



SEMANTICA OPERACIONAL

Repaso Clase Anterior

REPASO CLASE ANTERIOR

UNIDADES DE PROGRAMA

- **Atributos:**
 - **Nombre**
 - Declaración/invocación
 - **Alcance**
 - Dónde se conoce el identificador nombre Rutina
 - Ambiente de referencia de la Rutina: global, local, no local
 - **Tipo**
 - Dado por los parámetros y valor de retorno. **Signatura**
 - **L-Valor:**
 - Relacionado a dónde se almacena las sentencias en la memoria
 - **R-Valor:**
 - Relacionado al momento en que se hace la referencia a esa Rutina
- Diferencias entre **Declaración** y **Definición**
 - Extensión del alcance
- **Comunicación** entre Rutinas
 - Pasaje parámetros y entorno global

REPASO CLASE ANTERIOR

ESQUEMAS DE EJECUCIÓN

- **Estático**
- **Basado en pila**
- **Dinámico**

REPASO CLASE ANTERIOR

PROCESADOR ABSTRACTO - MEMORIA

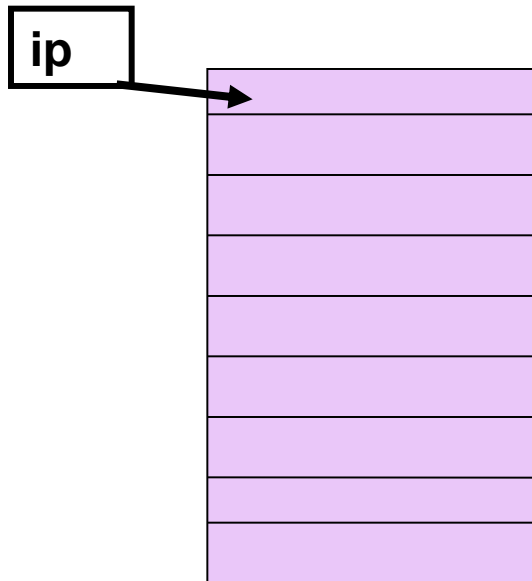
- **Sentencias:**

- **Set**

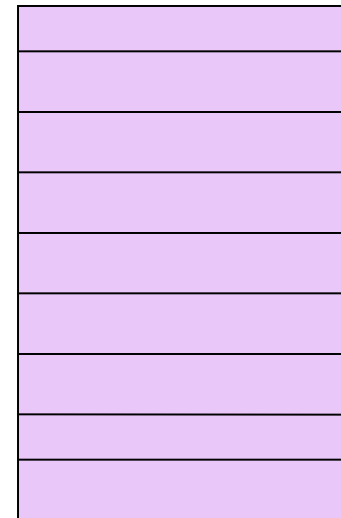
- Para asignar valores. Ej: **set** 10,D[20]; **set** 20, **read**; **set** **write**,D[50]

- **Jump y jump**

- Bifurcación incondicional y condicional. Ej: **jump** 47, **jump** 47,D[13]>D[8]



C: Zona de código



D: Zona de datos

REPASO CLASE ANTERIOR

ESQUEMAS DE EJECUCIÓN - C1 Y C2

- Esquemas de ejecución
 - Estático, basado en pila y dinámico
- Estático:
 - C1:
 - Programa sencillo sin bloques internos, solo código programa principal
 - En zona de datos: SOLO datos locales
 - C2:
 - Programa con bloques internos SIN anidamientos
 - Compilación junta y separada (C2')
 - En zona de datos: **datos locales + pto. de retorno**

ESQUEMAS DE EJECUCIÓN

CASOS DE C3 A C6

- **Estático**
- **Basado en pila**
- **Dinámico**

C3

C2 + RECURSIÓN Y VALOR DE RETORNO

- Esquema basado en pila
- Rutinas con capacidad de llamarse a si mismas (recursión directa) o de llamar a otra rutina en forma recursiva (recursión indirecta).
- Rutinas con la capacidad de devolver valores, es decir, funciones.

C3: FUNCIONAMIENTO

- El registro de activación de cada unidad será de tamaño fijo y conocido, pero no se sabrá cuantas instancias de cada unidad se necesitarán durante la ejecución.
- Igual que en C2 el compilador puede ligar cada variable con su desplazamiento dentro del correspondiente registro de activación. El desplazamiento es estático
- La dirección donde se cargará el registro de activación, es dinámica, por lo tanto, la ligadura con la direcciones absolutas en la zona de Datos de la memoria, solo puede hacerse en ejecución.
- Cada nueva invocación aloca un nuevo registro de activación y se establecen las nuevas ligaduras ente el segmento de código y el nuevo registro de activación.

C3: EJEMPLO DEL FACTORIAL

```
int n;
```

```
int fact()
```

```
{
```

```
int loc;
```

```
if (n>1) {
```

```
    loc= n--;
```

```
    return loc * fact(); →
```

```
}
```

```
else return 1;
```

```
}
```

```
main ()
```

```
{
```

```
get (n);
```

```
if (n>0) print (fact());
```

```
else print( "input error");
```

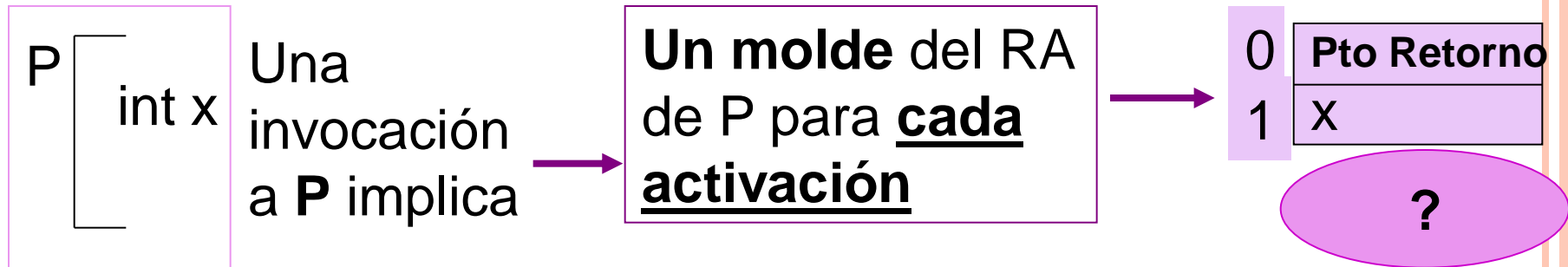
```
}
```

Recursión

+

**Valor de
retorno**

C3: FUNCIONAMIENTO



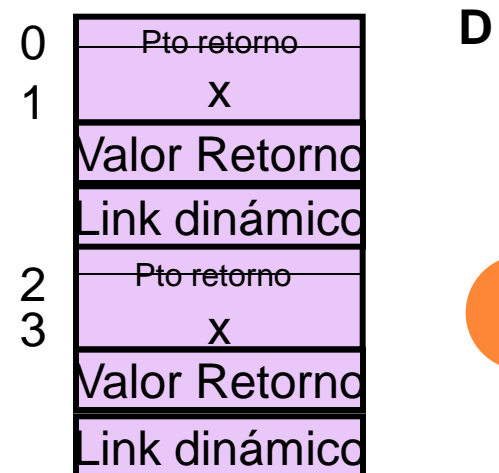
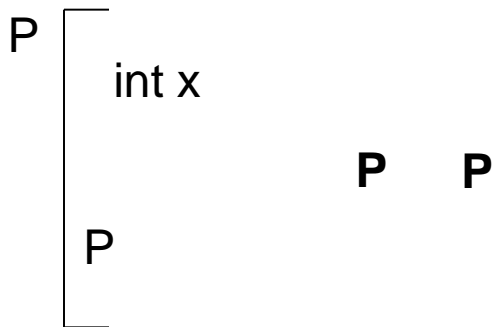
En el caso C2:

D(P,0)	→	pto de retorno
D(P,1)	→	x

¿Qué información necesitaríamos adicionar en el RA en este caso?

