Conceptos y Paradigmas de Programación

SEMANTICA OPERACIONAL

Repaso Clase Anterior

Repaso Clase Anterior
Unidades de programa

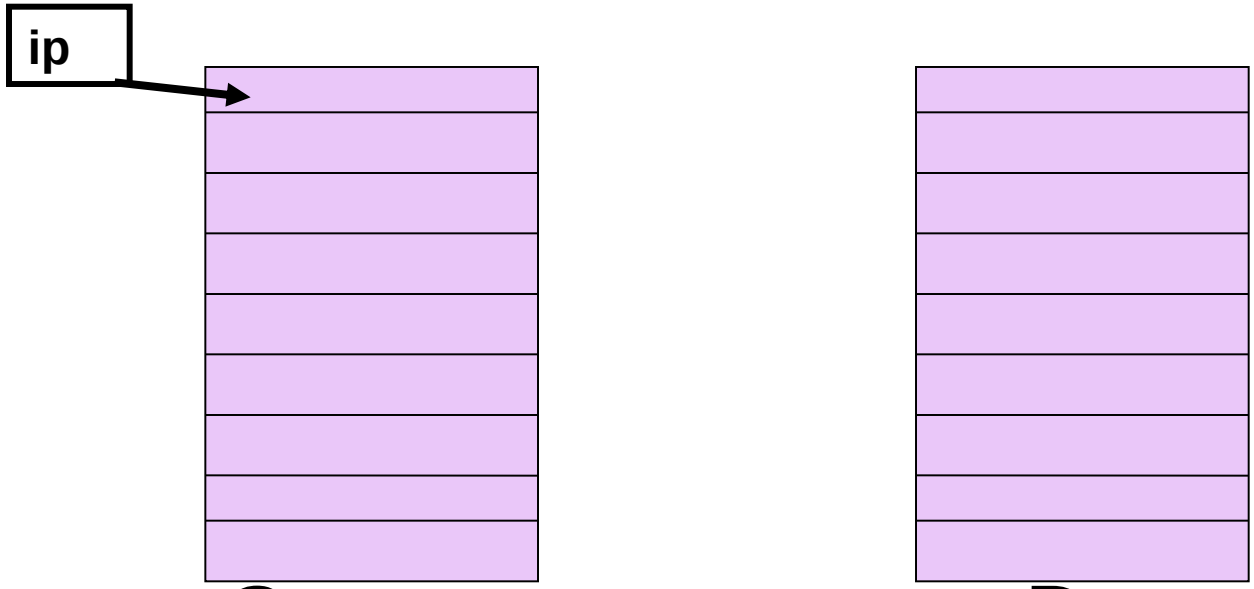
- **Atributos:**
  - **Nombre**
    - Declaración/invocación
  - **Alcance**
    - Dónde se conoce el identificador nombre Rutina
    - Ambiente de referencia de la Rutina: global, local, no local
  - **Tipo**
    - Dado por los parámetros y valor de retorno. **Signatura**
  - **L-Valor:**
    - Relacionado 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 jumpt**
    - Bifurcación incondicional y condicional. Ej: **jump** 47**, jumpt** 47,D[13]>D[8]



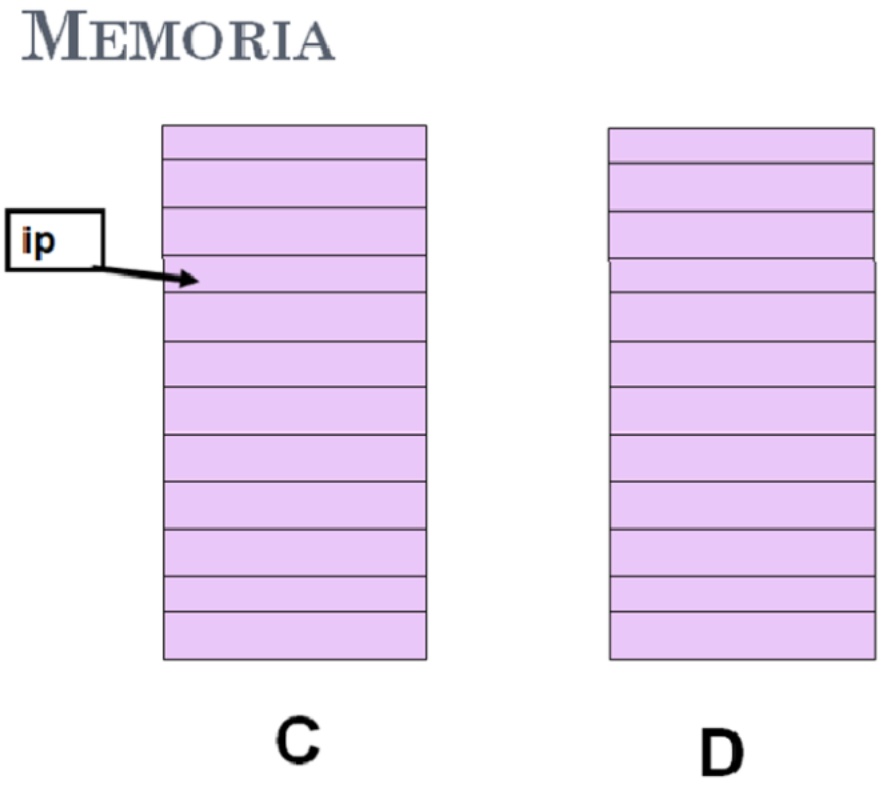
C: Zona de código **D**: Zona de datos

Repaso Clase Anterior
Esquemas de ejecución - C1 y C2

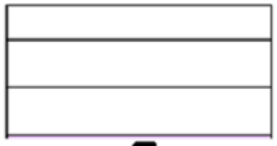
- Esquemas de ejecución
  - Estático, basado en pila y dinámico
- **Estático**:
  - **C1**:
    - Programa sencillo sin bloques internos, solo código programa principal
    - En zona de datos: SOLO **datos locales**
  - **C2**:
    - Programa con bloques internos SIN anidamientos
    - Compilación junta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

Repaso Clase Anterior
Esquemas de ejecución - C1 y C2

- Resumen esquema C1 yC2:



Por cada unidad de programa



COMO MÍNIMO ESE **RA** DEBE CONTENER:
-DATOS LOCALES Y
-**PUNTO DE RETORNO** AL CÓDIGO

- DATOS NO LOCALES: AL TENER TODOS LOS DATOS CARGADOS EN LA MEMORIA, SE RESUELVEN TODAS LAS REFERENCIAS A ESOS DATOS

TIPO de variables según su TIEMPO DE VIDA: Solo ESTÁTICAS

Esquemas de ejecución
Casos de C3 a C6

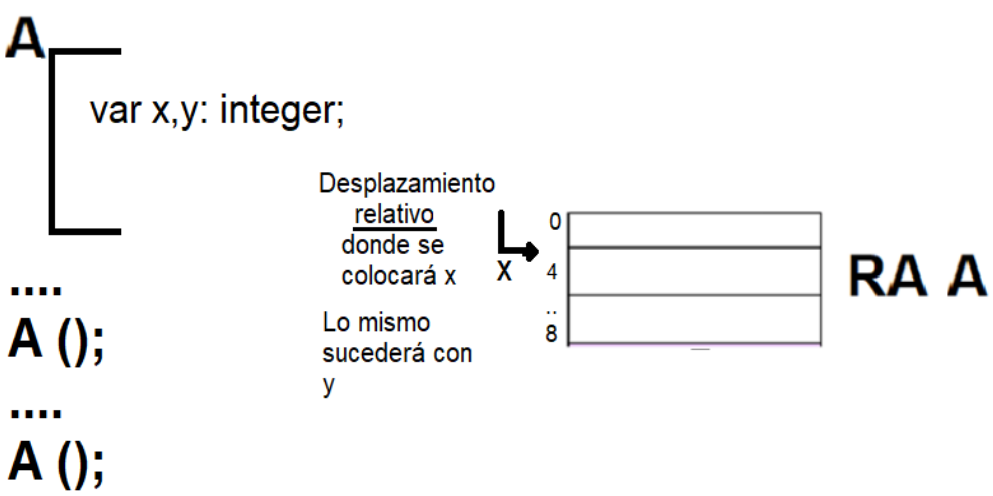
- **Estático**
- **Basado en pila**
- **Dinámico**

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**,



EN EJECUCIÓN...

CUANDO SE INVOCA POR PRIMERA VEZ A LA RUTINA A:

SI EL **RA A** SE CARGA EN LA ZONA DE MEMORIA **1000** ENTONCES LA **VARIABLE x** TOMARÁ LA **DIRECCIÓN ABSOLUTA 1004**

CUANDO SE INVOCA POR SEGUNDA VEZ A LA RUTINA A

SI EL **RA A** SE CARGA EN LA ZONA DE MEMORIA 2050, ENTONCES LA **VARIABLE x** TOMARÁ LA **DIRECCIÓN ABSOLUTA 2054**

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

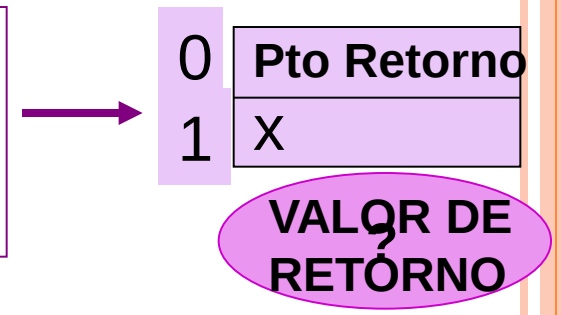
+

Valor de retorno

C3: Funcionamiento

P [int x]

Una invocación a **P** implica → **Un molde** del RA de P para **cada activación**



En el caso C2:

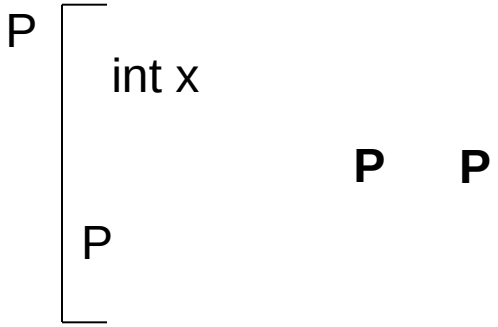
D(P,0) → **pto de retorno**

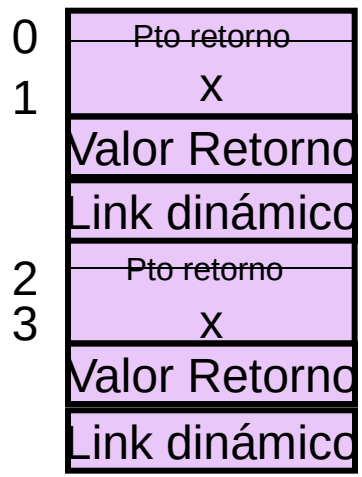
D(P,1) → **x**

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

C3: FUNCIONAMIENTO

- Ahora hay que tener en cuenta que:
  - Las unidades pueden devolver **valores** (funciones) y esos valores **NO** deberían perderse cuando se desactive la unidad... Pero también 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 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 lllamante debe guardar en su RA sel valor de retorno de la rutina llamada.
- **Link dinámico:** Contiene un puntero a la dirección base del registro de activación de la rutina llamadora

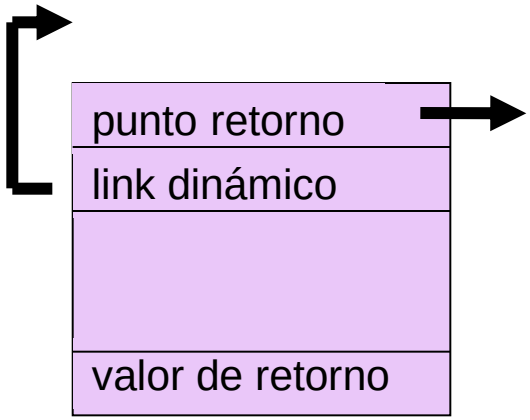
Pero además …. se necesita:

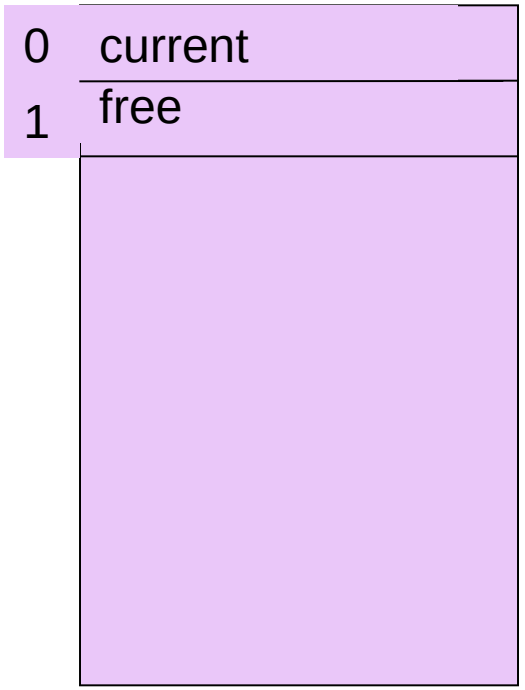
- **Current:** Dirección base del registro de activación de la unidad que se este ejecutando actualmente
- **Free:** Próxima dirección libre en la pila

