

SEMANTICA OPERACIONAL

Formas de comunicación entre las rutinas - Parámetros

Rutinas

- llamadas **subprogramas, subrutinas**, son una **Unidad de Programa** (función, procedimiento. *(según los distintos lenguajes)*)
- Están formadas por un **conjunto de sentencias** que representan una **acción abstracta**
- **Permiten al programador definir una nueva operación** a semejanza de las operaciones primarias ya integradas en el lenguaje
- **Permiten ampliar** a los lenguajes, dan **modularidad, claridad y buen diseño**
- Se **lanzan** con una **llamada explícita (se invocan por su nombre)** y luego retornan a **algún punto de la ejecución** (responden al ***esquema call/return***)

Los **subprogramas** son el ejemplo más usual y útil presente desde los primeros lenguajes ensambladores.

Rutinas

- Formas de **Subprogramas**
 - **Procedimientos** (no devuelven valor o null)
 - **Funciones** (devuelven un valor)

Semánticamente distintos, veamos sus características

Rutinas

■ Tipo:

■ Procedimientos

- Un **procedimiento** es una **construcción** que permite **dar nombre a un conjunto de sentencias y declaraciones asociadas** que se usarán para **resolver un subproblema** dado.
- Brindará una **solución de código más corta, comprensible y fácilmente modificable.**
- Permiten al **programador definir y crear nuevas acciones/sentencias.**
- El **programador** las **invocará**
- **Pueden no recibir ni devolver ningún valor.**
- Los resultados los produce en **variables no locales** o en **parámetros** que **cambian su valor.**

Rutinas

■ Tipo:

■ Funciones

- Mientras que **un procedimiento ejecuta un grupo de sentencias**, una **función además devuelve un valor al punto donde se llamó.**
- El **valor** que **recibe** la función se usa para **calcular** el valor total de la **expresión** y **devolver** algún **valor.**
- Permite al programador **crear nuevas operaciones.**
- Similar a las funciones matemáticas ya que hacen algo y luego devuelven un valor y no producen efectos colaterales.
- **Se las invoca dentro de expresiones y lo que calcula reemplaza a la invocación dentro de la expresión.**
- **Siempre** **deben retornar un valor.**

Rutinas/Subprogramas

■ Conclusiones:

- Cuando se diseña un subprograma el programador se concentra en el cómo trabajará dicho subprograma.
- Cuando se usa un subprograma se ignorará el cómo. Sólo interesará el qué me permite hacer. (La implementación permanece oculta)

Abstracción

- Con una sola definición se pueden crear muchas activaciones. La definición de un subprograma es un patrón para crear activaciones durante la ejecución.
- Un subprograma es la implementación de una acción abstracta y su invocación representa el uso de dicha abstracción.
- Codificar un subprograma es como si hubiéramos incorporado una nueva sentencia a nuestro lenguaje.

Formas de conectar y "compartir datos" entre diferentes Unidades de Programa:

- Var locales (que hay muchos)
- Var No locales

USO AMBIENTE
NO LOCAL

USO PARAMETROS

Subrute
NO local
IMPLICITO

AMBIENTE
comun
explícito

X Almacen
estático

X Almacen
DINAMICO

AREAS
COMUNES

Formas de conectar y "compartir datos" entre diferentes Unidades de Programa:

Resumen:

1. Variables locales (**no hay problema**)
2. Variables no locales:
 - 1) acceso al ambiente no local (**puede llevar a errores, puede ser menos claro**)
 - Ambiente no local implícito (**Es automático**)
 - Utilizando ***regla de alcance dinámico*** (**quién me llamó**)
 - Utilizando ***regla de alcance estático*** (**dónde está contenido**)
 - Ambiente común explícito (**interviene el programador**)
 - Se definen **áreas comunes** de código
 - 2) A través del uso de parámetros (**la mejor forma de compartir y más clara**)

Formas de conectar y "compartir datos" entre diferentes Unidades de Programa:

Procedure Main;

var x,z,n: integer;

Procedure A1()

var m: integer;

Begin

m:=3; x:= x+m+1; z:=z+1+n;

end;

Procedure A2()

var x, z: integer;

Procedure A3();

var z, n: integer;

begin

n:=3; z:= x + n; **A1();**

end;

begin

x:= 1; z:= x +n; **A3();**

end;

begin

x:=2; z:=1; n:=4; **A2();**

end.

- **A través del acceso al ambiente no local – Ambiente no local implícito**

¿Que valores resultarán?

- **todos usan x z n**
- **Main llama A2(),**
- **A2() llama a A3(),**
- **A3() llama a A1(),**

El resultado dependerá de si usa alcance dinámico o alcance estático

Formas de conectar y "compartir datos" entre diferentes Unidades de Programa:

1) A través del acceso al ambiente no local

■ Ambiente común explícito

- Se **definen áreas comunes de código**
- El **programador** debe **especificar** que **comparte**
- **Cada lenguaje** tiene **su forma** de realizarlo

■ Ejemplos :

- **ADA**: uso de **paquetes** con cláusula **PACKAGE** (para **Tipos Abstractos de Datos - TAD**)
- **PL/1**: **variables externas** con cláusula **DECLARE**,
- **FORTRAN**: **área común** con cláusula **COMMON**

Formas de conectar y "compartir datos" entre diferentes Unidades de Programa:

- Var locales (que hay muchos)
- Var No locales

USO AMBIENTE
NO LOCAL

USO PARAMETROS

Subrute
NO local
IMPLICITO

AMBIENTE
comun
explícito

X Almacen
estático

X Almacen
DINAMICO

AREAS
COMUNES

Formas de conectar y "compartir datos" entre diferentes Unidades de Programa:

2) Pasaje de Parámetros

– *Parámetro Real (Argumento):*

- Es un **valor u otra entidad** que se **pasa** a un **procedimiento o función**.
- Están **colocados** en la parte de la **invocación de la rutina**

– *Parámetro Formal (Parámetro):*

- Es una **variable** utilizada para **recibir valores de entrada** en una **rutina, subrutina etc.**
- Están **colocados** en la parte de la **declaración de la rutina**

Formas de conectar y "compartir datos" entre diferentes Unidades de Programa:

El pasaje de parámetros es mejor:

- **Porque el uso intensivo de *accesos al ambiente no local* puede provocar alguna *pérdida de control*, y puede provocar que las variables terminen siendo visibles donde no es necesario y llevar a *errores*.**

¿Qué otras ventajas tienen el uso de parámetros respecto a la forma de compartir accediendo al ambiente no local?

Ventajas del Pasaje de Parámetros

- **Flexibilidad**, se pueden transferir **más datos y de diferente tipo en cada llamada**.
- **Abstracción**, permite **compartir en forma más abstracta**, solo especificamos el **nombre a argumentos y parámetros** y el **tipo** de cada cosa que se comparta.
- **protección**: el uso intensivo de **accesos al ambiente no local** decrementa la **seguridad** ya que las **variables terminan siendo visibles** aun **donde no es necesario** o donde no deberían.

