

Tema 9: Generación de Código Intermedio

1. Introducción
2. Código de tres direcciones
3. Generación de código intermedio mediante gramáticas de atributos
4. Estructuras y elementos de una matriz
5. Expresiones lógicas
6. Referencias no satisfechas. Relleno por retroceso
7. Instrucciones de control de flujo
8. Llamadas a subprogramas

1. Introducción

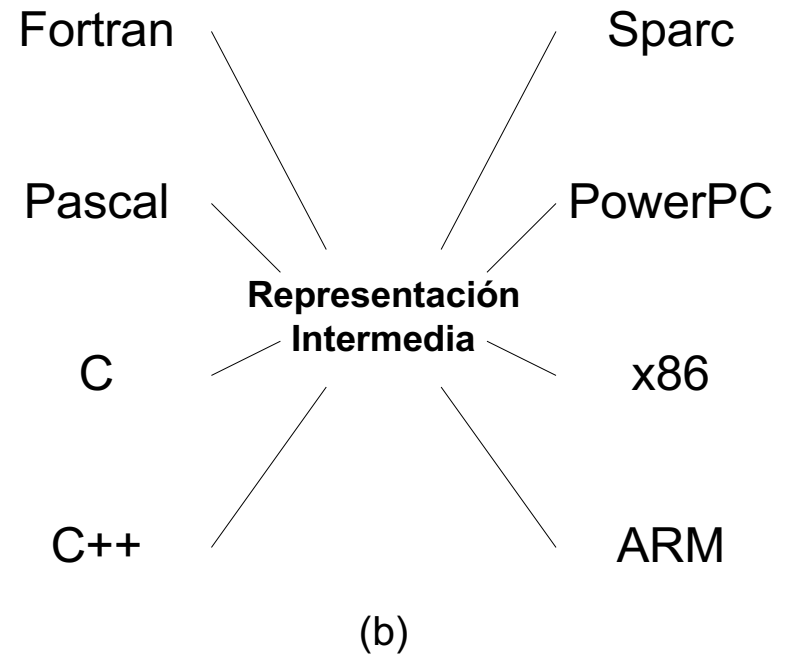
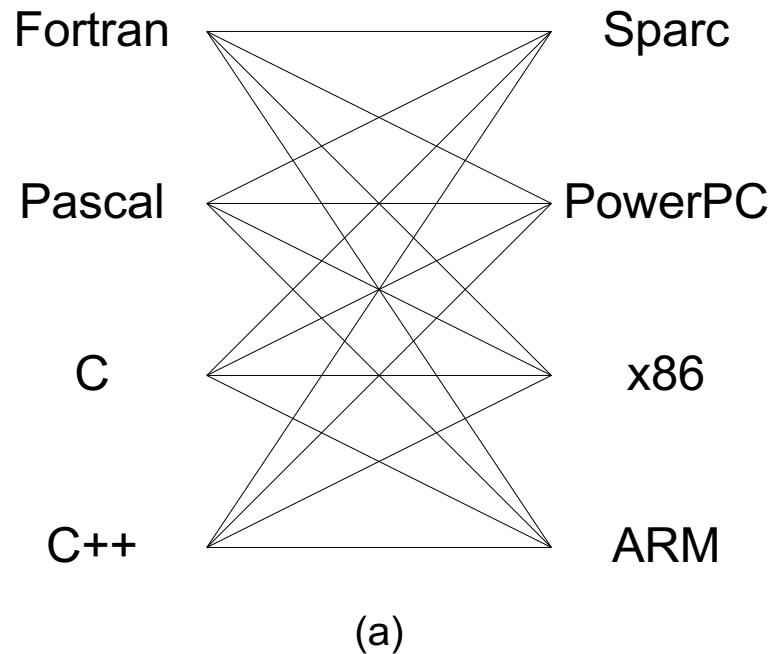
¿Qué lenguaje usar como destino del proceso de traducción?