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

C3: Moldes

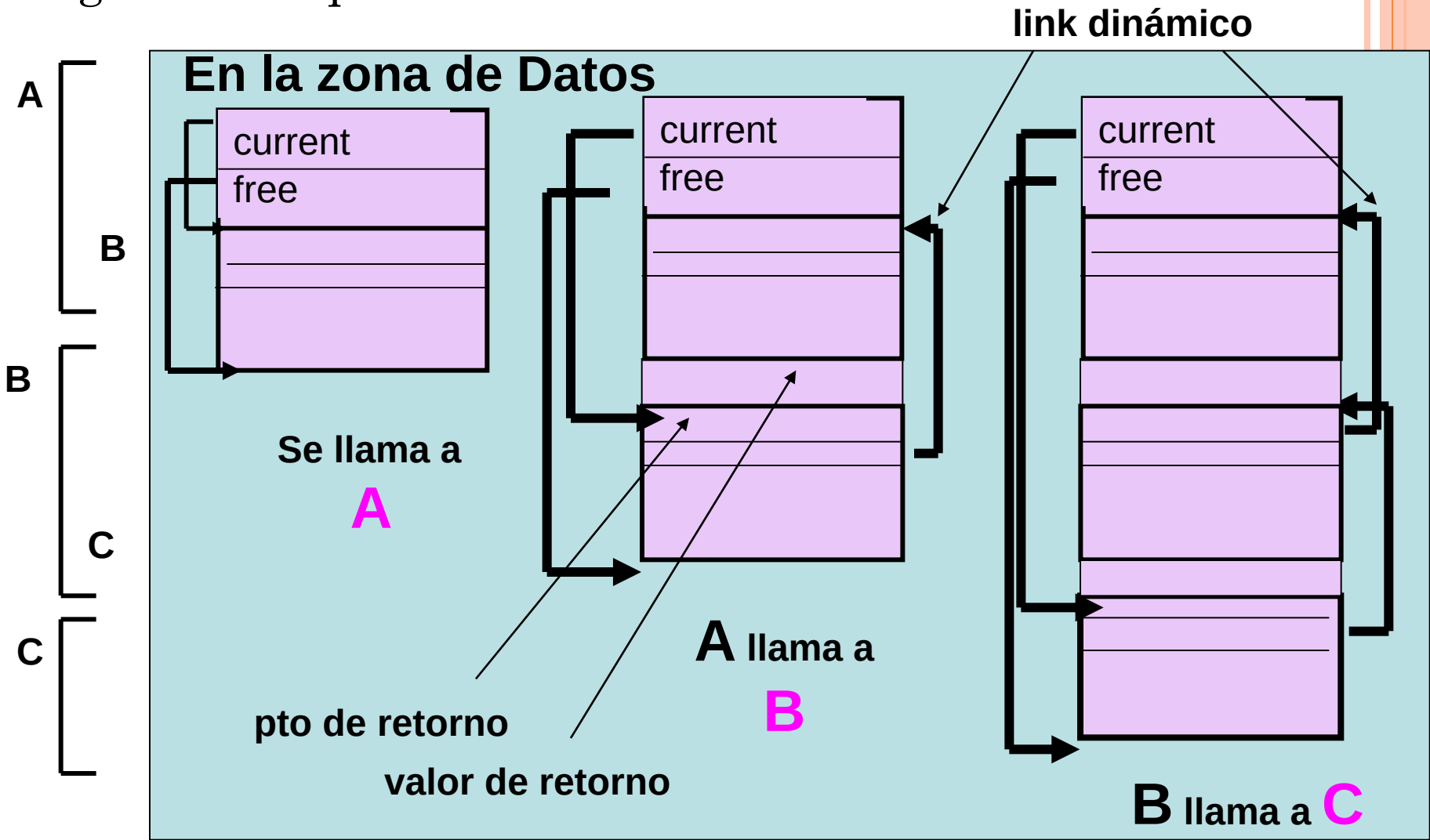
D


punto retorno
link dinámico
valor de retorno


0
current
1
free

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?

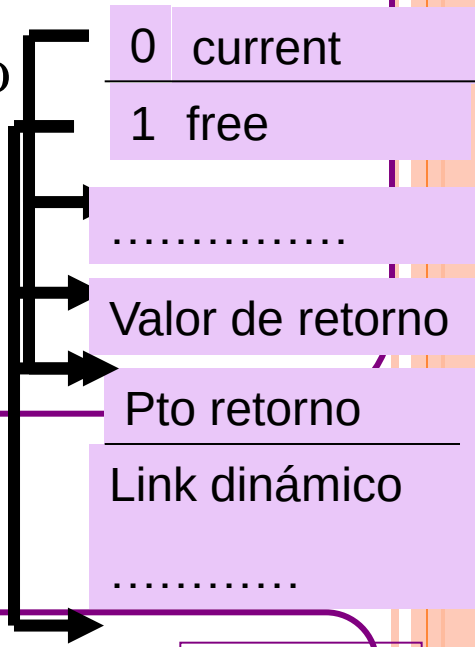
call

free

free + 1

set 1,D[1]+1	Aloca espacio para el v. de retorno
set D[1],ip+4	Salva el punto de retorno al código
set D[1]+1,D[0]	Salva el link dinámico
set 0,D[1]	Actualiza el current
set 1,D[1]+ size	Actualiza el free
jump comienzo	Transfiere el control a la unidad llamada

ip+4



return

set 1,D[0]	Reestablece el free
set 0,D[D[0]+1]	Reestablece el current
jump D[D[1]]	Bifurca al punto de retorno

link dinamico

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]]
```

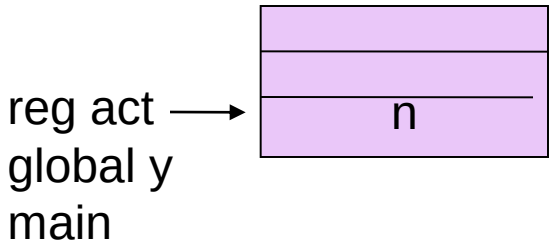
```
int f1()
 {
   int a;
      ...
 }
void main ()
 {
   int b;
   ...
   f1();
   ..
 }
```

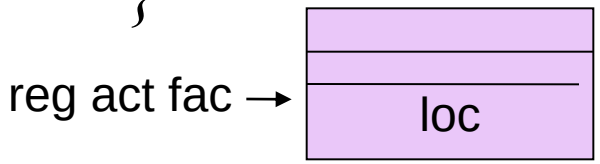
C3: Ejemplo del factorial

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");
}
```