Ventajas del Pasaje de Parámetros

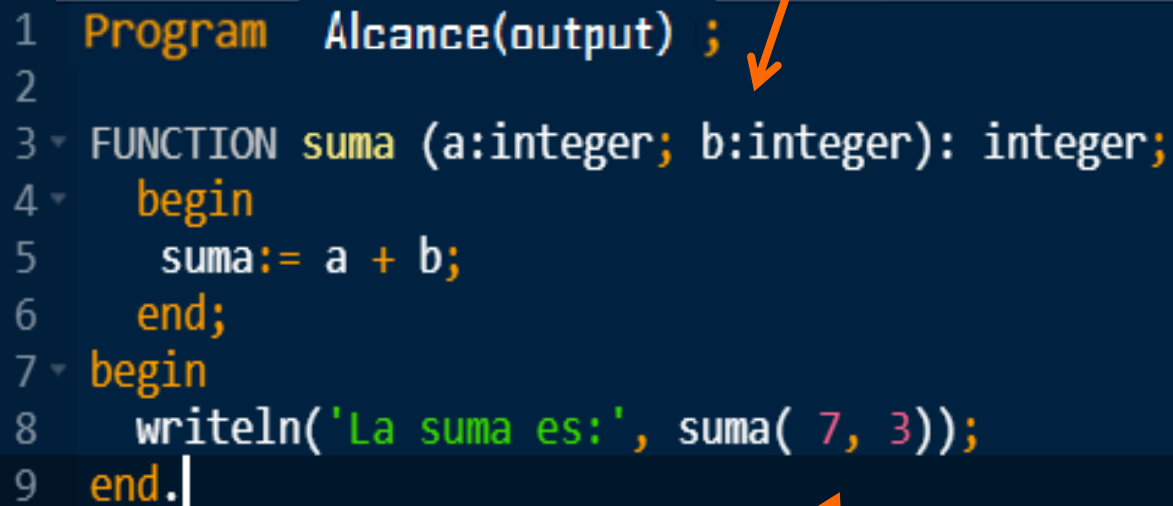
- **Legibilidad**: hace más **claro** y **entendible** el código al no tener accesos al ambiente no local, menos repetitivo, ayuda a **encontrar** más fácilmente los **errores**, sabemos que compartimos.
- **Modificabilidad**: si hay **errores** uno se focaliza en **qué cosas comparto, para que** los estoy utilizando y su **tipo**. Ese cambio **aplica a todos**. Fácil de **depurar**, y no chequear cada repetición en el código.

**una buena cualidad de un programador es
minimizar el acceso a datos no locales !!**

Parámetros

Parámetros Formales en declaración (PARÁMETRO)

```
1 Program Alcance(output) ;  
2  
3 FUNCTION suma (a:integer; b:integer): integer;  
4 begin  
5     suma:= a + b;  
6 end;  
7 begin  
8     writeln('La suma es:', suma( 7, 3));  
9 end.
```



The diagram illustrates the distinction between formal and real parameters in Pascal. An orange arrow points from the text 'Parámetros Formales en declaración (PARÁMETRO)' to the parameter list '(a:integer; b:integer)' in the function declaration on line 3. Another orange arrow points from the text 'Parámetros Reales en invocación (ARGUMENTOS)' to the argument list '(7, 3)' in the function call on line 8.

Parámetros Reales en invocación (ARGUMENTOS)

Parámetros

- **¿Los parámetros formales son variables locales?**

Si, Un parámetro formal es una **variable local** a su **entorno**.

Se declara con una **sintaxis** particular a **cada lenguaje**.
Sirve para **intercambiar información** entre la función/rutina que hace la llamada y la que la recibe.

- **¿Qué datos pueden ser los parámetros reales?**

Un parámetro real puede ser un **valor, entidad, expresión,** etc., que pueden ser **locales, no locales o globales,** y que se especifican en la llamada a una función/rutina dentro de la unidad llamante.

Lo importante es que depende de cada lenguaje y hay que conocerlos

Vinculación de los Parámetros

■ **Momento de vinculación entre los PR y PF comprende la evaluación de los parámetros reales y la ligadura con los parámetros formales**

- **Evaluación:**

1. En general **antes de la invocación** primero se **evalúan** los **parámetros reales**, y **luego** se hace la **ligadura**. **Se verifica que todo esté bien antes de transferir el control a la unidad llamada.**

- **Ligadura:**

1. **Por posición:** Se **corresponden** con la **posición** que ocupan en la lista de parámetros. **Van en el mismo orden**
2. **Por Nombre o palabra clave:** Se **corresponden** **con el nombre** por lo tanto pueden estar **colocados en distinto orden** en la lista de parámetros.

Parámetros

- Evaluación de los parámetros reales y ligadura con los parámetros formales

Ejemplos:

- en **Python light**

llamo con ligadura por nombre

$P(y \Rightarrow 4; x \Rightarrow z);$

Procedure P (x : IN integer, y : IN float)

Llamo con PR a $P(4,Z)$ y recibe PF en $P(Z,4)$

Desventaja: cuando invocas **hay que conocer/recordar el nombre de los parámetros formales para poder hacer la asignación**. Esto puede llevar a cometer **errores**.

Parámetros

Evaluación de los parámetros reales y ligadura con los parámetros formales

Ejemplos:

- Cada lenguaje tiene sus reglas sintácticas y semánticas para el pasaje de parámetros. Hay que conocerlos para evitar errores.
 - En **Ada** se pueden usar **ambos métodos**.
 - En **C++, Ada, Python, JavaScript**, los parámetros formales en la **definición** pueden tener *valores por defecto* (se puede *realizar una asignación*), *no todos en invocación* (no es necesario listarlos todos en la invocación)

```
function saludar(nombre = 'Miguel Angel') {  
  console.log('Hola ' + nombre);  
}
```

Ej. JavaScript

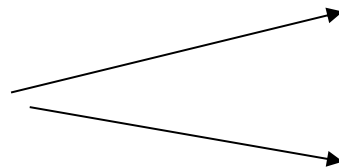
Esta función recibe un parámetro llamado "nombre", con un valor predeterminado. Este valor se asignará en caso que al invocar a la función no le pasemos nada.

```
saludar();
```

Eso produciría la salida por consola "Hola Miguel Angel".

Parámetros

**Tipos de
Parámetros:**



1) Datos

2) Subprogramas

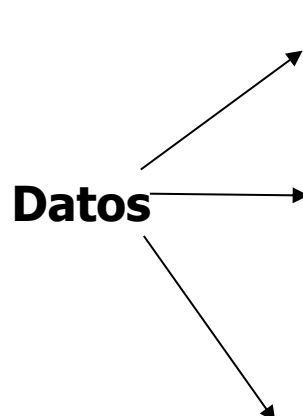
- Cada uno tiene distintos subtipos
- Depende de cada lenguaje

Parámetros

1) Datos pasados como Parámetros

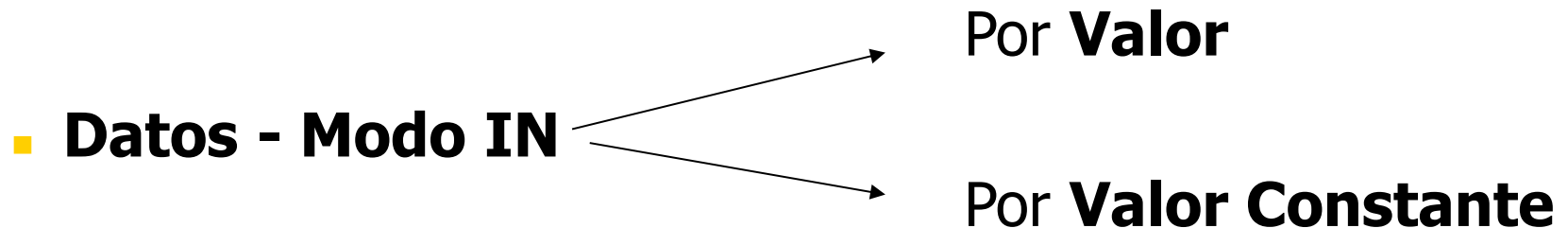
Hay diferentes formas de transmitir los parámetros hacia y desde la rutina llamada.

