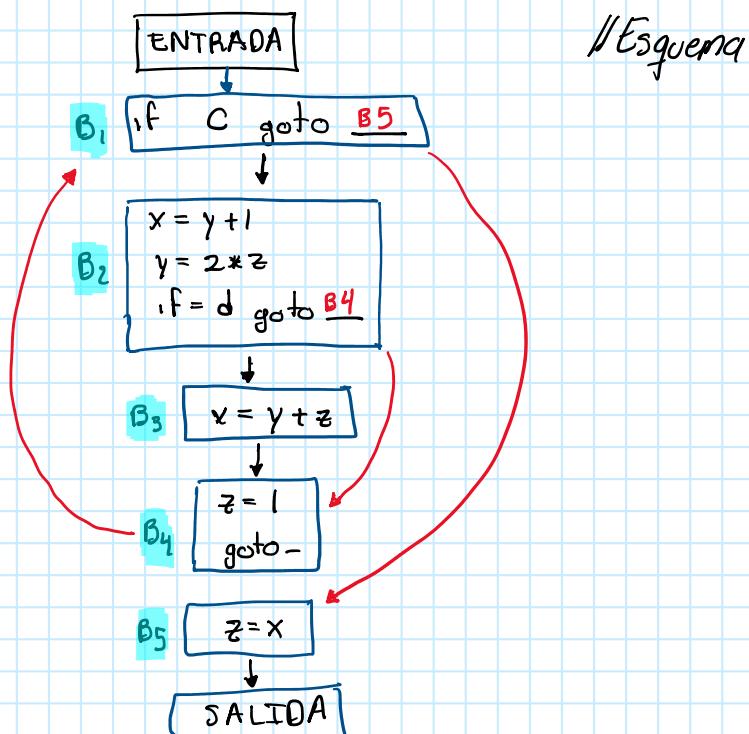
$L_1: z = x$

## Bloque Básico

- Secuencia de instrucciones de 3 direcciones que no contiene saltos excepto en la última instrucción

► Debemos encontrar instrucciones líderes (Primera linea del Bloque Básico):

- ① El código intermedio es líder
- ② Cualquier instrucción que sea el destino de un salto es líder  $\hookrightarrow \text{goto}$
- ③ Cualquier instrucción después de un salto es líder



$$ENT[I] = \underbrace{Var\ entrada}_{} + \underbrace{Var\ salida}_{} - \underbrace{Var\ Definidas}_{} \cup \underbrace{Var\ Uso}_{} \quad \boxed{I}$$

$$SAL[B] = \bigcup ENT[B]$$

$B' \in Sucesor(B)$

	Definidas	Uso	Entrada	Salida
$B_1$	—	C	X, C	X, Y, Z, C, D
$B_2$	X	Y	Y, Z, C, D	X, Z, C, D
$B_3$	Y	Z	X, Z, C, D	X, Y, Z, D, C
$B_4$	—	D	X, Y, Z, D, C	X, C, Y, Z
$B_5$	X	Y, Z	Y, Z, C	X, C
$B_6$	Z	—	X, C	X, C
$B_7$	—	—	X, C	X, C
$B_8$	Z	X	X	—

$$\begin{array}{l} B \stackrel{\Delta}{=} \text{Bloque} \\ I \stackrel{\Delta}{=} \end{array}$$

$$SAL[ENTRADA] = \emptyset$$

$$ENT[SALIDA] = \emptyset$$

$$SAL[B_{log}] = SAL[\text{var. sal } 1 \text{ ultimo previo}]$$

$$ENT[B_{log}] = ENT[\text{var. entra}]$$



- \* Definidas → Lado izquierdo de una igualdad
- \* Uso → los que están del otro lado

\* NOTA: Parece a un ciclo  
 $\therefore$  Regresamos y reevaluamos

	DEFINIDOS	USO	ENTRADA	SALIDA
$B_1$	—	C	X, Y, Z, C, D	X, Y, Z, C, D
$B_2$	X	Y	—	—
$B_3$	Y	Z	—	—
$B_4$	Z	—	—	X, Y, C, D
$B_5$	—	—	—	—
$B_6$	Z	X	X	X