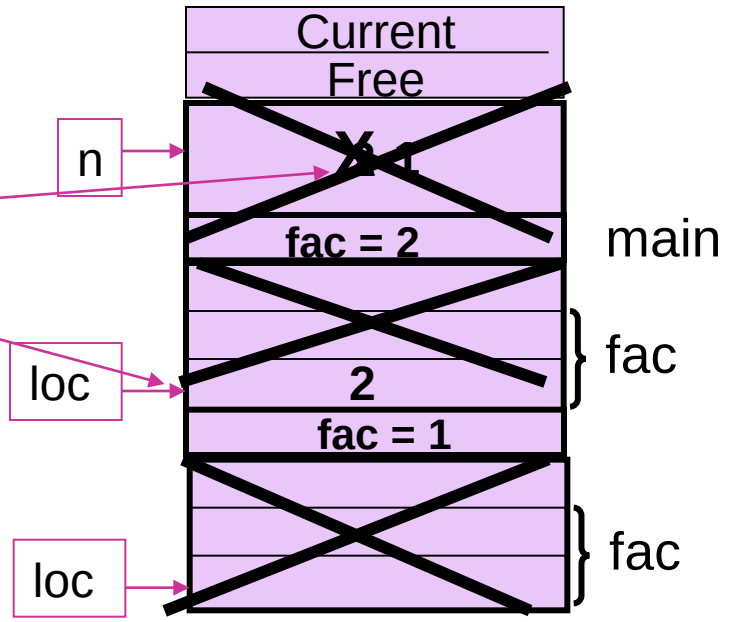


C3: Ejemplo del efecto en memoria de la ejecución de factorial

main fac fac

```
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");
}
```

Invocación a fac con n=2



imprime 2

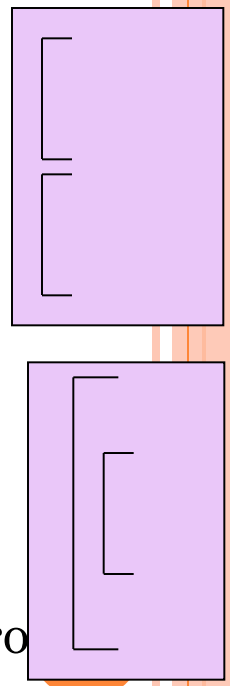
C4: Estructura de bloque

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

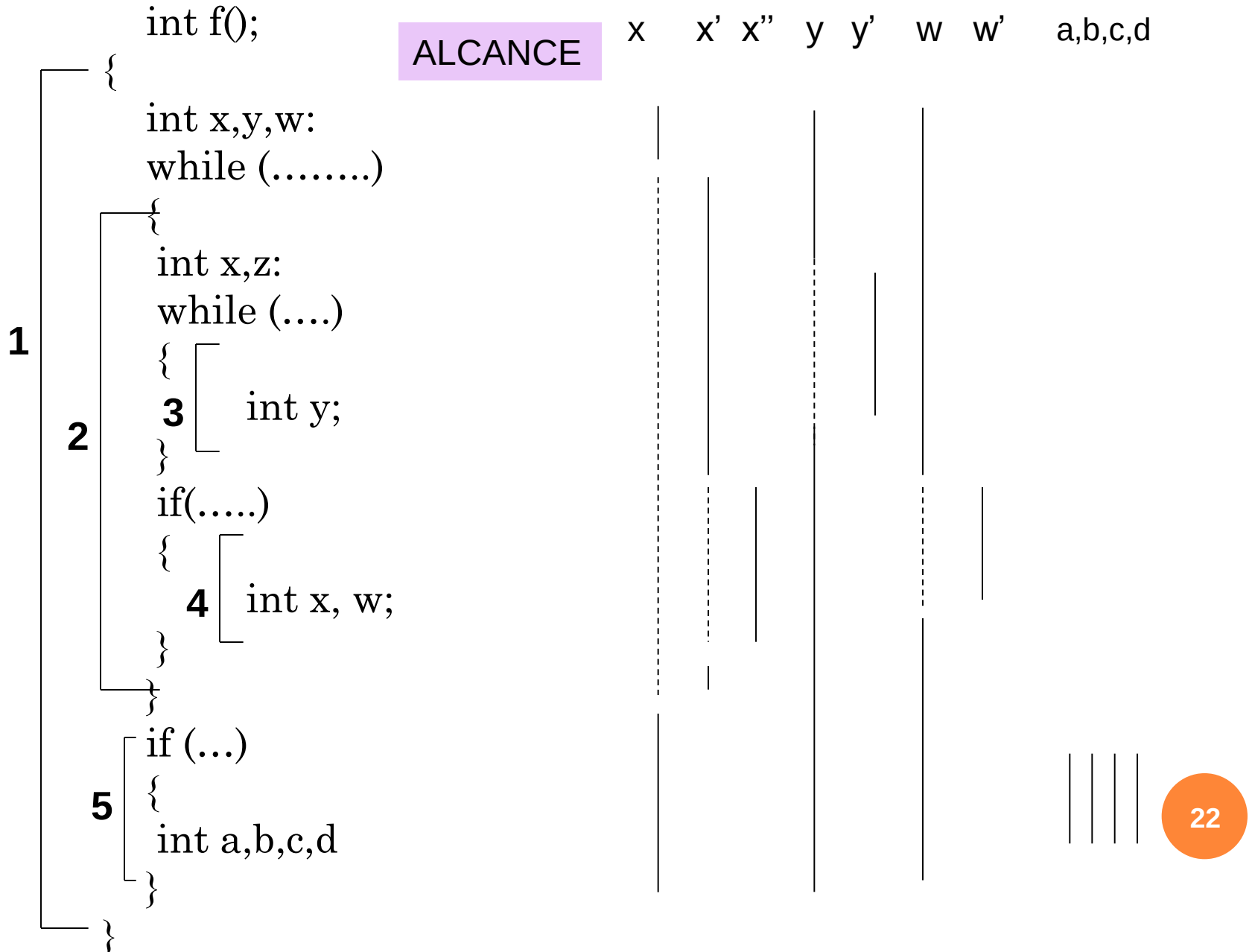
C4': Anidamiento vía sentencias compuestas

- Un bloque tiene forma de una sentencia compuesta:

  {<lista de declaraciones>;<lista de sentencias>}

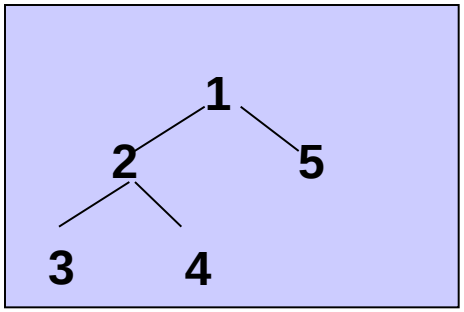
- Las variables tienen alcance local: son visibles dentro de la sentencia compuesta, incluyendo cualquier sentencia compuesta anidada en ella
- Si en el anidamiento, hay una nueva declaración de un nombre, la declaración interna enmascara la externa del mismo nombre.