Desde el punto de vista semántico la **clasificación del tipo de comunicación permitida para los parámetros** puede ser:

- 
- Datos**
- **Modo IN:** El *parámetro formal recibe* el dato desde el *parámetro real*. La *conexión es al inicio, se copia y se corta la vinculación*
 - **Modo OUT:** se invoca la rutina y cuando esta termina *devuelve el parámetro formal al parámetro real*. La *conexión es al final*
 - **Modo IN/OUT:** El *parámetro formal recibe* el dato del parámetro real y el *parámetro formal le envía* el dato al parámetro real al finalizar la rutina. La *conexión es al inicio y al final*

Parámetros

Diferentes formas de transmitir los parámetros



Conexión al inicio

Parámetros

Datos - Modo IN - por Valor:

- El *valor* del *parámetro real* se usa para *inicializar* el correspondiente *parámetro formal* al invocar la unidad.
- Se *transfiere* el *dato real* y *se copia*
- En este caso el *parámetro formal actúa como una variable local de la unidad llamada, y crea otra variable.*
- la *conexión es al inicio* para pasar el valor y *se corta la vinculación.*
- *Es el mecanismo por default y el más usado*

Desventaja:

- **consume tiempo** para hacer la copia de cada parámetro
- **consume almacenamiento** para duplicar cada dato (pensar grandes volúmenes)

Ventaja:

- **protege los datos** de la unidad llamadora, el **parámetro real no se modifica.**
- **No hay efectos negativos** o colaterales

Parámetros

Veamos **ejemplos** de cómo funciona el **pasaje de parámetros**, como es el **manejo de la pila (stack)**, y como se crea el **registro de activación** *para contener los objetos necesarios para su ejecución, eliminándolos una vez terminada.*

Parámetros

Datos – Modos IN– Por Valor

Program main

```
var i:integer;
```

```
Procedure P(a:integer)
```

```
var x,y: integer;
```

```
Begin
```

```
  a=a+3;
```

```
  x=a+1;
```

```
  y=x+1;
```

```
end;
```

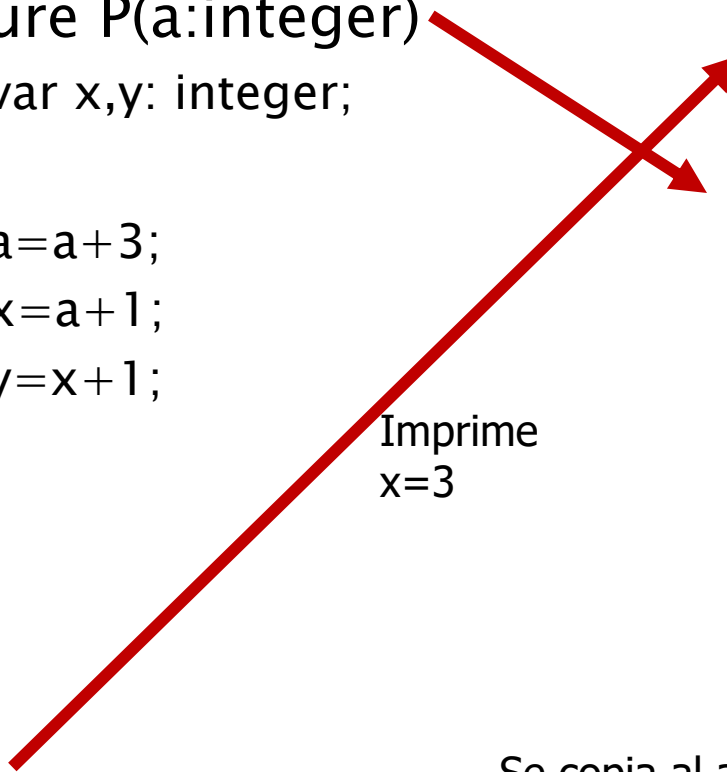
```
Begin
```

```
  i=3;
```

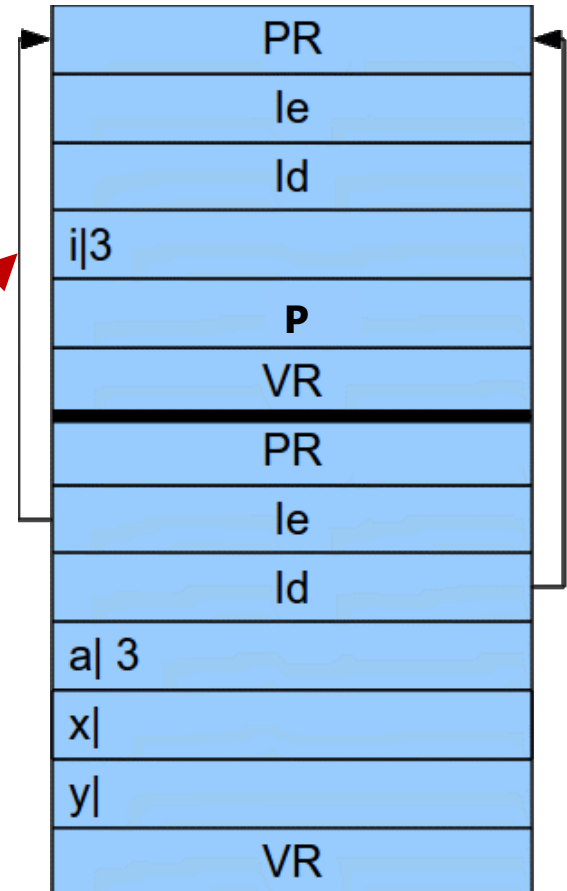
```
  P(i);
```

```
  Print(i);
```

```
End.
```



Imprime
x=3



Se copia al aloca el registro en memoria

PASCAL

Parámetros

Datos – Modos IN– Por Valor
Program main

```
var i:integer;
```

```
Procedure P(a:integer)
```

```
var x,y: integer;
```

```
Begin
```

```
a=a+3;
```

```
x=a+1;
```

```
y=x+1;
```

```
end;
```

```
Begin
```

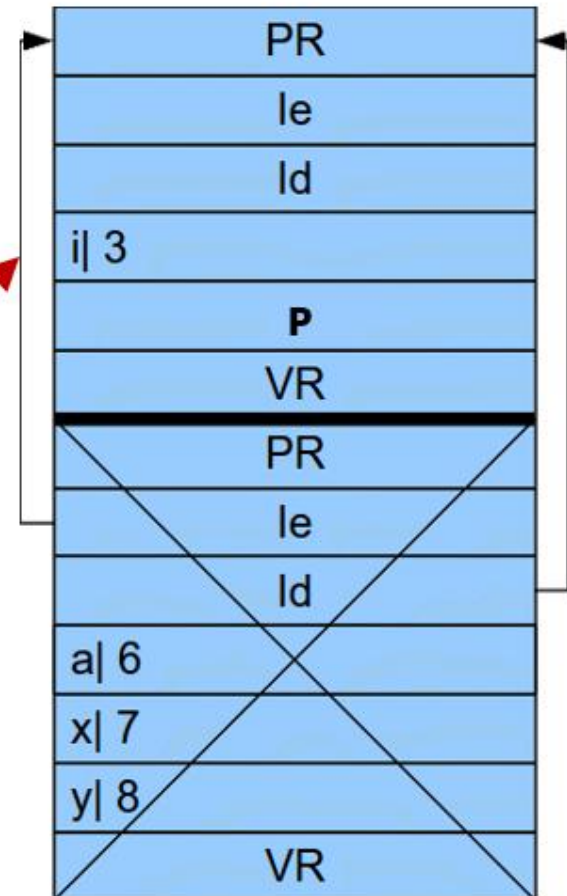
```
i=3;
```

```
P(i);
```

```
Print(i);
```

```
End.
```

RESULTADO FINAL



Imprime
x=3

Se copia al aloca el registro en memoria

PASCAL

Parámetros

Datos - Modo IN - Por valor constante:

- **No** todos los lenguajes permiten el modo **IN** con pasaje por valor constante.
- Se **envía un valor**, pero la rutina receptora **no puede modificarlo**, queda con **un valor fijo**. (*Regla de semántica estática de ese lenguaje*)
- **No** indica si se realiza o no la copia. (*dependerá del lenguaje*)
- La implementación **DEBE** contemplar que el **parámetro real no sea modificado** (*requiere un control*)

Ejemplos:

C/C++ usa const en declaración

```
void ActualizarMax( const int x, const int y )  
{if ( x > y ) Max= x ;  
  else Max= y ;}
```

Parámetros

Datos - Modo IN - Por valor constante:

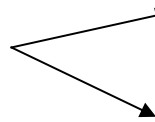
Desventaja:

- requiere realizar **más trabajo** para implementar los **controles**.