- Otro lenguaje de alto nivel
- Código máquina
- Representación intermedia

Ventajas de usar representación intermedia

- Permite la división del proyecto en varias fases
- Facilita la reutilización de la etapa inicial (front-end) del compilador
- Se facilita la aplicación de transformaciones de mejora al código independientes de la arquitectura destino

Representaciones intermedias



Tipos de representaciones intermedia

- Grafos Dirigidos Acíclicos (GDA)
- Árbol de sintaxis abstracta (AST)
- Formato SSA (Static Single-Assignment)
- Código de tres direcciones

2. Código de 3 direcciones

Juego de instrucciones

Asignación	$x := y \text{ op } z$
Salto incondicional	goto E
Salto condicional	if $x \text{ op } y$ goto E
Pila de activación	push x $x := \text{pop}$
Llamada y	call E
Retorno de subprograma	ret
Asignaciones relativas	$a[i] := x$ $x := a[i]$
Acceso a la pila	FP TOP

3. Generación de CI mediante gramáticas atribuidas

Generación de CI para una asignación

Usaremos un procedimiento **emite** con un efecto colateral: Almacena código intermedio

Ej.

emite (x `:= ' 5 '+' 9)

SIGINST: Variable global con el número de la "SIGuiente INStrucción"

Asig \rightarrow id = E

E \rightarrow E + E

E \rightarrow id

Asig \rightarrow id = E	{ Asig.pos := BuscaPos (id.nom) ; emite (Asig.pos `:= ' E.pos); }
E \rightarrow E ₁ + E ₂	{ E.pos := CrearVarTemp (); emite (E.pos `:= ' E ₁ .pos '+' E ₂ .pos) }
E \rightarrow id	{ E.pos := BuscaPos (id.nom) }

Árbol anotado: Ejemplo de generación de CI

Cadena: $a = b + c$

Nom	tipo	posición [nivel, desp]
a	tentero	[0,24]
b	tentero	[0,26]
c	tentero	[0,28]

Código tres direcciones generado

(usando posiciones de memoria)