C4': SENTENCIAS COMPUESTAS



C4': Sentencias compuestas

Alocación: implementación



estático

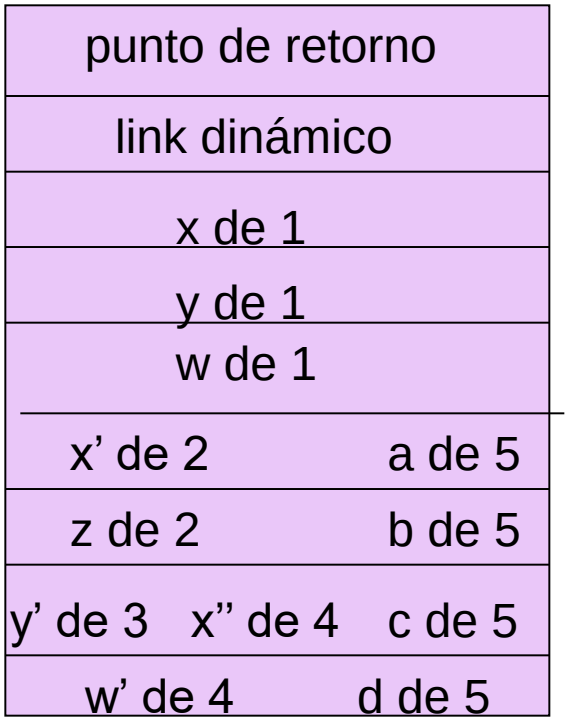
incluir las necesidades dentro del registro de activación de la unidad a la que pertenece

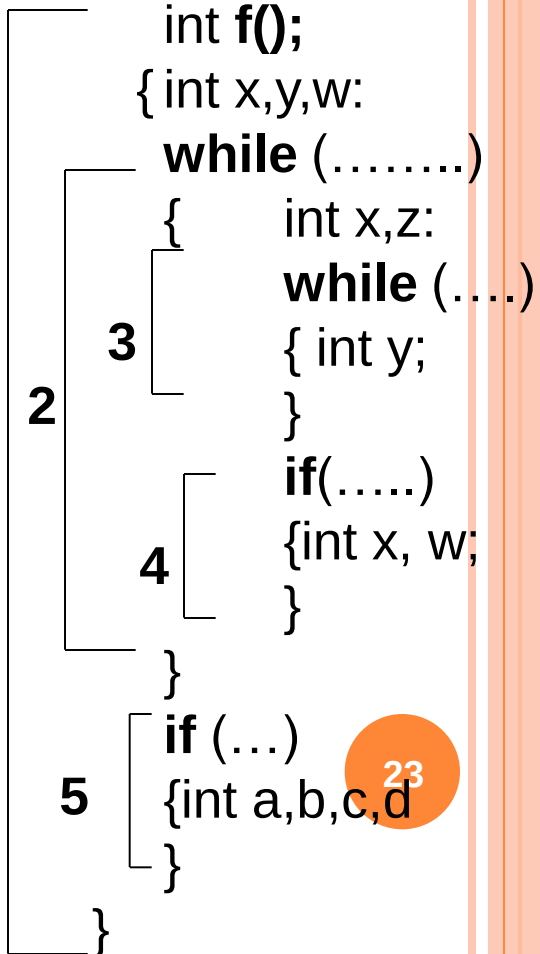
simple y eficiente en tiempo

dinámico

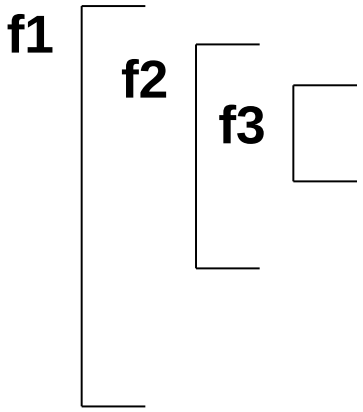
alocar el espacio dinámicamente cuando se ejecutan las sentencias

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;
    }
}
```

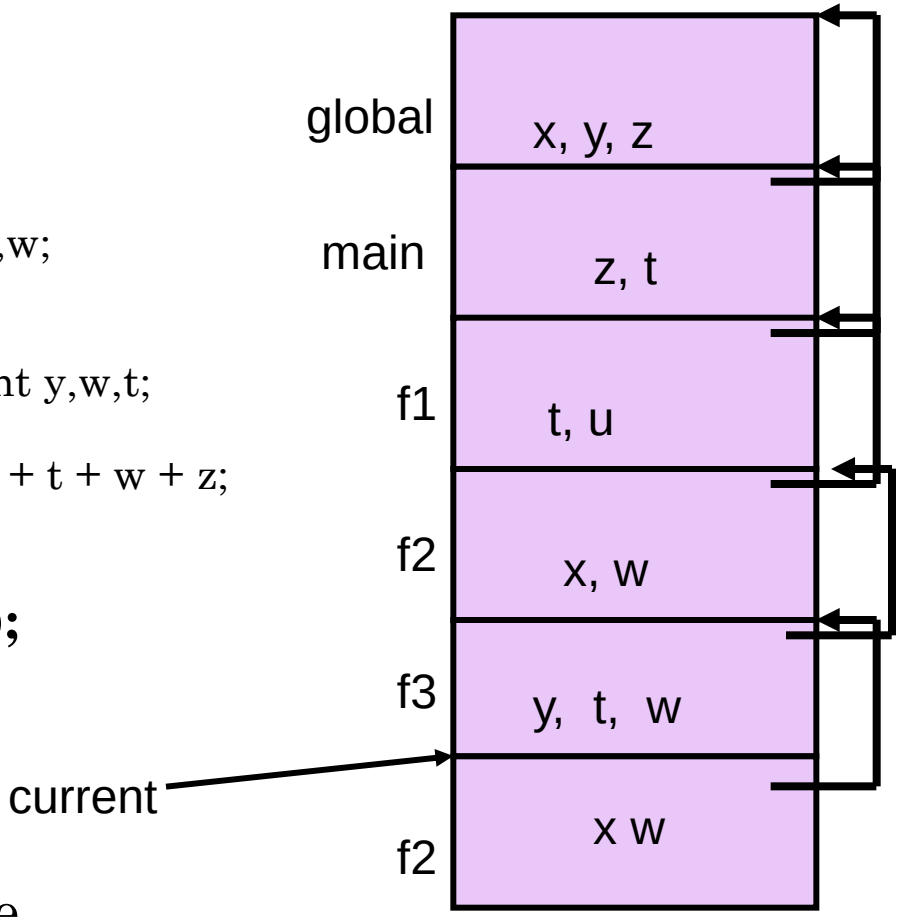
```
main ();
{
    int z,t;

}
//end file
```

x = y + t + w + z

local global f1 local global

C4": Rutinas anidadas
Acceso al ambiente no-local

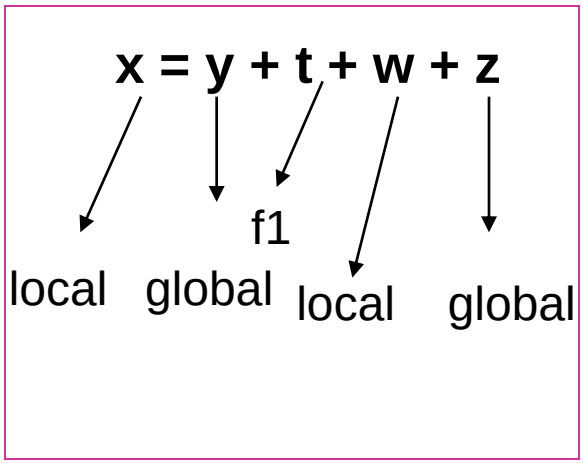
```
//file
int x,y,z;
f1()
{
   int t,u;
   f2()
    {
      int x,w;
      f3()
       {
          int y,w,t;
       }
       x = y + t + w + z;
      }
}
main ();
{
int z,t;