Ventaja:

- **protege** los **datos** de la **unidad llamadora** (el **parámetro real no se modifica**)

Parámetros

- **Datos - Modo OUT** 
 - Por **Resultado**
 - Por **Resultado de funciones**

**El Parámetro Formal devuelve el resultado al Parámetro Real.
La conexión se da al final de la ejecución de la rutina.**

Parámetros

• Datos - Modo OUT - Por Resultado:

- La **conexión** es al final
- El **valor** del **parámetro formal** (rutina) se **copia al parámetro real al terminar de ejecutarse** la unidad llamada.
- El **parámetro formal** es una **variable local del entorno de la rutina**
- El **parámetro formal** es una **variable sin valor inicial** porque **no recibe nada**. *¿Hay regla de inicialización por defecto en el lenguaje?*

Parámetros

■ Datos - Modo OUT - Por Resultado:

■ **Desventajas:**

- **Consume tiempo y espacio** porque **hace copia al final** (*pensar grandes volúmenes de datos*)
- **Debemos inicializar la variable en la unidad llamada** de alguna forma (*si el lenguaje no lo hace por defecto*)

■ **Ventaja:**

- **protege los datos** de la **unidad llamadora**, el **parámetro real no se modifica** durante la ejecución de la unidad llamada. (*No hay vínculo entre el PF y PR durante la ejecución del cuerpo de la rutina*)

Parámetros

Datos – Modos OUT– Por Resultado

Program main

var i:integer;

Procedure P(res a:integer)

var x,y: integer;

Begin

a=3

Inicializa a

x=a+1;

y=x+1;

a=y;

Asigna valor a devolver

end;

Begin

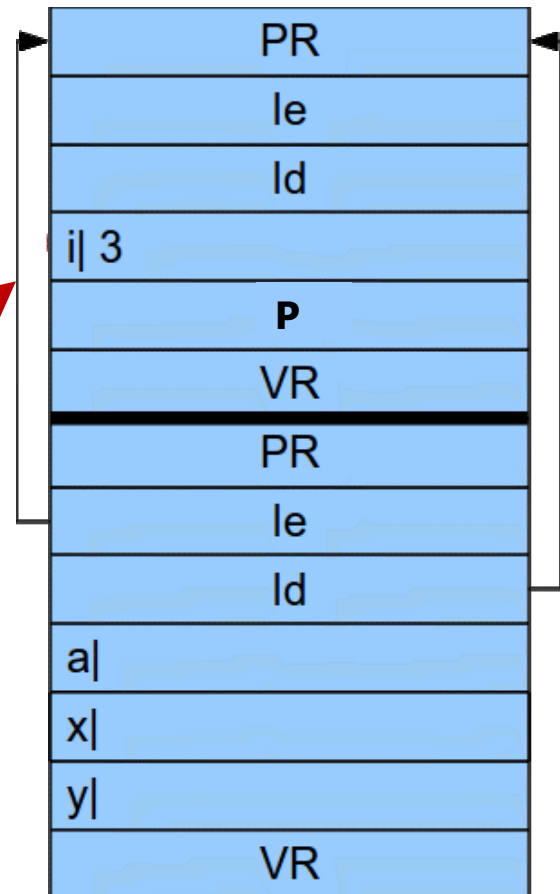
i=3;

P(i);

Print(i);

End.

Imprime 5



Antes de desalocar el registro de memoria se copia el valor del parámetro F. en el registro que llamó al proc o fun.

Parámetros

RESULTADO FINAL

Datos – Modos OUT– Por Resultado

Program main

var i:integer;

Procedure P(res a:integer)

var x,y: integer;

Begin

a=3

Inicializa a

x=a+1;

y=x+1;

a=y;

Asigna valor a devolver

end;

Begin

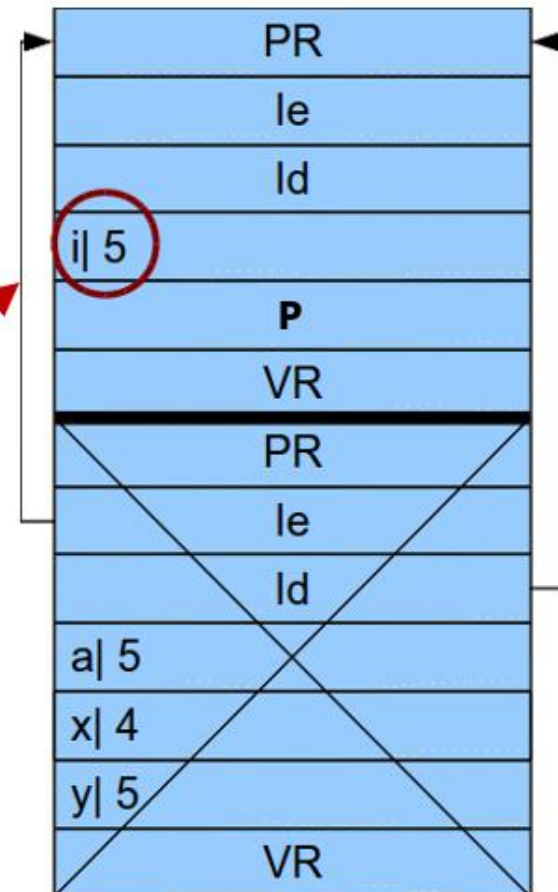
i=3;

P(i);

Print(i);

End.

Imprime 5



Antes de desalocar el registro de memoria se copia el valor del parámetro F. en el registro que llamó al proc o fun.

Parámetros

- **Datos - Modo OUT - Por Resultado de funciones:**
 - Es el **resultado** que me devuelven las funciones.
 - El **resultado** se **reemplaza** en la **invocación** de la **expresión** que contiene el llamado.
 - **Distintas formas de devolución** según lenguaje:
 - ***return*** como en **Python, C, etc.** (*valor o expresión*)
 - ***nombre de la función*** (*ultimo valor asignado*) que se considera como una **variable local** como en **Pascal**.