Hacemos  
otra pasada

► Generación de Código Objeto

- a) Para una instrucción de la forma  $x = y + z$   
 + 1 cualquier operador binario
- 1] Llamar a la función obtener registro para  $x = y + z$  que selecciona los registro para "x, y, z".  
 Nombrarlos como  $rX$ ,  $rY$ ,  $rZ$
  - 2] Si y no está en  $rY$ , generar una instrucción de carga
  - 3] De forma similar, si z no está en  $rZ$   
 4] Generar la instrucción para la suma, usando  $rX, rY$  y  $rZ$
- b) Instrucción forma  $x = y$ , suponer que obtener registro siempre seleccionará el mismo para "x" y "y"  
 Si y no está en registro generar instrucción de carga

Funció n obtener registro

- c) Para una instrucción  $x = y + z$  hacer lo siguiente
- 1] Si y se encuentra en un registro elegir ese registro como  $rY$ .  
 No generar instrucción de carga
  - 2] Si y no está en un registro, pero hay uno vacío seleccionamos ese registro como  $rY$
  - 3] Si y no está en un registro y no hay registros vacíos  $\therefore$  Liberamos registro  
 Suponga que R es candidato y que v es la variable almacenada en R.  
 $\hookrightarrow$  Se debe asegurar que el valor de v ya no se necesita o exista otras formas de reciclarlo

→ Se debe asegurar que el valor de  $v$  ya no se necesite o exista alguna forma de recuperarlo opciones:

- ① Si el descriptor de dirección para  $v$  dice que  $v$  se encuentra en otro lado aparte de  $R$ , estamos bien //
- ② Si  $v$  es  $x$  (o sea el mismo), entonces seleccionamos ese registro
- ③ Si  $v$  no es una variable viva más adelante entonces seleccionar registro // O sea, no se usará en bloques de adelante
- ④ Si no se cumple ninguno de los anteriores generar instrucción de carga del registro a la dirección de  $v$ .  
Seleccionar el registro

#### d) Seleccionar para registros de $X$

- Si  $x$  está en un registro, seleccionar ese registro
- Se aplica incluso si " $y$ " y " $z$ " son " $x$ "  
↳ " $x = x + z$ " o " $x = y + z$ "
- Si  $y$  no se utiliza después de la instrucción de la misma forma que  $v$  seleccionar a  $Ry$  como  $Rx$
- Se aplica lo mismo para  $Rz$

#### e) Obtención de registros para $x = y$

- Seleccionar  $Ry$  como en los puntos anteriores y hacer  $Rx = Ry$

- Seleccionar  $R_y$  como en los puntos anteriores y hacer  $R_x = R_y$

## ► Administración de los descriptores de registro y de dirección

① Para una instrucción de carga → LD R<sub>i</sub>, X

- a) Modificar el descriptor de registro de R de forma que sólo contenga a X
- b) Modificar el descriptor de dirección para X, agregando a R como ubicación adicional

② Para una instrucción de almacenamiento → ST R, X

- a) Modificar el descriptor para X de manera que incluya su propia ubicación en memoria

③ Para una operación de la forma → OP Rx, Ry, Rz

- a) Modificar el descriptor de registro para Rx de manera que sólo contenga a X
- b) Modificar el descriptor de dirección para X de manera que su única ubicación sea Rx
- c) Eliminar Rx del descriptor de dirección de cualquier variable distinta de X