}
//end file
```



Secuencia de llamadas:

main f1 f2 f3 f2

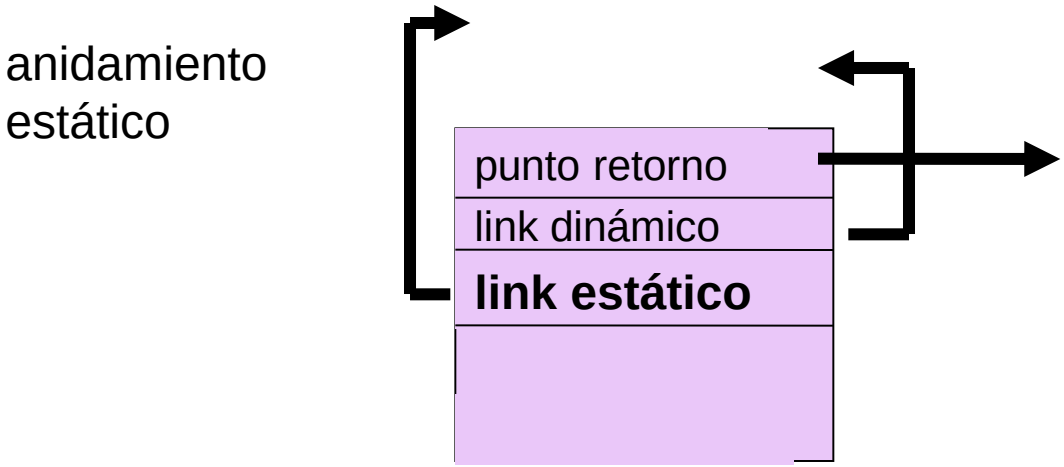
cadena dinámica



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	aloca espacio para el v. de retorno
set D[1],ip+ 5	salva el punto de retorno
set D[1]+1,D[0]	salva el link dinámico
set 0,D[1]	
set 1,D[1]+ size	
ip+ 5 jump comienzo	llamada

Se agrega una sentencia:

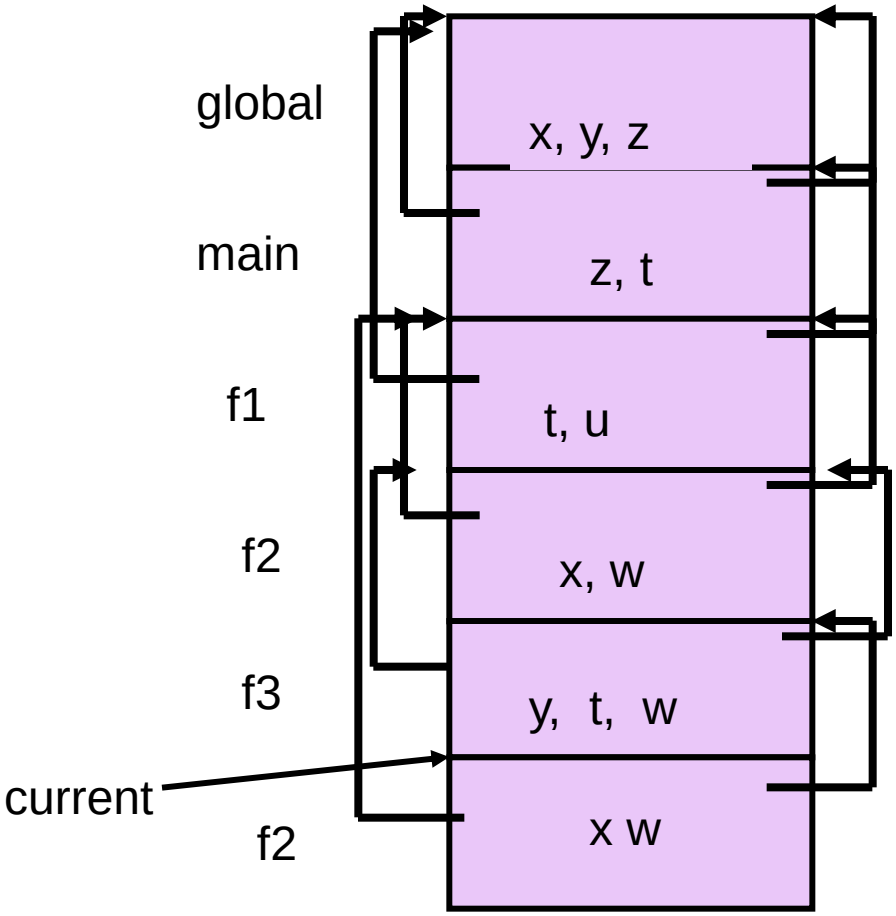
Colocar en D[1] + 2 el link estático

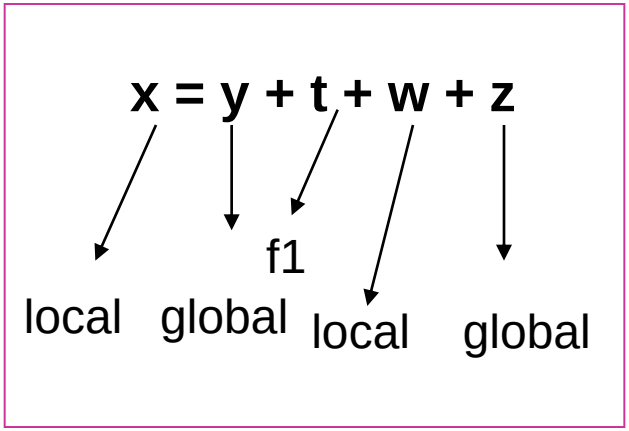
set 1,D[0]	reestablece el free
set 0,D[D[0]+1]	reestablece el current
jump D[D[1]]	bifurca al punto de retorno