■ Ejemplos:

En C

```
int f1(int m);  
{...  
    return(m)  
}
```

En Pascal

```
Function F1(m:integer):integer;  
begin  
    F1:=m + 5;  
end;
```

```
1  int addition(int a, int b)  
2  {  
3      return (a + b);  
4  }  
5  int main()  
6  {  
7      int x = 10;  
8      int y = 20;  
9      int z;  
10  
11      z = addition(x, y);  
12  }
```

Parámetros

- **Datos - Modo IN/OUT**
 - Por **Valor-Resultado**
 - Por **Referencia**
 - Por **Nombre**

Conexión al inicio y al final

Parámetros

Datos - Modo IN/OUT - Por Valor/Resultado:

- La **rutina recibe** un **valor** y **devuelve** un **resultado**.
- Cuando se **invoca la rutina**, el **parámetro real** le da **valor al parámetro formal** (se genera copia) y se **desliga** en ese momento.
- La **rutina trabaja** sobre **ese parámetro formal** pero **no afecta al parámetro real** **trabaja sobre su copia**. Cada **referencia** al parámetro formal es una referencia local.
- una vez **que termina de ejecutar** el **parámetro formal** le **devuelve un valor al parámetro real** y se genera copia.
- Se dice que hay una ligadura y una conexión entre **parámetro real y el formal** cuando se **inicia la ejecución** de la rutina y cuando se **termina, pero no en el medio**.
- **Tiene las desventajas y las ventajas de ambos.**

Parámetros

Modos IN/OUT- Por Valor/Resultado

Program main

```
var i:integer;
```

```
Procedure P(in-out a:integer)
```

```
    var x,y: integer;
```

```
Begin
```

```
    a=4
```

```
    x=a+1;
```

```
    y=x+1;
```

```
    a=y;
```

```
end;
```

```
Begin
```

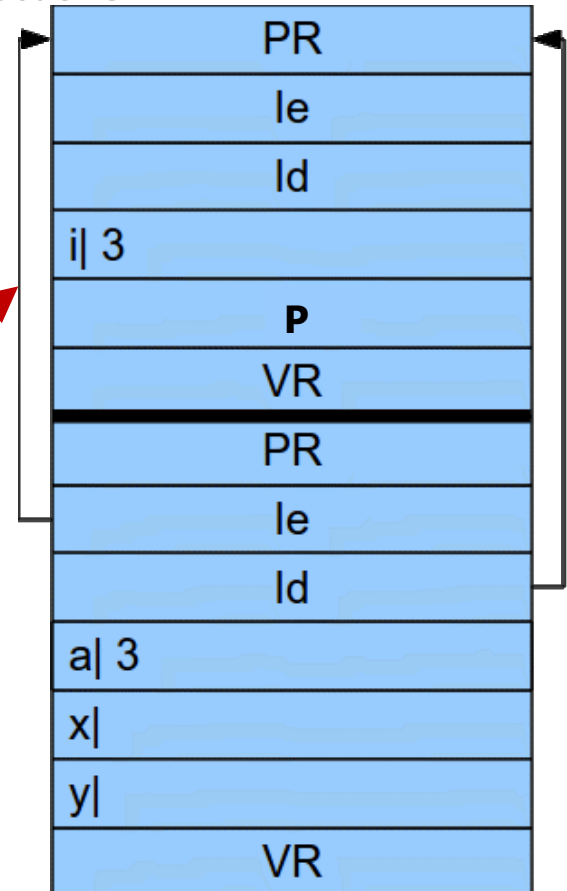
```
    i=3;
```

```
    P(i);
```

```
    Print(i);
```

```
End.
```

Imprime 6



Se copia el valor al aloear el registro de la rutina, se modifica el parámetro formal, al finalizar la ejecución de la rutina se copia al parámetro real antes de desalocar.

Parámetros

Modos IN/OUT- Por Valor/Resultado

Program main

var i:integer;

Procedure P(in-out a:integer)

var x,y: integer;

Begin

a=4

x=a+1;

y=x+1;

a=y;

end;

Begin

i=3;

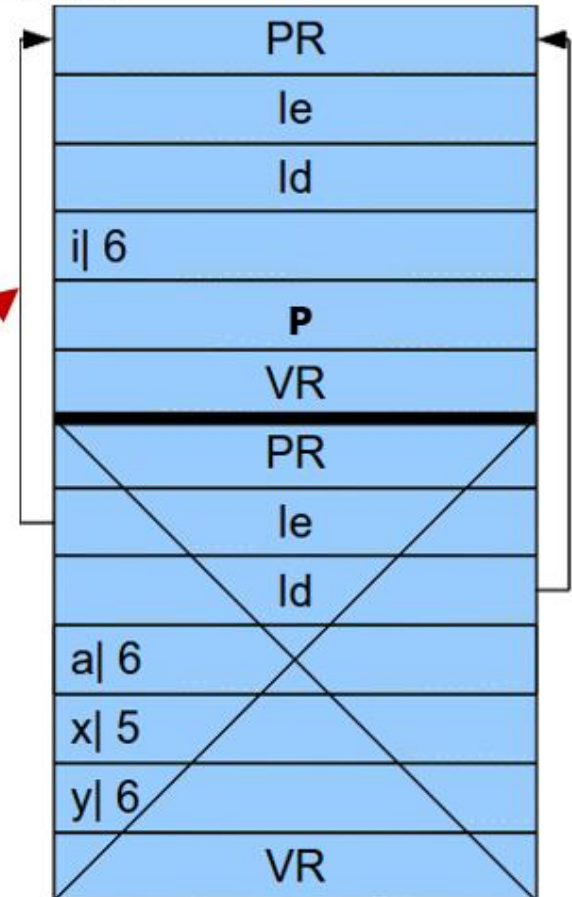
P(i);

Print(i);

End.

Imprime 6

RESULTADO FINAL

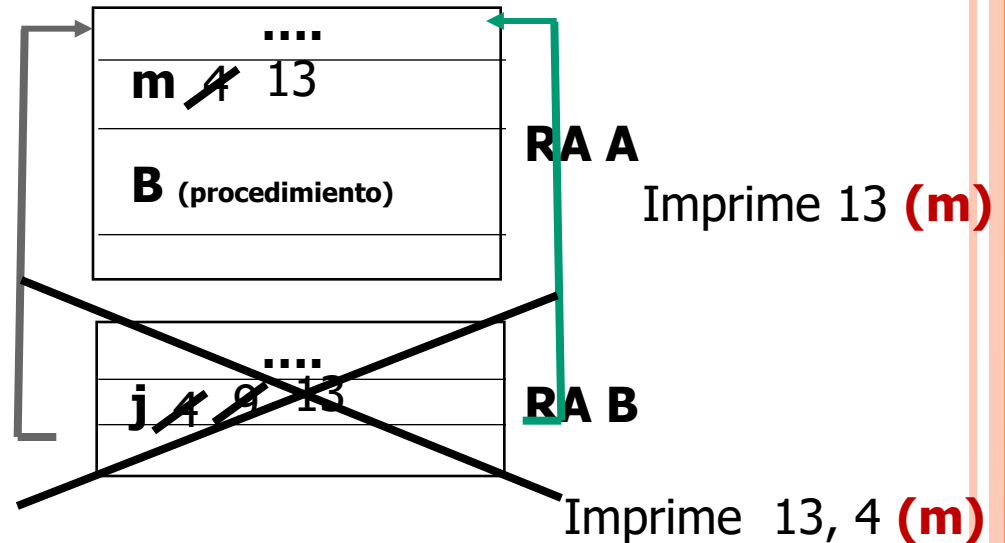


Se copia el valor al aloear el registro de la rutina, se modifica el parámetro formal, al finalizar la ejecución de la rutina se copia al parámetro real antes de desalocar.

Parámetros

Modos IN/OUT- Por Valor/Resultado

```
Procedure A ();  
var m:integer;  
Procedure B (valor-  
resultado j:integer);  
  begin  
    j:=j+5;    j:= j+ m;  
    write (j,m);  
  end;  
begin  
  m:=4;  B(m);  
  write (m);  
end;
```



Caso:

- **J es local**
- **m es "no local"**
- **busco por LE o LD**
- **m es modificada al final no en medio**

Se copia al aloca el registro y se modifica el parámetro formal al finalizar la ejecución de la rutina.