C3: FUNCIONAMIENTO

- Ahora hay que tener en cuenta que:
 - Cuando la instancia actual de la unidad termine de ejecutarse, su registro de activación no se necesitará mas, por lo tanto se puede liberar el espacio ocupado por su registro de activación y dejar el espacio disponible para nuevos registros de activación y que..
 - Las unidades pueden devolver valores (funciones) y esos valores NO deberían perderse cuando se desactive la unidad. Entonces....



C3: DATOS NECESARIOS

Para manejar la alocación dinámica necesitamos nuevos elementos:

Como vimos se necesita el:

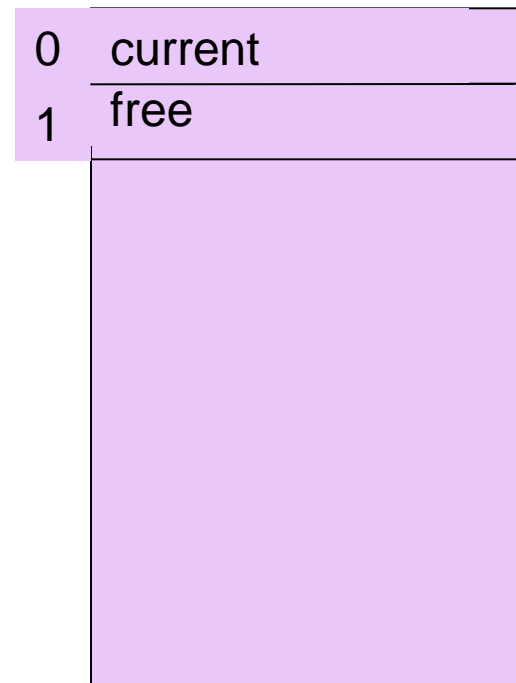
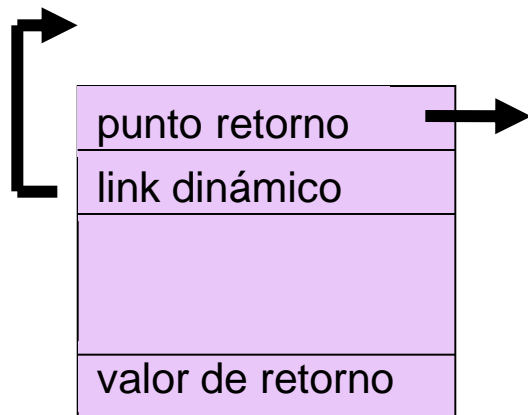
- Valor de retorno: Al terminar una rutina se desaloca su RA, por lo tanto la rutina llamante debe guardar en su RA el valor de retorno de la rutina llamada.
- Link dinámico: Contiene un puntero a la dirección base del registro de activación de la rutina llamadora

Pero además se necesita:

- Current: Dirección base del registro de activación de la unidad que se este ejecutando actualmente
- Free: Próxima dirección libre en la pila

Cadena dinámica: cadena de links dinámicos originada en la secuencia de registros de activación activos. Representa la secuencia dinámica de unidades activadas