link dinamico

return

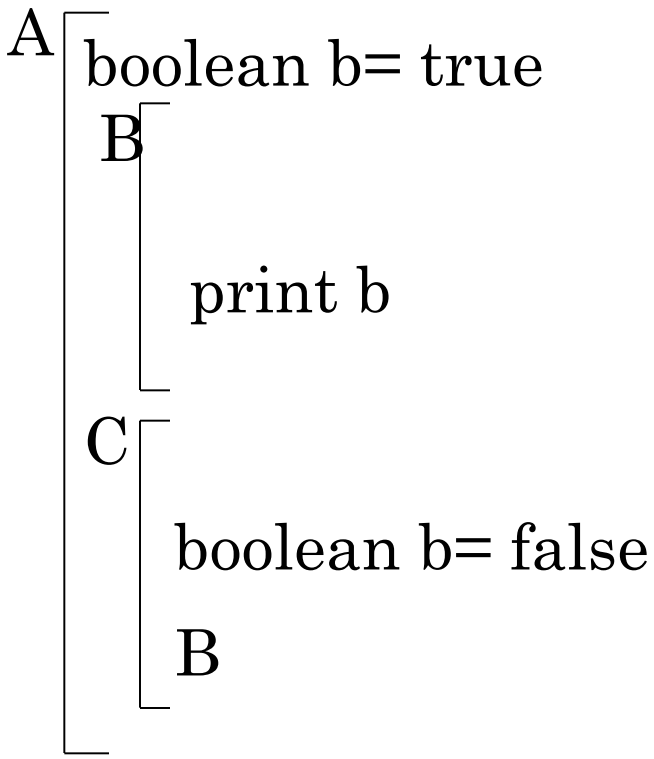
C4”: Cadena estática





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

Cadena estática ≠ Cadena dinámica



- A C B
  - estático
    - b es la de A → true
  - dinámico
    - b es la de C → 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

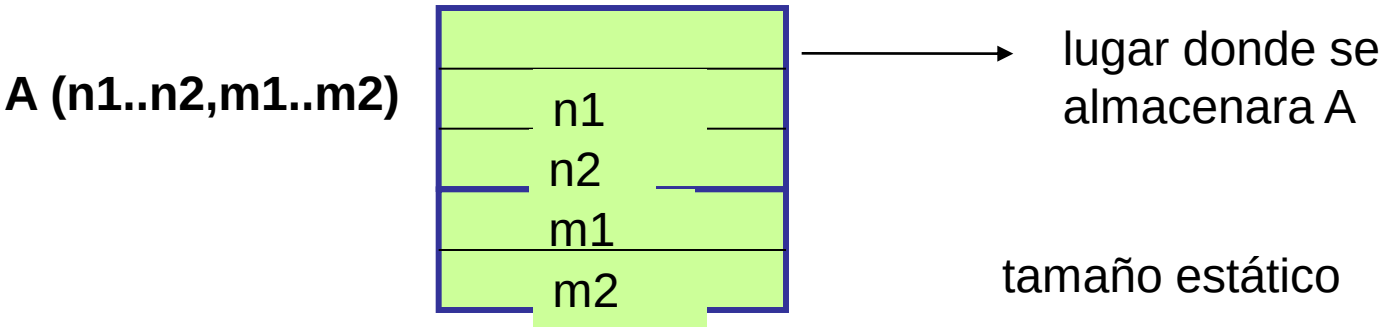
A: VECTOR (0..N);

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

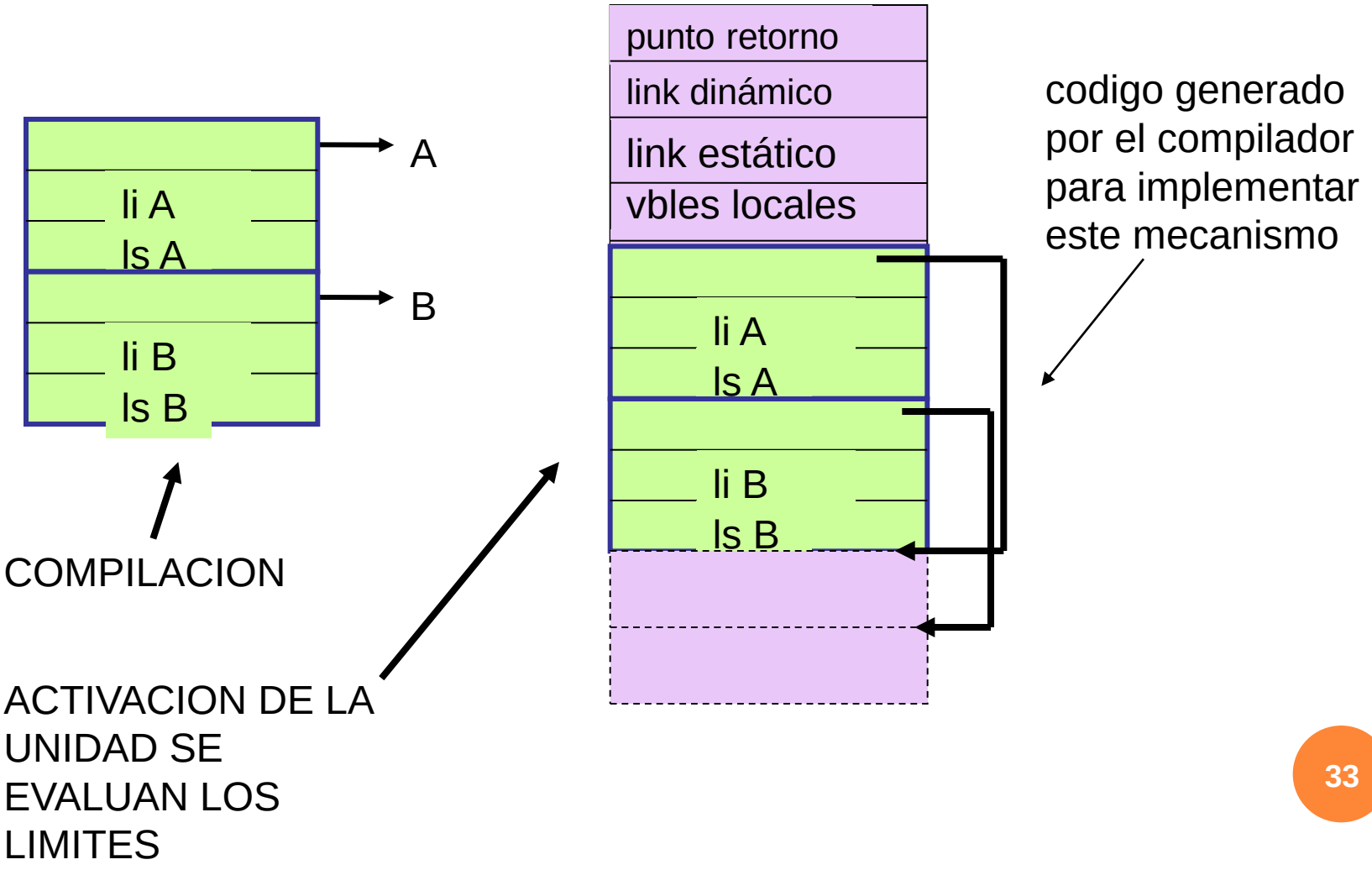
- **COMPILACION:** se reserva lugar en el registro de activación para los **descriptores** de los arreglos dinámicos.