Parámetros

Datos - Modos IN/OUT— Por Referencia:

- También llamada por "variable"
- No es copia por valor es copia por referencia a una posición.
- Se asocia la dirección (*l-valor*) del PR al PF.
- La **conexión** es al inicio y permanece hasta el final
- El **PF** será una **variable local a su entorno** pero que contiene la **dirección** al **PR** de la unidad llamadora que estará entonces en un **ambiente no local**.
Así **se extiende el alcance de la rutina** (*aliasing situation*)
- Cada referencia al PF será a un ambiente no local, entonces **cualquier cambio** que se realice en el **PF** dentro del cuerpo del subprograma **quedará registrado en el PR**. El **cambio será automático**.
- El PR queda compartido por la unidad llamadora y llamada. Será **bidireccional**.

Parámetros

Datos - Modos IN/OUT– Por Referencia:

Desventajas:

- Se puede llegar a **modificar el PR inadvertidamente**. Es el **peor problema**. **Perdida de control y llevar a errores**
- El **acceso al dato** es **más lento** por la **indirección** a resolver cada vez que se invoque.
- Se pueden generar **alias cuando dos variables o referencias diferentes se asignen a la misma dirección de memoria**. (*aliasing*)

Estas cosas **afectan la legibilidad** y por lo tanto la **confiabilidad**, se hace muy **difícil la verificación** de programas y **depuración de errores**

Parámetros

Datos - Modos IN/OUT– Por Referencia:

Ventajas:

- **eficiente en espacio y tiempo** ya que **no se realiza copias de los datos** *sobre todo en grandes volúmenes de datos*
- **La indirección es de bajo costo, es fácil de implementar por muchas arquitecturas**

Parámetros

Datos – Modos IN/OUT– Por Referencia

Program main

```
var i:integer;
```

```
Procedura P(var a:integer)
```

```
    var x,y: integer;
```

```
Begin
```

```
    a=4
```

```
    x=a+1;
```

```
    y=x+1;
```

```
    a=y;
```

```
end;
```

```
Begin
```

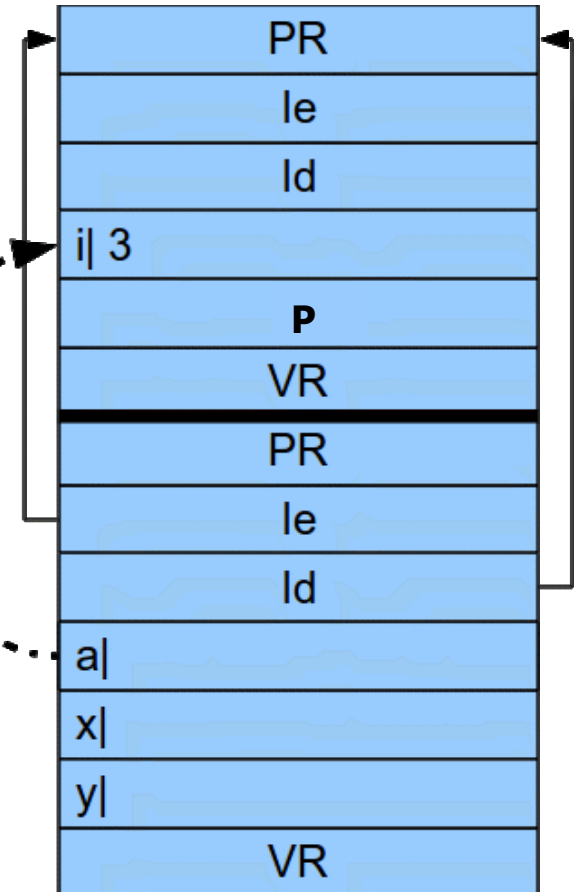
```
    i=3;
```

```
    P(i);
```

```
    Print(i);
```

```
End.
```

Imprime 6



Se trabaja directamente sobre la variable referenciada
Por Referencia también **es conocido por Variable (var)**

Parámetros

RESULTADO FINAL

Datos – Modos IN/OUT– Por Referencia

Program main

var i:integer;

Procedura P(var a:integer)

var x,y: integer;

Begin

a=4

x=a+1;

y=x+1;

a=y;

end;

Begin

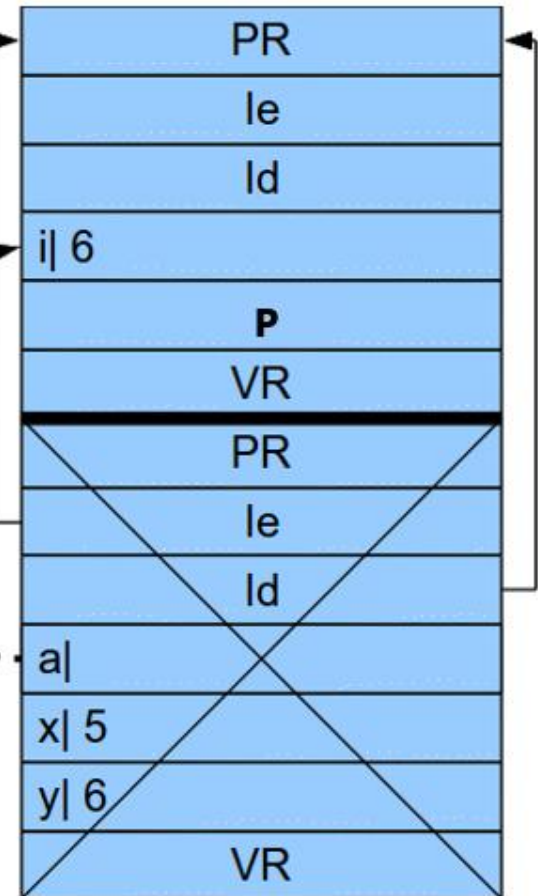
i=3;

P(i);

Print(i);

End.

Imprime 6

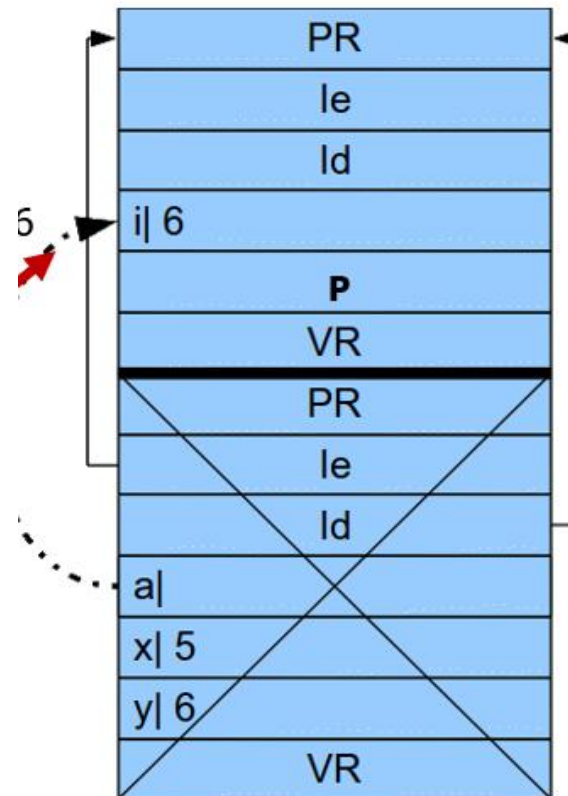
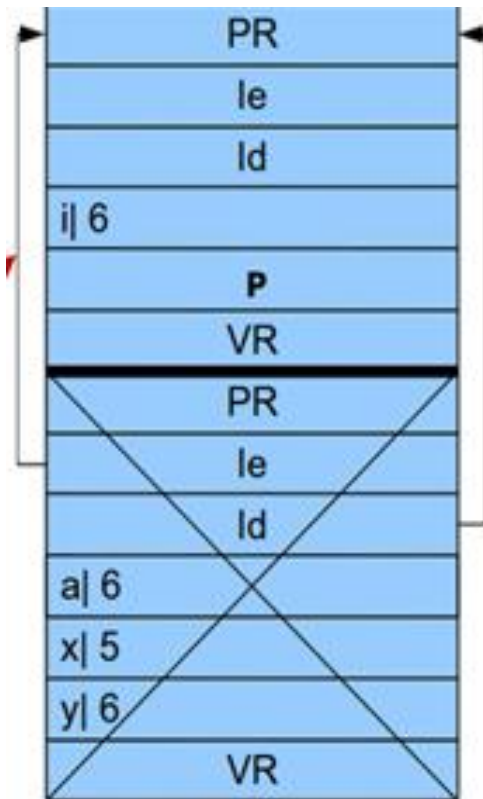