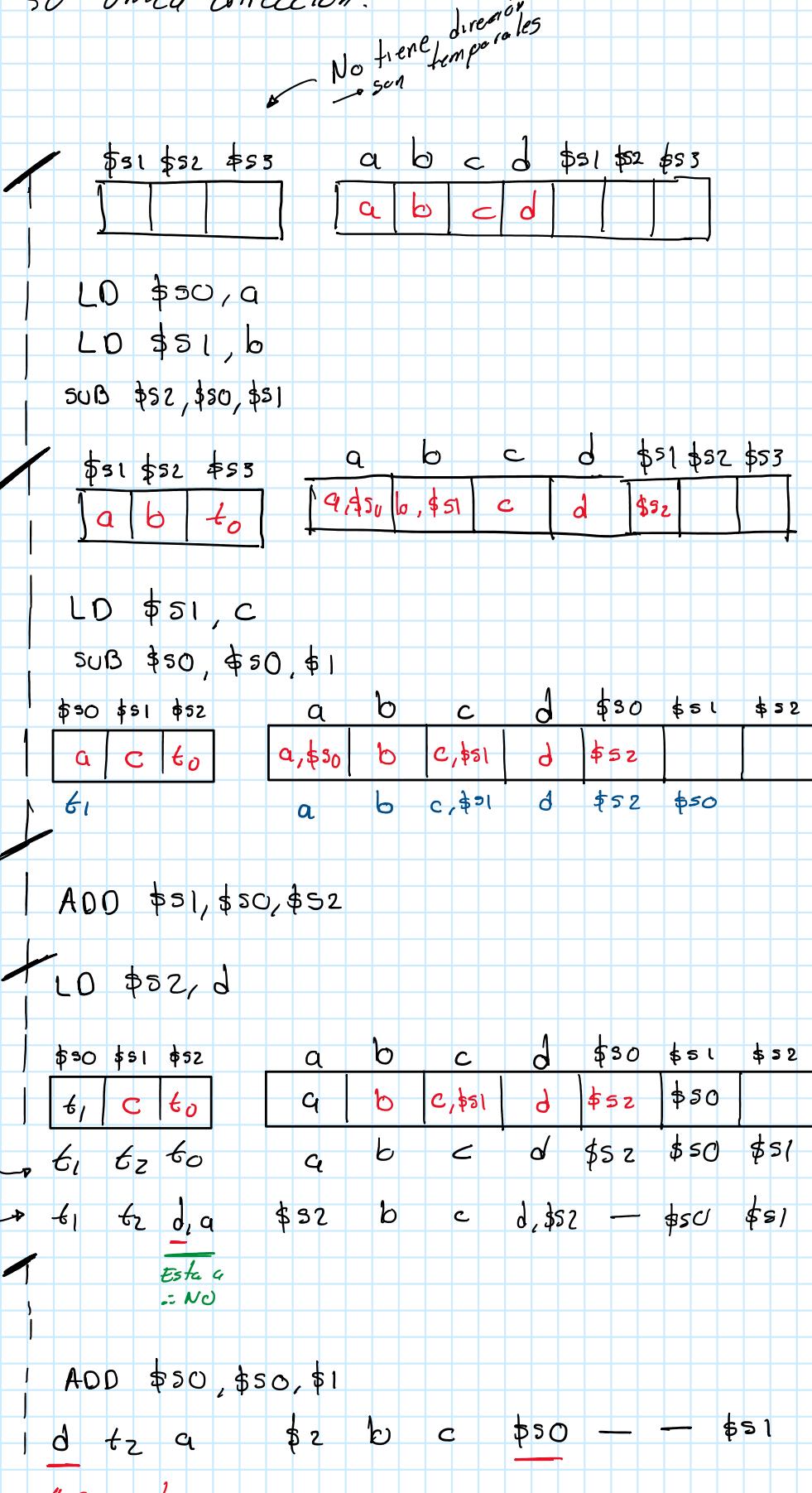
④ Para la instrucción → X = Y

- Después de administrar los directores de acuerdo a la regla 1:
- a) Asignar X al descriptor de registro para Ry

- a) Agregar  $X$  al descriptor de registro para  $Ry$
  - b) Modificar el descriptor para  $Rx$  de manera que  $Ry$  sea su única dirección.

## //Ejercicio

$$t_{ij} = a - b$$



| d t<sub>2</sub> a \$2 b c \$50 — — \$51

// Garantizar

sw \$50, d

sw \$52, a

# ENSAMBLADOR

Wednesday, 20 November 2019 13:21

$$t_1 = x + y$$

## // ENSAMBLADOR

```
LW $t0, x
LW $t1, y
ADD $t2, $t0, $t1
SW $t2, [t1]
```

## TABLA SIMBOLOS

POS	ID	TIPO	DIR	ARGS	VAR
0	x	0	0	-	Var
1	y	0	4	-	var
2	t1	0	8	-	temp

// Es caso de que estuviera dentro de una función

```
LW $t0, 0($SP)
LW $t1, 4($SP)
ADD $t2, $t0, $t1
SW $t2, 8($SP)
```

// Función de registro de activación

► MIPS

(Ensamblador) →

SIMULADOR  
"MARS"

- Registros Temporales : \$t0 - \$t9
- Registros "s" : \$s0 - \$s7 ← Programar para que los guarde en la pila  
\$zero ← Almacena un cero lógico (No se puede modificar)