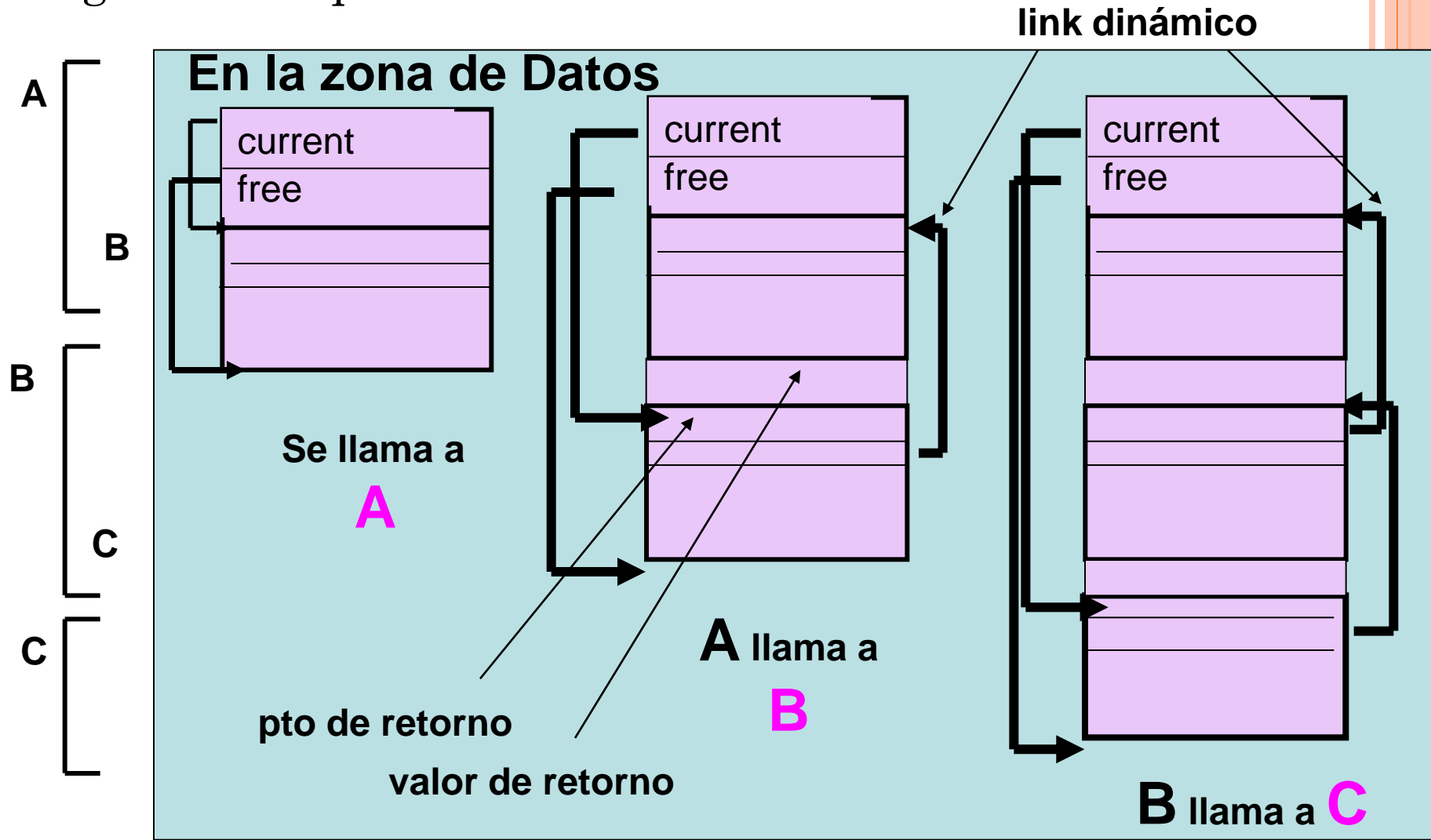
C3: MOLDES



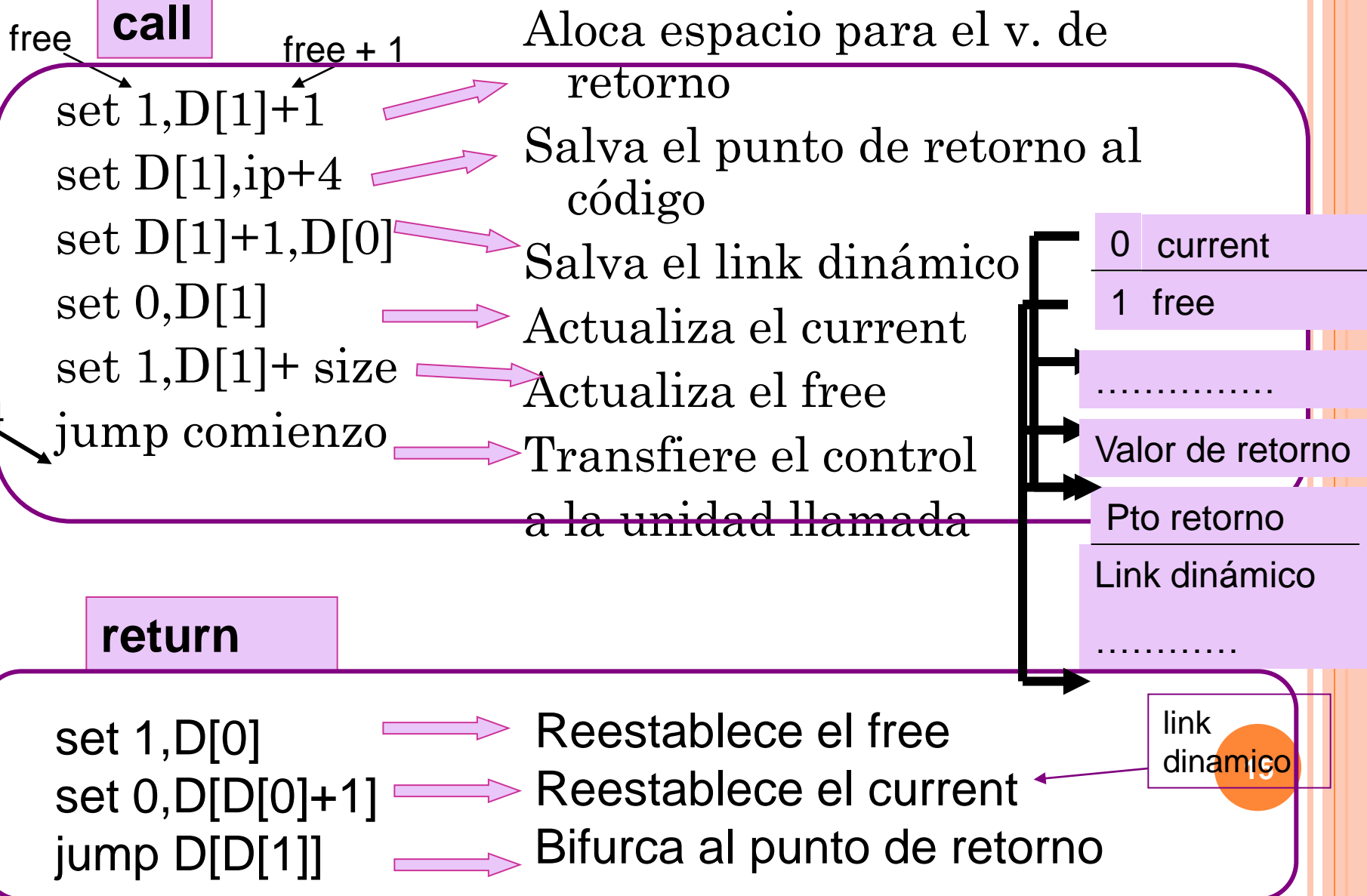
D

C3: EJEMPLO

Supongamos la siguiente estructura de programa y el siguiente esquema de llamado..



C3: ¿QUÉ SUCEDE CUANDO SE LLAMA O RETORNA DE UNA RUTINA?



C3: SEMÁNTICA DEL CALL - RETURN

Cuando se llama...

set 1,D[1]+1
set D[1],ip+4
set D[1]+1,D[0]
set 0,D[1]
set 1,D[1]+ size
jump comienzo

Cuando se retorna...

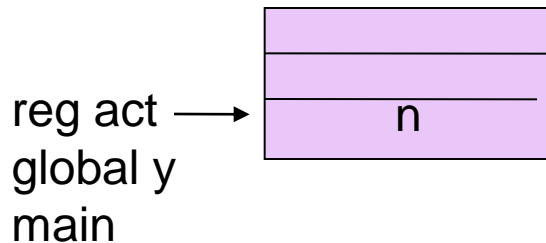
set 1,D[0]
set 0,D[D[0]+1]
jump D[D[1]]

```
int f1()  
{  
    int a;  
    ...  
}  
void main ()  
{  
    int b;  
    ...  
    f1();  
    ..  
}
```

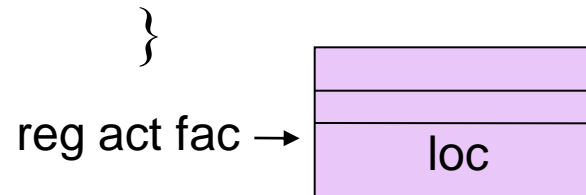

C3: EJEMPLO DEL FACTORIAL

Basándonos en el primer ejemplo: llamada recursiva a la función factorial tenemos:

```
int n;  
int fact()  
{  
    int loc;  
    if (n>1) {  
        loc = n--;  
        return loc*fact();  
    }  
    else  
        return 1;  
}
```



```
main()  
{  
    get(n);  
    if (n>=0)  
        print(fact());  
    else  
        print("error  
entrada");  
}
```