Se trabaja directamente sobre la variable referenciada
Por Referencia también **es conocido por Variable (var)**

Parámetros – Comparación

Por Valor/Resultado

Por Referencia



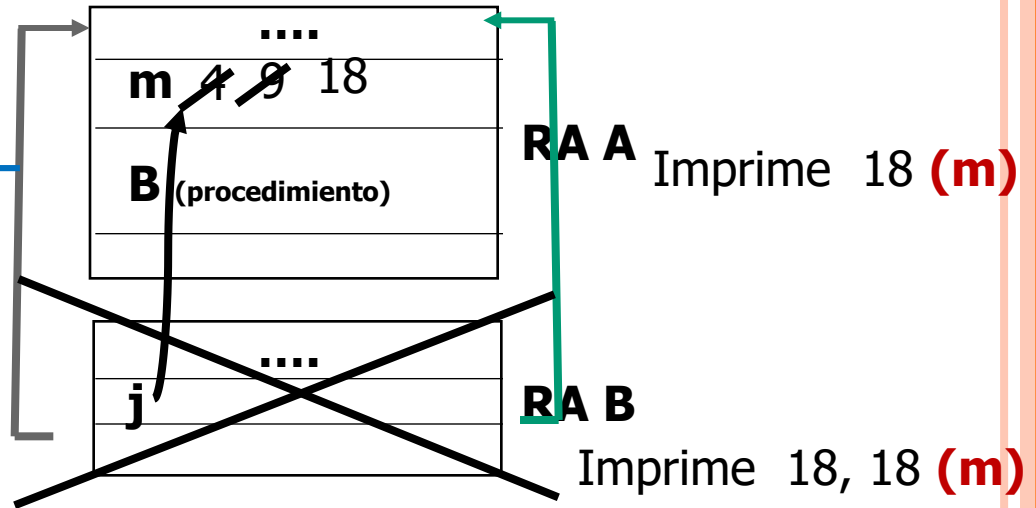
Comparación de ambas modalidades:

- 1. Mismo resultado**
- 2. Semánticamente distintos**

Parámetros

Datos – Modos IN/OUT- Por Referencia

```
Procedure A ();  
var m:integer;  
Procedure B (var j:integer);  
begin  
    j:=j+5;    j:= j+ m;  
    write (j,m);  
end;  
  
begin  
    m:=4;  B(m);  
    write (m);  
end;
```



Caso:

- J es local
- m es "no local"
- busco por LE o LD
- m si es modificada

J apunta a m por referencia
M de j+m la busca por link E o link D

Parámetros

Datos - Modo IN/OUT - Por Nombre:

- Introducido por ALGOL60. **No es utilizado** hoy en ningún **lenguaje imperativo**. Método conceptualmente **importante** por sus **propiedades e implementación**.
- **Solución propuesta** fue **definir la semántica** de la llamada a una función utilizando ***La Regla de la Copia***.
- La idea era **reemplazar todas las ocurrencias en la rutina por el parámetro real**. Pero no se puede hacer directo porque **podrían existir variables con el mismo nombre en ambos entornos** y esto alteraría el resultado. **Es necesario que el parámetro se evalúe en el entorno del llamador y no en el del llamado**.

La estrategia fue sustituir el parámetro formal por el parámetro real junto con su propio entorno de evaluación

Parámetros

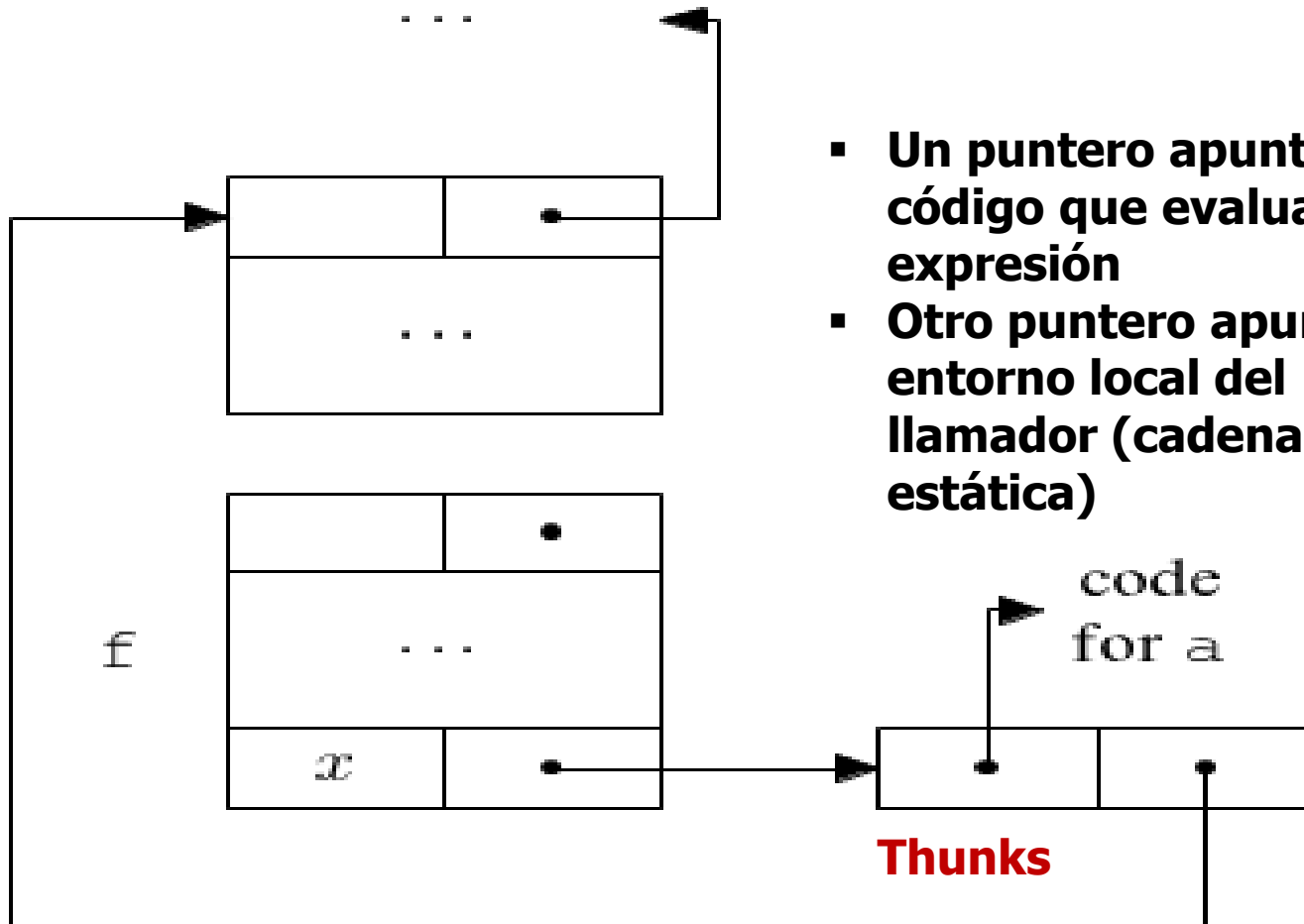
Datos - Modo IN/OUT - Por Nombre:

- **Parámetro formal** es sustituido "por una expresión textual" del parámetro real, más un "puntero al entorno" del parámetro real (*expresión textual, entorno real*). Se utiliza una **estructura** aparte que resuelve esto
- La **ligadura de parámetros** (entre parámetro formal y parámetro real) en el **momento de la invocación**
- La **"ligadura de valor"** se **difiere** hasta el momento en que **se lo utiliza** (*la dirección se resuelve en momento de ejecución*).
- **Semánticamente distinto a "Pasaje por Referencia"**
- El objetivo es otorgar **evolución de valor diferida**.