\$v0, \$v1 ← Valor de retorno de las funciones  
Para interrupciones ↑

\$GP ← Global pointer // No recomendable

\$SP ← Stack Pointer

\$ra ← Guarda dirección de retorno al llamar funciones

\$fp ← // No recomendable

\$a0,...,a3 ← Sirve para almacenar parámetros de una función

// Si son más de 3 se usa registro de activación

## #f0 - #f13 ← Aritmética "float" y "double"

## ▶ Instrucciones

- Aritméticas
    - add R<sub>0</sub>, R<sub>1</sub>, R<sub>3</sub>
    - add.s // Flotante
    - add.d // double
    - sub / sub.s / sub.d
    - mul / mul.s / mul.d
    - div / div.s / div.d
  
  - Carga
    - lb // Sólo carga un byte
    - li // Carga inmediata - Modo direccionamiento inmediato
    - l.s / l.d
    - lw // Carga 4 bytes
    - lwcl // Carga flotantes
  
  - Almacenamiento
    - sw Registro, Dirección
    - swc1
    - sb
    - // NOTA Si no existe
  
  - Movimiento
    - move R<sub>1</sub>, R<sub>2</sub>
      - ↑ Origen
      - ↑ Destino
  
  - Salto
    - begz // Salta si es igual o mayor a zero
    - beq R<sub>1</sub>, R<sub>2</sub>, Et // Salta si ambos tienen lo mismo
    - beg
    - bgte // Salta si el del primero es mayor
    - blte // Salta si el del primero
- condicionales*

<u>cond</u>	<ul style="list-style-type: none"> <li>• blt e</li> <li>• bne</li> </ul>	<ul style="list-style-type: none"> <li>// Salta si el del primero es mayor</li> <li>// Salta si el del primero es menor</li> </ul>
<u>Instrucción</u>	<ul style="list-style-type: none"> <li>• j</li> <li>• jr</li> <li>• jal</li> </ul>	<ul style="list-style-type: none"> <li>// Salta a etiqueta</li> <li>// Seguido de un registro</li> <li>// Salta a una subrutina</li> </ul>

<span style="background-color: yellow;">•</span> Lógicas	<ul style="list-style-type: none"> <li>• and</li> <li>• or</li> <li>• nor</li> <li>• xor</li> </ul>
--	---

<span style="background-color: cyan;">//</span> Casos	addi	← Inmediata
Especiales	addiq	← Enteros sin signo

### ► Estructura

data  $\triangleq$  Se definen variables globales y cadenas de texto

```
x: . word //Entero | str: . ascii "Hola mundo\n"
y: . float |
z: . double |
a: . byte |
```

```
.text
.globl main
main: $V0,4
    $a0,str
    syscall
    $V0,10 // retorna control al sos.
    syscall
```

## syscall

### Ejemplo

```

int a, b;
int suma(int x, int y) {
    int z;
    z = x + y;
    return z;
}

```

```

int main() {
    int z;
    a = 3 + 2;
    b = a * 4;
    z = suma(a, b);
    return 0;
}

```

### // TAC

```

suma: t0 = v0 + 4
      v8 = t0
      return v8

main: t1 = 3 + 2
      a = t1 // se pone directo
      t2 = a * 4 // se pone directo que es global
      b = t2
      param a
      param b
      t3 = call suma, 2
      z = t3
      return 0

```

### Lenguaje Ensamblador

```

.data
a: .word
b: .word
.text
.global main
summa: lw $t0, 0($sp)
        lw $t1, 4($sp)
        add $t0, $t1
        sw $t0, 8($sp)
        jr $ra

```

```

main: li $t0, 3
      add $t0, $t0, 2
      sw $t0, a

```

### \* mult

→ Guardar en  
\$t0  
\$t1

✓ No usar

PASO  
MORTAL

No 8 smo 12  
bytes

```

mul $t1, $t0, 4
sw $t1, b
addi $sp, $sp, -12
sw $t0, 0($sp)
sw $t1, 4($sp)
jal summa
lw $t0, 8($sp)
addi $sp, $sp, 12

```

lw \$t<sub>0</sub>, 8(\$sp)

addi \$sp, \$sp, 12

li \$v<sub>0</sub>, 4

move \$a0, \$t<sub>0</sub>

syscall

li \$v<sub>0</sub>, \$

syscall

## EXTRA

Friday, 22 November 2019 11:44 AM

- La **recursividad** en el TAC sólo es una llamada a la función

↳ Hay que tener cuidado lo siguiente:

- Registro de Activación

↳ En la pila hay que mencionar cuantas llamadas se hacen

- Lista o variable global para el tipo de la función