C3: EJEMPLO DEL EFECTO EN MEMORIA DE LA EJECUCIÓN DE FACTORIAL

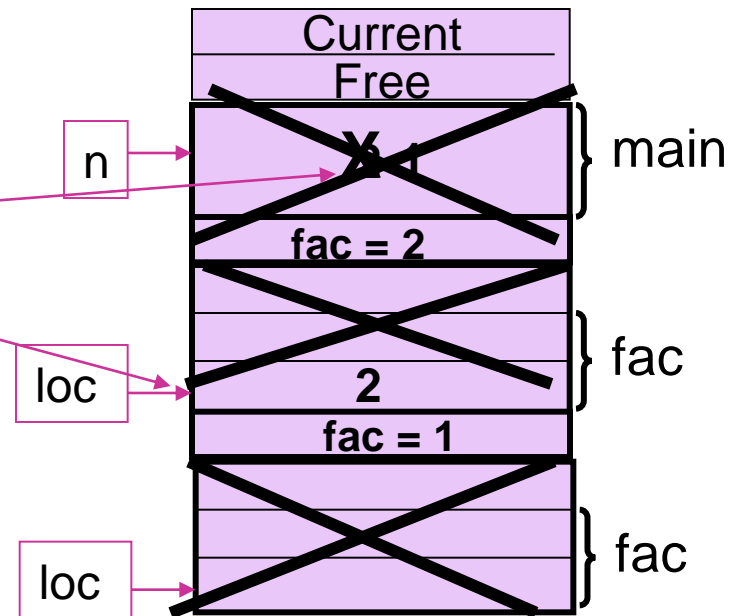
main

fac

fac

Invocación a fac con n=2

```
int n;  
int fact()  
{  
    int loc;  
    if (n>1) {  
        loc = n--;  
        return loc*fact();  
    }  
    else  
        return 1;  
}  
main()  
{  
    get(n);  
    if (n>=0)  
        print(fact());  
    else  
        print("error  
    entrada");  
}
```



imprime 2

C4: ESTRUCTURA DE BLOQUE

C4'

permite que dentro de las sentencias compuestas aparezcan declaraciones locales

C4''

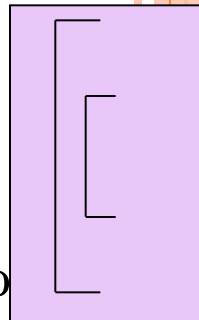
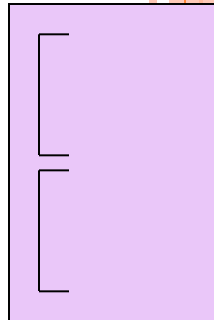
permite la definición de una rutina dentro de otras rutinas.
(anidamiento de rutinas)

Estas características conforman el concepto de **estructura de bloque**:

- controla el alcance de las variables,
- define el tiempo de vida de las variables
- divide el programa en unidades mas pequeñas.

Los bloques pueden ser:

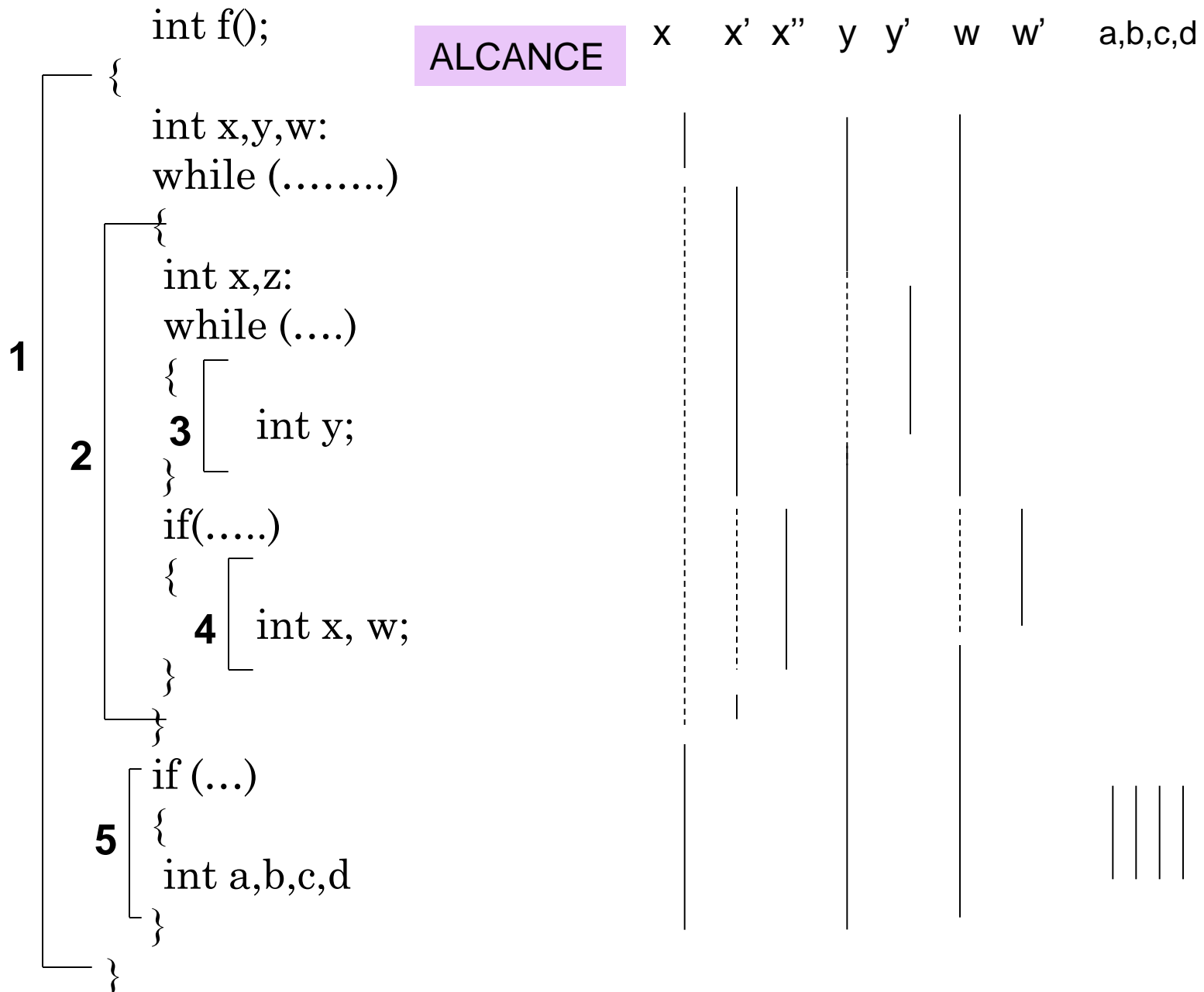
- disjuntos (no tiene porción común)
- anidados (un bloque esta completamente contenido en otro)



C4': ANIDAMIENTO VÍA SENTENCIAS COMPUESTAS

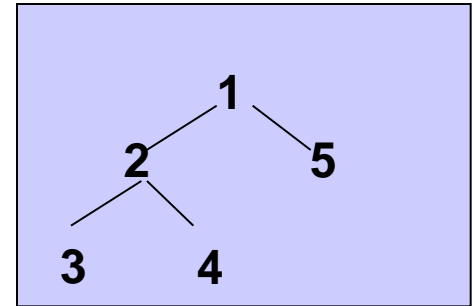
- Un bloque tiene forma de una sentencia compuesta:
 $\{ \langle \text{lista de declaraciones} \rangle ; \langle \text{lista de sentencias} \rangle \}$
- Las variables tienen alcance local: son visibles dentro de la sentencia compuesta, incluyendo cualquier sentencia compuesta anidada en ella
- Si en el anidamiento, hay una nueva declaración de un nombre, la declaración interna enmascara la externa del mismo nombre.