(1) $[0,30] := [0,26] + [0,28]$

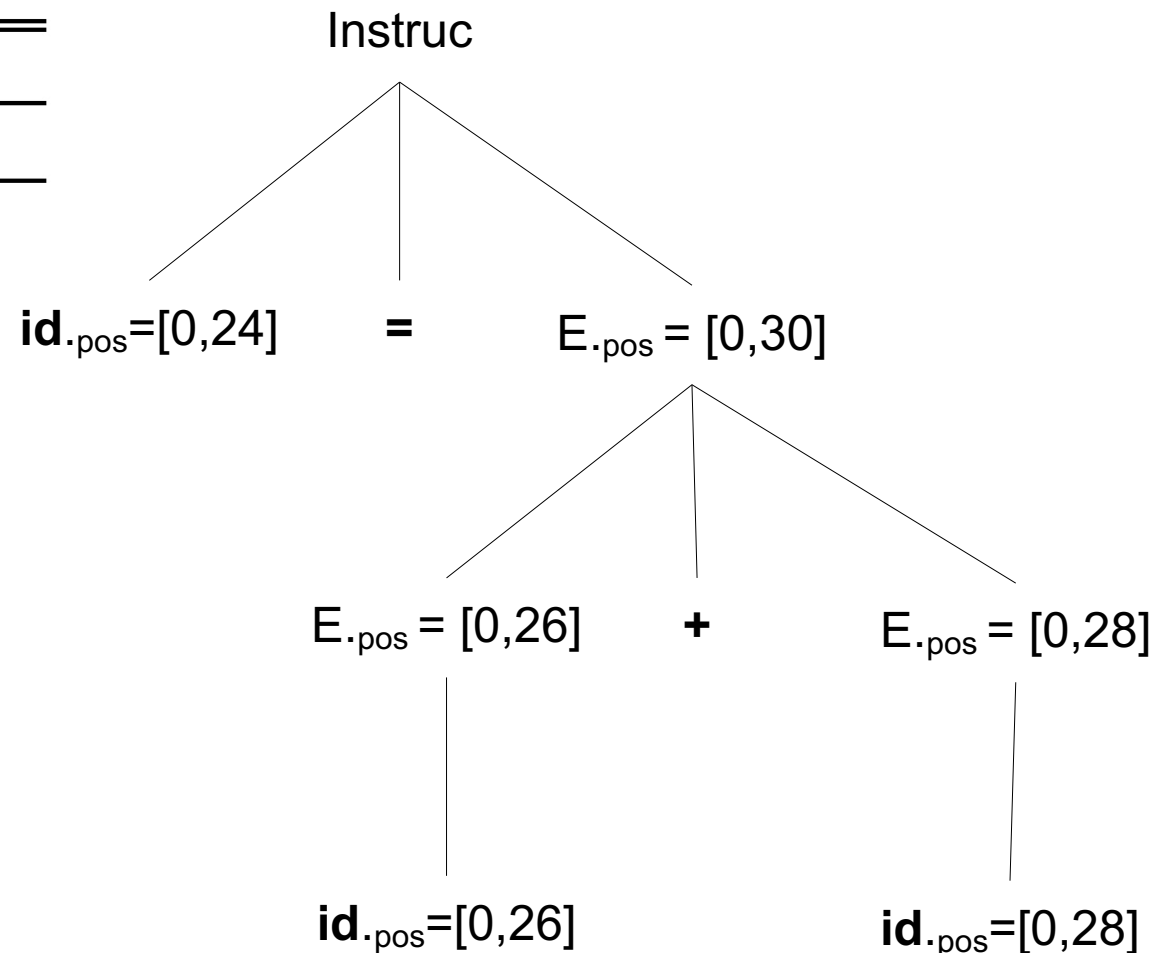
(2) $[0,24] := [0,30]$

Código tres direcciones generado

(usando nombres de posiciones de memoria)

(1) $t1 := b + c$

(2) $a := t1$



4. Estructuras y elementos de una matriz

Acceso a miembros de una estructura

```
struct ej {  
    int c1 ;  
    float c2 ;  
} a, b ;
```

TALLA_REAL = 4
TALLA_ENTERO = 2

TDS				
nom	tipo	posición	...	ref
a	testructura	0 , 124		
b	testructura	0 , 130		

Tabla de campos		
nom	tipo	posición
c1	tinteger	0
c2	treal	2



Acceso a miembros de una estructura. ETDS

$E \rightarrow id_1.id_2$	<pre>{ <u>si</u> BuscaTipo (id₁.nom) <> testructura <u>ent</u> MemError(); <u>sino</u> base := BuscaPos (id₁.nom) ; <u>si</u> no EsMiembro (id₁.nom, id₂.nom) <u>ent</u> MemError() <u>sino</u> desp_miembro:=BuscaPosMiembro(id₁.nom, id₂.nom); E.pos := base + desp_miembro ; }</pre>
$Asig \rightarrow id_1.id_2 = E$	<pre>{ <u>si</u> BuscaTipo (id₁.nom) <> testructura <u>ent</u> MemError(); <u>sino</u> base := BuscaPos (id₁.nom) ; <u>si</u> no EsMiembro (id₁.nom, id₂.nom) <u>ent</u> MemError() <u>sino</u> desp_miembro:=BuscaPosMiembro(id₁.nom, id₂.nom); Asig.pos := base + desp_miembro ; emite(Asig.pos `:=` E.pos); }</pre>

Acceso a elemento de una matriz

Declaración: `int A[n1][n2]... [nn];`

Acceso al elemento `A[i1][i2]...[in]`: $base + (((i_1 * n_2 + i_2) * n_3 + i_3 \dots) * n_n + i_n) * Talla$
 n_i es el número de elementos de la i -ésima dimensión

IS -> id	{ LI.nom := id.nom }
LI = E	{ <i>emite</i> (LI.pos := LI.pos '*' Talla (id.nom)) ; IS.pos := <i>BuscaPos</i> (id.nom) ; <i>emite</i> (IS.pos '[' LI.pos ']' := E.pos ; }
LI → [E]	{ LI.pos := <i>CrearVarTemp</i> () ; <i>emite</i> (LI.pos := E.pos); LI.ndim := 1; }
LI →	{ LI ₁ .nom := LI.nom ; }
LI ₁ [E]	{ LI.ndim := LI ₁ .ndim + 1; LI.pos := LI ₁ .pos ; <i>emite</i> (LI.pos := LI.pos '*' Num_elementos(LI.nom, LI.ndim)) ; <i>emite</i> (LI.pos := LI.pos '+' E.pos) ; }