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

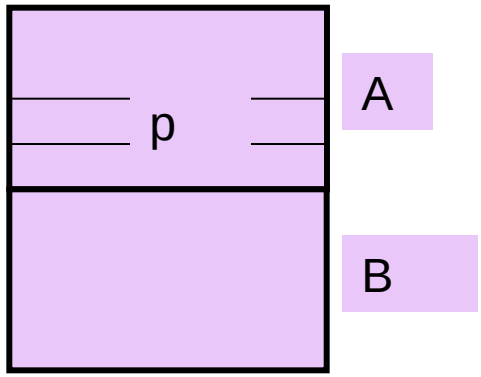
donde alocar el objeto apuntado?

A
p pointer to integer

B
new p

new p
B

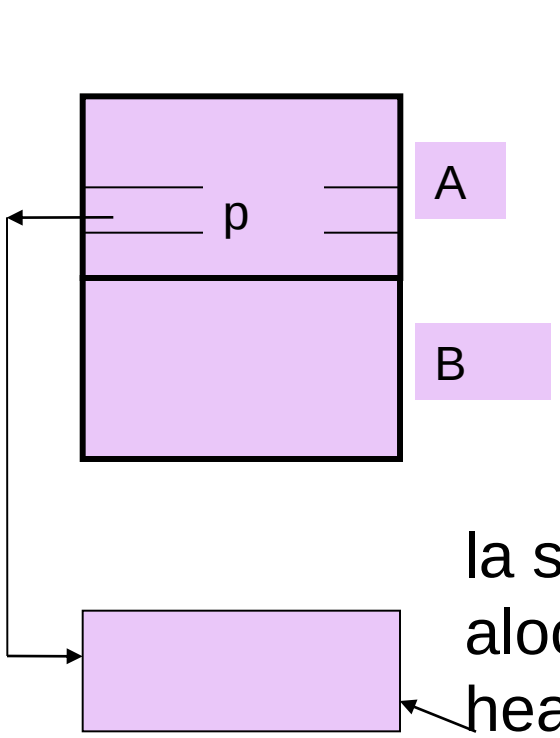
A B



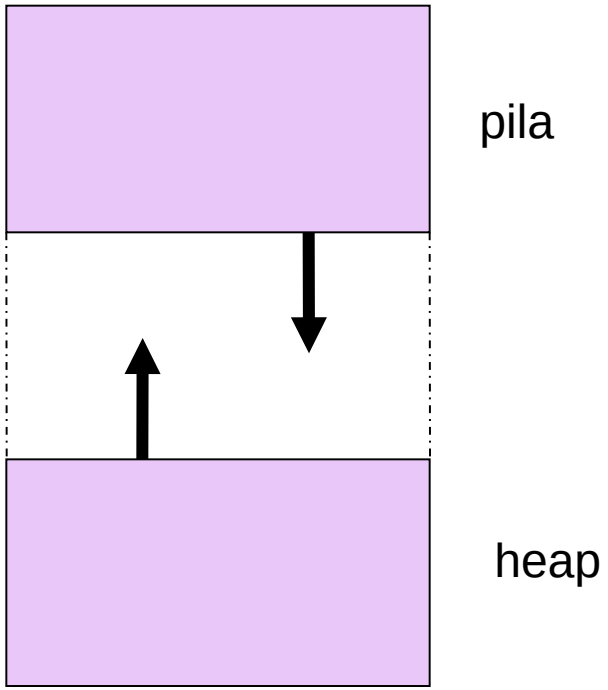
el reg de Act de A no esta en el tope

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



la solución es alocarla en la heap



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

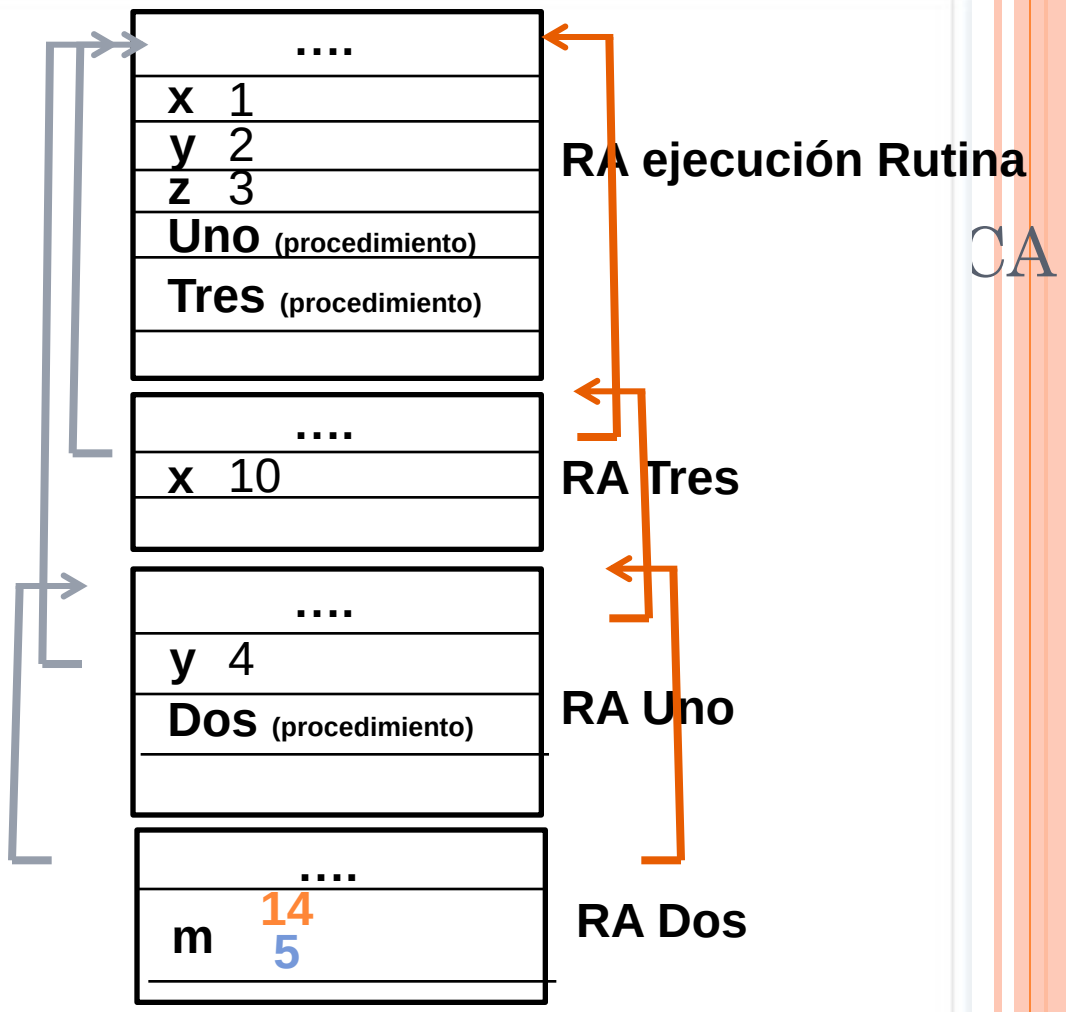
- Variables semidinámicas C5’

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.
```



Resultados por:

Cadena Dinámica

Cadena Estática