C4': SENTENCIAS COMPUESTAS



C4': SENTENCIAS COMPUESTAS

Alocación: implementación



estático

dinámico

incluir las necesidades dentro del registro de activación de la unidad a la que pertenece

alocar el espacio dinámicamente cuando se ejecutan las sentencias

punto de retorno		
link dinámico		
x de 1		
y de 1		
w de 1		
x' de 2		a de 5
z de 2		b de 5
y' de 3	x'' de 4	c de 5
w' de 4		d de 5

1

2

3

4

5

```
int f();
{ int x,y,w;
  while (.....)
  {   int x,z;
      while (....)
      { int y;
        }
      if(.....)
      {int x, w;
        }
      }
  }
  if (...)
  {int a,b,c,d
```

simple y eficiente en tiempo

eficiente en espacio

C4”: RUTINAS ANIDADAS

```
//file
int x,y,z;
f1()
{
    int t,u;
    f2()
    {
        int x,w;
        f3()
        {
            int y,w,t;
        }
        x = y + t + w + z;
    }
}
```

x = y + t + w + z

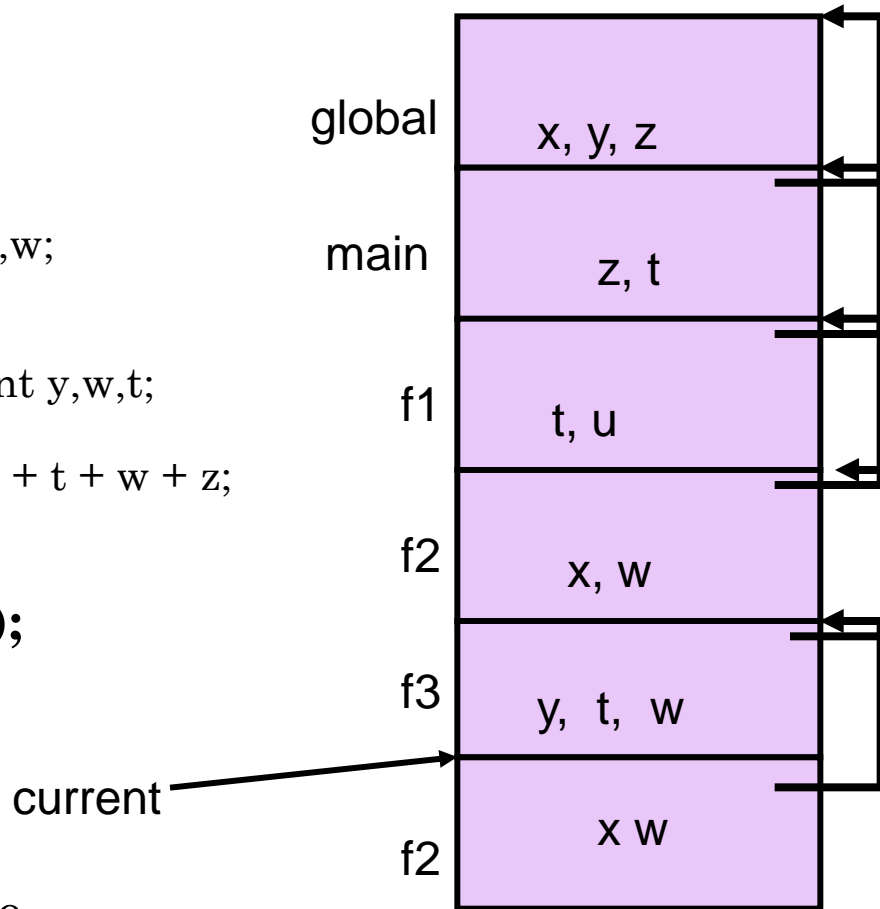
local global local global

```
main ();
{
    int z,t;
}
//end file
```

C4'': RUTINAS ANIDADAS

ACCESO AL AMBIENTE NO-LOCAL

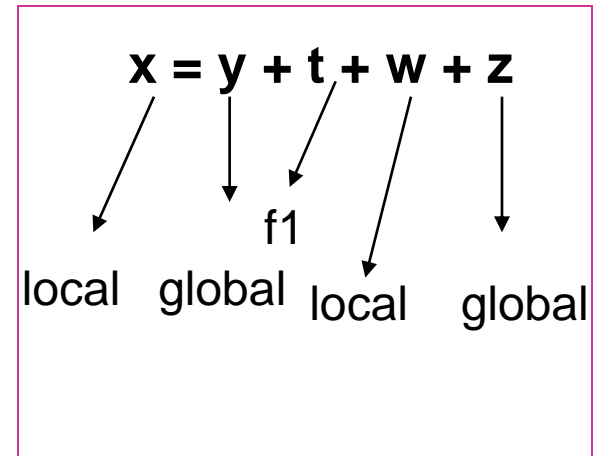
```
//file
int x,y,z;
f1()
{
  int t,u;
  f2()
  {
    int x,w;
    f3()
    {
      int y,w,t;
    }
    x = y + t + w + z;
  }
}
main ()
{
  int z,t;
}
//end file
```



Secuencia de llamadas:

main f1 f2 f3 f2

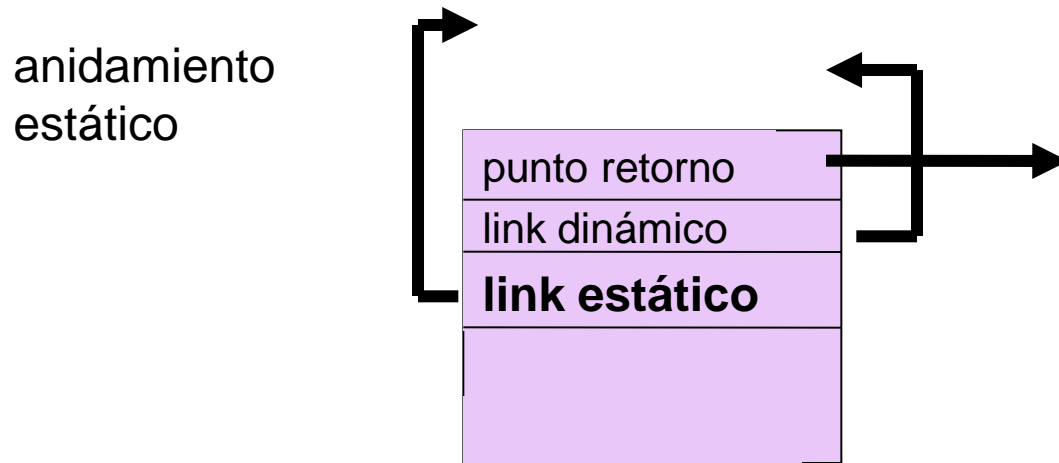
cadena dinámica



La cadena dinámica no sirve para localizar esas referencias no locales, en lenguajes que siguen la cadena estática