5. Expresiones lógicas

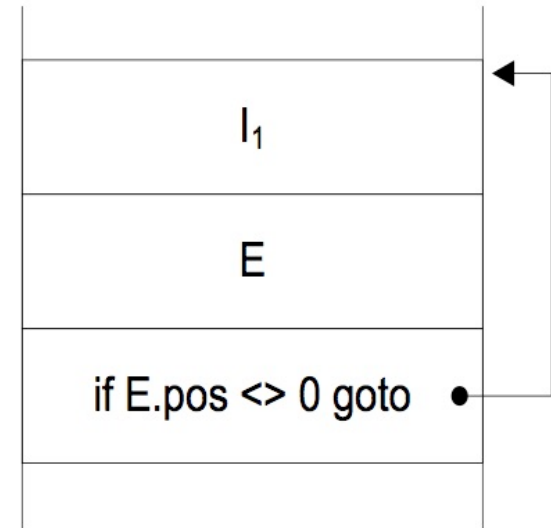
$E \rightarrow \text{true}$	$\{ E.\text{pos} := \text{CrearVarTemp}();$ $\text{emite}(E.\text{pos} \text{ ':=' } 1) \}$
$E \rightarrow \text{false}$	$\{ E.\text{pos} := \text{CrearVarTemp}();$ $\text{emite}(E.\text{pos} \text{ ':=' } 0) \}$
$E \rightarrow (E_1)$	$\{ E.\text{pos} := E_1.\text{pos} \}$
$E \rightarrow \text{id}$	$\{ E.\text{pos} := \text{BuscaPos}(\text{id}.\text{nom}) \}$

$E \rightarrow E_1 \text{ oprel } E_2$	$\{ E.\text{pos} := \text{CrearVarTemp}();$ $\text{emite}(E.\text{pos} \text{ ':=' } 1);$ $\text{emite}(\text{'if' } E_1.\text{pos oprel.op } E_2.\text{pos 'goto' SIGINST+2});$ $\text{emite}(E.\text{pos} \text{ ':=' } 0) \}$
--	---

$E \rightarrow E1 \text{ or } E2$	<pre> { E.pos:= CrearVarTemp(); emit(E.pos ':=' E1.pos '+' E2.pos) emit('if' E.pos '<= 1' goto' SIGINST+2); emit(E.pos ':=' 1') } </pre>
$ \quad E1 \text{ and } E2$	<pre> { E.pos:= CrearVarTemp(); emit(E.pos ':=' E1.pos '*' E2.pos) } </pre>
$ \quad \text{not } E1$	<pre> { E.pos:= CrearVarTemp(); emit(E.pos ':=' 0); emit('if' E1.pos '<> 0 goto' SIGINST+2); emit(E.pos ':=' 1) } </pre>

Instrucciones de control de flujo: do-while


$I \rightarrow \text{do } I_1 \text{ while } (E)$



$I \rightarrow \text{do}$	{ I.inicio := SIGINST }
I_1	
$\text{while } (E)$	{ emite('if' E.pos '<>' o goto' I.inicio) }

Instrucciones de control de flujo: while: *No funciona*

$I \rightarrow \text{while} (E) I$

$I \rightarrow \text{while}$ (E) I_1	<pre>{ I.inicio:= SIGINST } { emite('if' E.pos '=o goto' I.fin) } { emite('goto' I.inicio) ; I.fin := SIGINST }</pre> 
--	---

Esto no funciona. ¿Qué podemos hacer?

6. Relleno por retroceso

Perfil de las funciones usadas

ptro CreaLans(E)

Crea una lista que solo contiene un número de instrucción E a rellenar posteriormente. Devuelve un puntero a dicha lista.

void CompletaLans(ptro, E)

Rellena todas las instrucciones incompletas, cuyo número está contenido en la lista apuntada por ptro, con el valor de argumento E.

ptro FusionaLans (ptro, ptro)

Concatena las listas apuntadas por sus dos argumentos y devuelve un puntero a la nueva lista.