C4'': ACCESO AL AMBIENTE NO LOCAL

- Link estático: apunta al registro de activación de la unidad que estáticamente la contiene



- La secuencia de links estáticos se denomina cadena estática

C4: ¿QUÉ SUCEDE CUANDO SE LLAMA O RETORNA DE UNA RUTINA?

call

free → free + 1

set 1, D[1] + 1

set D[1], ip - 5

set D[1] + 1, D[0]

set 0, D[1]

set 1, D[1] + size

ip + 5 → jump comienzo

aloca espacio para el v. de retorno
salva el punto de retorno
salva el link dinámico

Con el link dinámico se agrega una sentencia:

Colocar en D[1] + 2 el link estático

llamada

set 1, D[0]

set 0, D[D[0] + 1]

jump D[D[1]]

reestablece el free

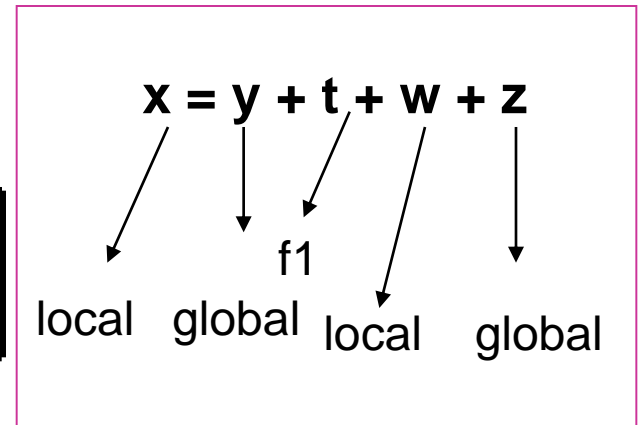
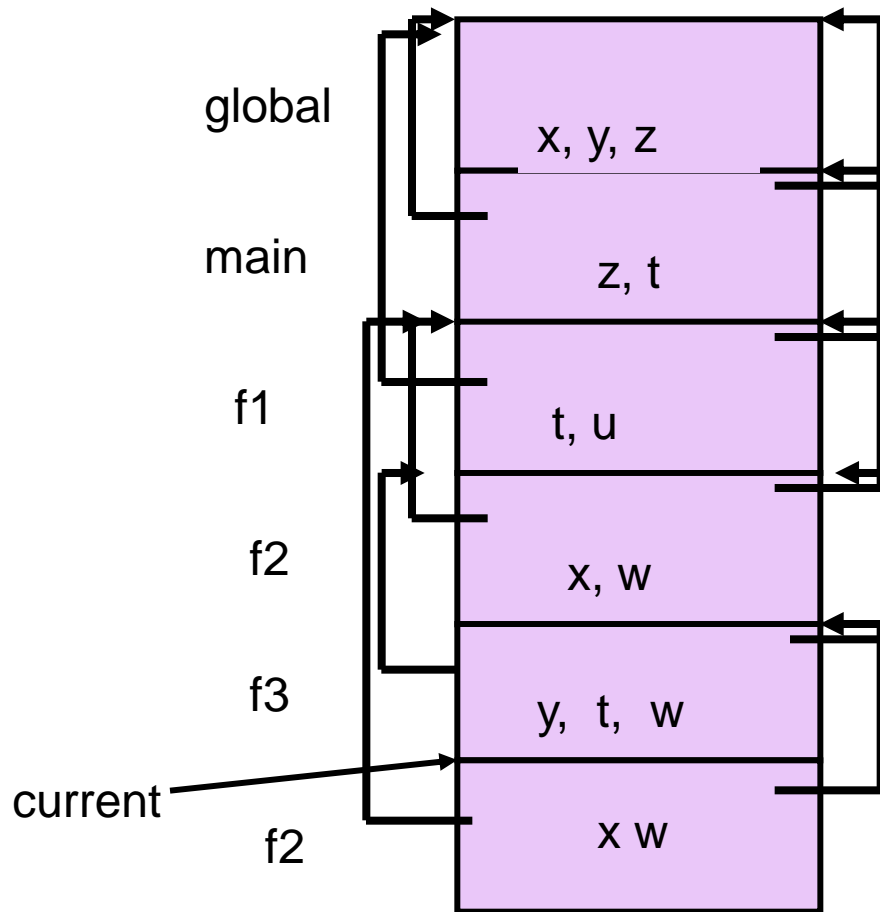
reestablece el current

bifurca al punto de retorno

link
dinamico

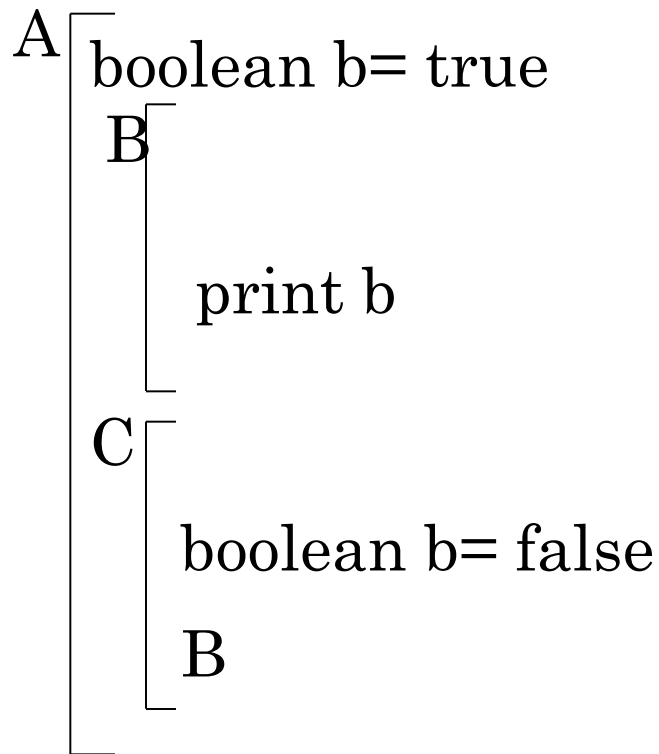
return

C4'': CADENA ESTÁTICA



Con la cadena estática podemos localizar esas referencias no locales, en lenguajes que siguen la cadena estática

CADENA ESTÁTICA \neq CADENA DINÁMICA



- A C B
 - estático
 - b es la de A \rightarrow true
 - dinámico
 - b es la de C \rightarrow false

C5: DATOS MÁS DINÁMICOS

C5'

Registro de
activación cuyo
tamaño se conoce
cuando se activa la
unidad.

Datos
semidinámicos

C5''

Los datos pueden
alocarse durante
la ejecución.

Datos
dinámicos

C5': DATOS SEMIDINÁMICOS

Variables cuyo tamaño se conoce en compilación

- **Arreglos dinámicos**

type VECTOR is array (INTEGER range <>);

define un arreglo con índice irrestricto

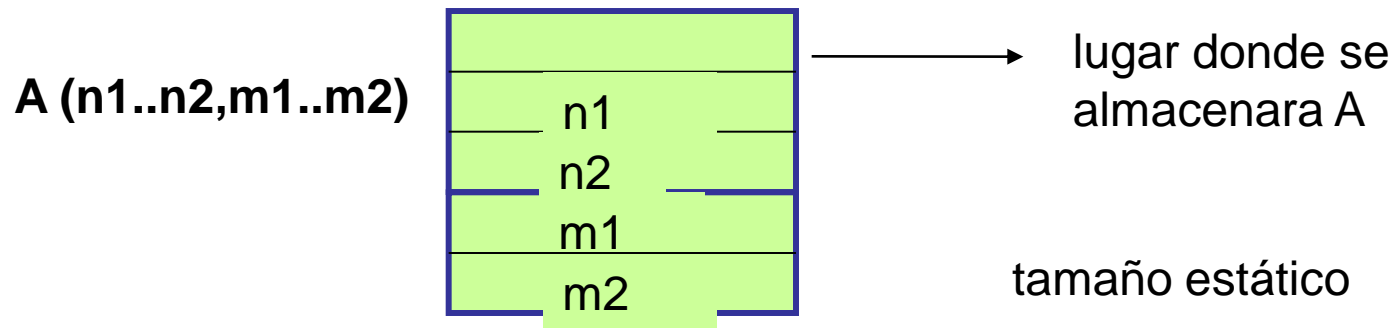
A: VECTOR (0..N);

B: VECTOR(1..M);

N y M deben ligarse a algún valor entero para que A y B puedan alocarse en ejecución
(referencia al ambiente no local o parámetros)

C5': IMPLEMENTACIÓN DE ARREGLOS DINÁMICOS

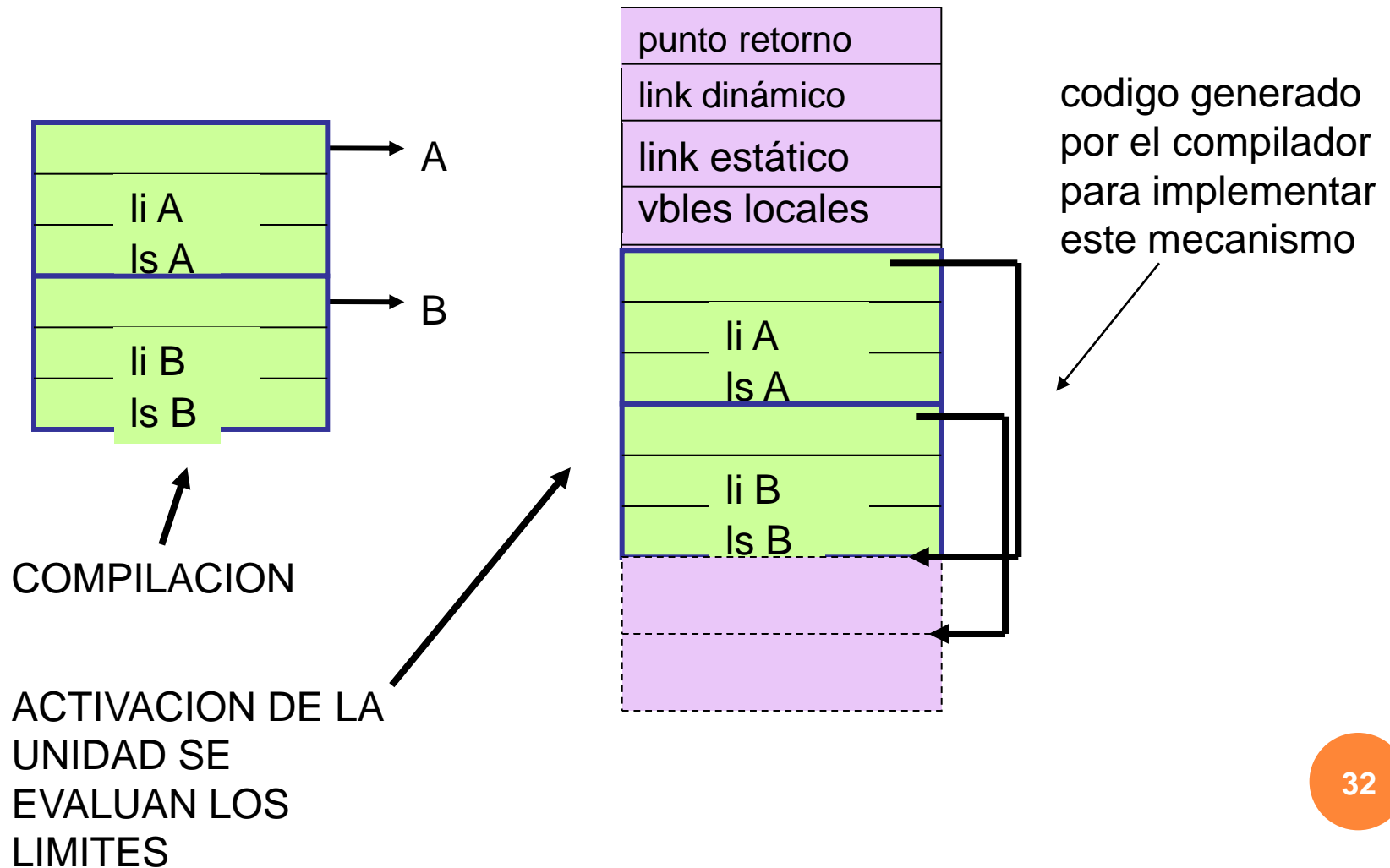
- **COMPILACION:** se reserva lugar en el registro de activación para los **descriptores** de los arreglos dinámicos.



Todos los accesos al arreglo dinámico son traducidos como referencias indirectas a través del puntero en el descriptor, cuyo desplazamiento se determina estáticamente.