7. Instrucciones de control de flujo

while y if-else (usando relleno por retroceso)

$I \rightarrow \text{while}$ (E) I_1	$\{ I.inicio := SIGINST \}$ $\{ I.final := CreaLans(SIGINST);$ $emite('if' E.pos '=o goto' ---) \}$ $\{ emite('goto' I.inicio);$ $CompletaLans(I.final, SIGINST) \}$
--	--

$I \rightarrow \text{if } E$ $I_1 \text{ else}$ I_2	$\{ I.falso := CreaLans(SIGINST);$ $emite('if' E.pos '=o goto' ---); \}$ $\{ I.fin := CreaLans(SIGINST);$ $emite('goto' ---);$ $CompletaLans(I.falso, SIGINST) \}$ $\{ CompletaLans(I.fin, SIGINST) \}$
---	--

For (usando relleno por retroceso)

$I \rightarrow \text{for} (I_1 ;$ $E ;$	$\{ I.\text{cond} := \text{SIGINST} ; \}$ $\{ I.\text{fin} := \text{CreaLans} (\text{SIGINST}) ;$ $\text{emite} (\text{'if' } E.\text{pos} \text{'=0 goto' ---}) ;$ $I.\text{cuerpo} := \text{CreaLans} (\text{SIGINST}) ;$ $\text{emite} (\text{'goto' ---}) ;$ $I.\text{incr} := \text{SIGINST} ; \}$
$I_2)$	$\{ \text{emite} (\text{'goto' } I.\text{cond}) ;$ $\text{CompletaLans} (I.\text{cuerpo}, \text{SIGINST}) ; \}$
I_3	$\{ \text{emite} (\text{'goto' } I.\text{incr}) ;$ $\text{CompletaLans} (I.\text{fin}, \text{SIGINST}) ;$ $\}$

8. Llamadas a subprogramas

Decl_Subprg \rightarrow Tipo **id** (Param_Form)

Bloque ;

```
{ emite('push FP');  
  emite('FP := TOP');  
  area_datos := CreaLans(SIGINST);  
  emite('TOP := TOP + ---'); }  
{ CompletaLans (area_datos, DESP)  
  emite('TOP := FP');  
  emite('FP := pop');  
  emite('ret') }
```

E \rightarrow **id**

(Param_Act)

```
{ E.pos := CreaVarTemp();  
  emite('TOP := TOP +' Talla_Tipo(id.nom)) }  
{ emite('push' <estado máquina>);  
  emite('call' BuscaPos(id.nom));  
  emite('pop' <estado máquina>);  
  emite('TOP := TOP -' TallaParam(id.nom));  
  emite('pop' E.pos) }
```

Param_Act \rightarrow E

```
{ emite('push' E.pos) }
```

Param_Act \rightarrow E , Param_Act

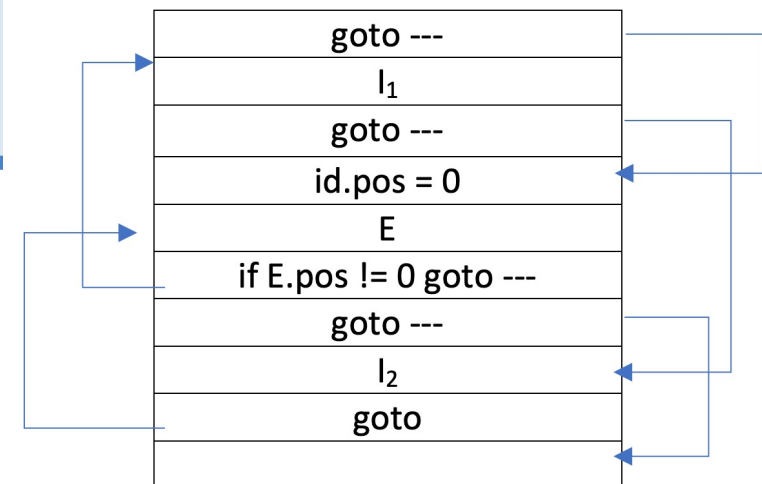
```
{ emite('push' E.pos) }
```

Escribe un ETDS que genere código intermedio para la instrucción:

$I \rightarrow \text{do } \{ I_1 \} \text{ for } (id ; E ; E_2)$

do-for es una instrucción repetitiva. Lo primero que hace la instrucción es inicializar la variable **id** a 0. Después de ello, mientras la expresión **E** sea cierta, se ejecutará el código de **I₁**, después el código de **I₂** y de nuevo se volverá a evaluar la expresión **E**. El bucle finalizará cuando **E** sea falso.

Solución Ejercicio 1



l -> do {	<code>l.inicia = CreaLans(SIGINST); emite ('goto' ---); l.cuerpo = SIGINST ;</code>
l₁ }	<code>l.incr = CreaLans(SIGINST) ; emite ('goto' ---) ; CompletaLans(l.inicia, SIGINST) ;</code>
for (id ;	<code>id.pos = BuscaPos(id.nom) ; emite(id.pos ':= 0') ; l.expr = SIGINST ;</code>
E ;	<code>emite ('if' E.pos '!= 0 goto' l.cuerpo) ; l.fin = CreaLans(SIGINST) ; emite('goto' ---); CompletaLans(l.incr, SIGINST);</code>
l₂)	<code>emite('goto', l.expr); CompletaLans(l.fin, SIGINST) ;</code>

Ejercicio 2

Dada la siguiente gramática (inspirada en la proposición “switch” del C):

$I \rightarrow \text{switch} (E) \{ L \}$

$L \rightarrow \text{case num} : P ; L$

 | **default** : P ;

 | ε

...

$P \rightarrow P ; P$

 | **break**

 | I

 | ε

...

Donde:

Tanto “E” como la constante “num” deben ser enteras.

Si la constante de un “case” coincide con el valor de la expresión E, se deberá comenzar a ejecutar todas las instrucciones asociadas a ese “case” y a los que le siguen.

La proposición “break” provoca una salida inmediata del “switch”.

Las instrucciones asociadas al “default” se ejecutarán siempre (excepto si se encontró antes una instrucción “break”).

Construir un ETDS que realice las acciones semánticas necesarias para la generación de código intermedio de tres direcciones.

Solución ejercicio 2

$I \rightarrow \text{switch} (E)$ $\{ L \}$	$\{ L.pos:=E.pos; L.anterior := nil; L.fin:=nil \}$
$L \rightarrow \text{case num} :$ $P ;$ L_1	$\{ I.sigcond:=CreaLans(SIGINST) ;$ $\text{emite('if' num.val '<>' L.pos 'goto' --)};$ $\text{CompletaLans}(L.anterior, SIGINST) \}$ $\{ L_1.anterior:=CreaLans(SIGINST) ; \text{emite('goto' --)};$ $L_1.pos:=L.pos; \text{CompletaLans}(I.sigcond, SIGINST);$ $L_1.fin:=FusionaLans(P.fin, L.fin) \}$
$L \rightarrow \text{default} :$ $P ;$	$\{ \text{CompletaLans}(L.anterior, SIGINST) \}$ $\{ \text{CompletaLans}(L.fin, SIGINST);$ $\text{CompletaLans}(P.fin, SIGINST) \}$
$L \rightarrow \varepsilon$	$\{ \text{CompletaLans}(L.anterior, SIGINST)$ $\text{CompletaLans}(L.fin, SIGINST); \}$
$P \rightarrow P_1 ; P_2$	$\{ P.fin:=FusionaLans(P_1.fin, P_2.fin) \}$
$P \rightarrow \text{break}$	$\{ P.fin:=CreaLans(SIGINST); \text{emite('goto' --)} \}$
$P \rightarrow I$	$\{ P.fin:=I.fin \}$
$P \rightarrow \varepsilon$	$\{ P.fin:=nil \}$