A: VECTOR (0..N);
B: VECTOR(1..M);

EJECUCION



C5': DATOS SEMIDINÁMICOS

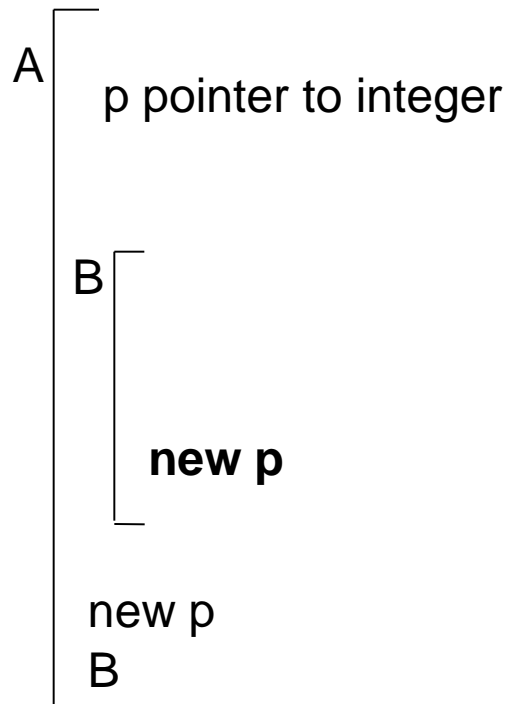
EJECUCION: el registro de activación se aloca en varios pasos:

1. Se aloca el almacenamiento para los datos de tamaño conocido estáticamente y para los descriptores de los arreglos dinámicos.
2. Con la declaración se calculan las dimensiones en los descriptores y se extiende el registro de activación para incluir el espacio para la variable dinámica.
3. Se fija el puntero del descriptor con la dirección del área alocada

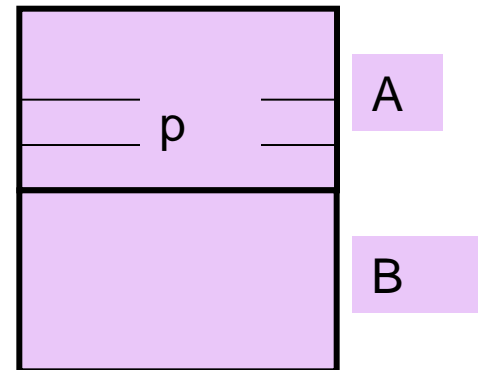
C5'': DATOS DINÁMICOS

- Se alocan explícitamente durante la ejecución mediante instrucciones de
alocación

donde aloca el objeto
apuntado?



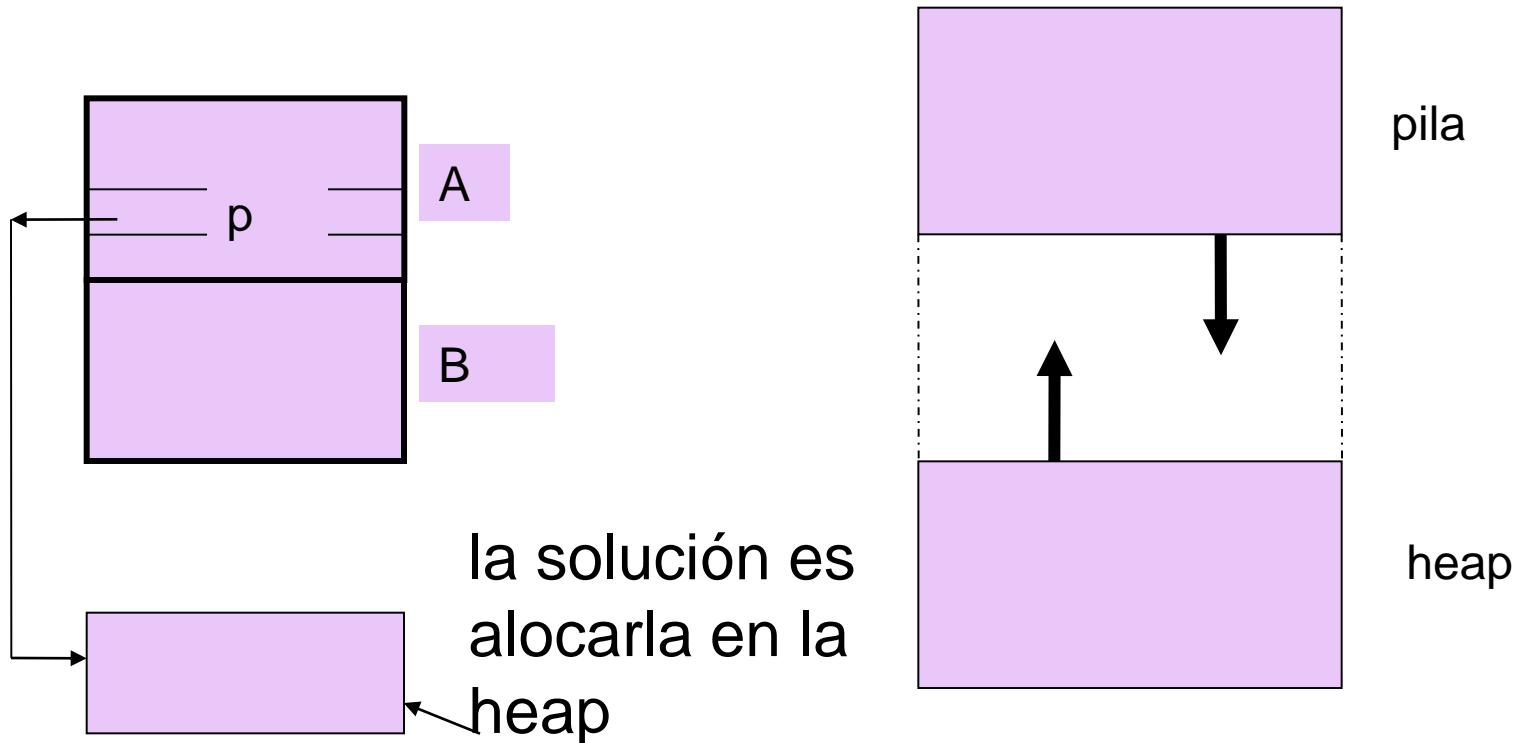
A B



el reg de Act de A no
esta en el tope

el reg de Act de B no permite
que luego se use en A

EL TIEMPO DE VIDA NO DEPENDE DE LA SENTENCIA
DE ALOCACIÓN, VIVIRÁ MIENTRAS ESTE APUNTADA



C6: LENGUAJES DINÁMICOS

- Se trata de aquellos lenguajes que adoptan más reglas dinámicas que estáticas.
- Usan **tipado** dinámico y reglas de **alcance** dinámicas.
- Se podrían tener reglas de tipado dinámicas y de alcance estático, pero en la practica las propiedades dinámicas se adoptan juntas.
- Una propiedad dinámica significa que las ligaduras correspondientes se llevan a cabo en ejecución y no en compilación.

- Variables estáticas C1-C2

estático

- Variables semiestáticas o automáticas C3-C4
- Variables semidinámicas C5'

pila

- Variables dinámicas C5''
- Tipos y alcance dinámico C6

heap

```
1 Program ejecucionRutina(output);
2   var
3       x , y , z: integer;
4   procedure Uno();
5       var
6           y :integer;
7   procedure Dos();
8       var
9           m:integer;
10      begin
11          m:= x+y;
12          writeln(m);
13      end;
14      begin
15          y :=4;
16          Dos();
17      end;
18  procedure Tres();
19      var
20          x:integer;
21      begin
22          x:=10;
23          Uno();
24      end;
25  begin
26      x:=1;      y:=2;      z:=3;
27      writeln('estamos probando ejecución de rutinas!');
28      Tres();
29      writeln(x,y,z)
30  end.
```