Parámetros

Datos - Modos IN/OUT– Por Nombre (continuación):

Implementación de llamada por nombre



- Un puntero apunta al código que evaluará la expresión
- Otro puntero apunta al entorno local del llamador (cadena estática)



Parámetros

Datos - Modos IN/OUT– Por Nombre (continuación):

- **Thunks:** es una es una **unidad pequeña de código (función) que encapsula y representa a una expresión que *pospone su evaluación hasta que sea necesario*.**
- **Es un concepto clave en la “*evaluación perezosa/diferida*”**
- **Para implementar el pasaje por Nombre se utilizan:**
 - los **thunks** que son **procedimientos sin nombre**.
 - Cada aparición del **parámetro formal se reemplaza** en el cuerpo de la unidad llamada **por una invocación** a un **thunks,**
 - en el **momento de la ejecución** activará al **procedimiento que evaluará el parámetro real en el ambiente apropiado.**

Parámetros

Datos – Modos IN/OUT– Por Nombre

Program main

```
var i:integer;
```

```
Procedure P(nombre a:integer)
```

```
    var vec[1..3] of integer;
```

```
Begin
```

```
    vec[1]=0;
```

```
    a=a-1;
```

```
    vec[i]=a;
```

```
    vec[a+1]=1;
```

```
end;
```

```
Begin
```

```
    i=3;
```

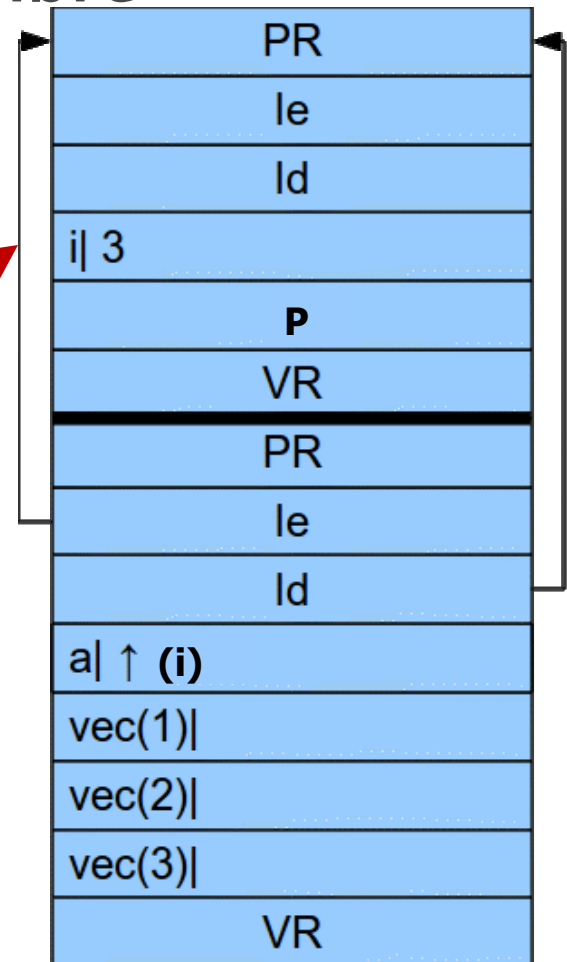
```
    P(i);
```

```
    Print(i);
```

```
End.
```

Imprime
2

Caso:
índice de un vector
como parámetro
por nombre



El PF es sustituido textualmente apuntando al PR
Nos preguntamos ¿que significa a?

Parámetros

Datos – Modos IN/OUT– Por Nombre

RESULTADO FINAL

Program main

var i:integer;

Procedure P(nombre a:integer)

var vec[1..3] of integer;

Begin

vec[1]=0;

a=a-1;

vec[i]=a;

vec[a+1]=1;

end;

Begin

i=3;

P(i);

Print(i);

End.

Imprime
2

**Caso:
índice de un vector
como parámetro
por nombre**

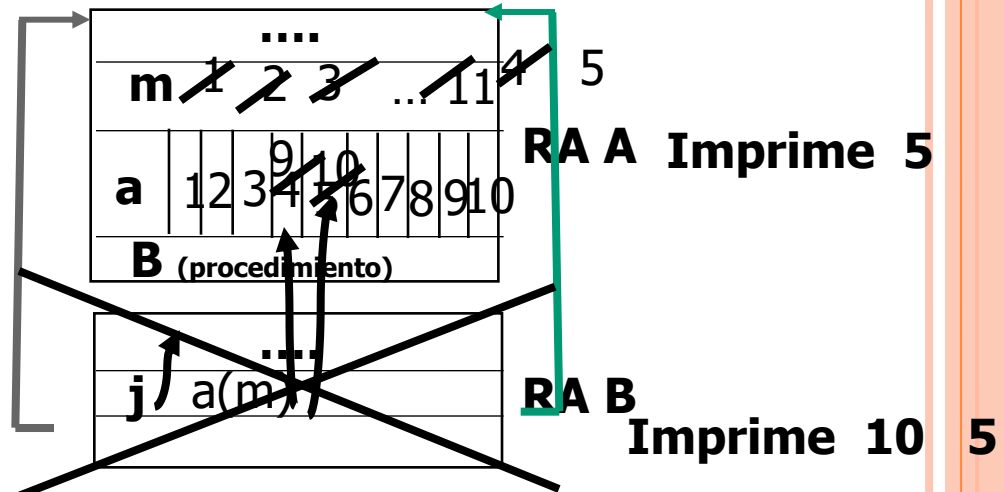
PR
le
ld
i 2
P
VR
PR
le
ld
a ↑ (i)
vec(1) 0
vec(2) 2
vec(3) 1
VR

El PF es sustituido textualmente apuntando al PR
Nos preguntamos ¿que significa a?

Parámetros

Datos – Modos IN/OUT–
Por Nombre

```
Procedure A ();  
  var m:integer;  
  a: array[1..10] of integer;  
Procedure B (nombre j:integer);  
  begin  
    j:=j+5;  
    m:=m+1;  
    j:= j+ m;  write (j,m);  
  end;  
begin  
  for m=1 to 10 begin  a[m]=m;  end;  
  m:=4;  B(a[m]);  
  write (m);  
end;
```



Caso:
un vector como
parámetro por
nombre

Se calcula la dirección
durante la ejecución

Es fundamental
preguntarse ¿qué
significa "a(m)" en el
registro de activación del
"llamante".

Este es el secreto del
parámetro por nombre.

Parámetros

Datos - Modos IN/OUT– Por Nombre (continuación):

- **Desventajas y ventajas:**
 - Es un **método** que **extiende el alcance del procedimiento al entorno del parámetro real**, esto puede llevar a errores.
 - **Posee evaluación diferida al ejecutar**
 - Es más **lento** por **evaluación repetida del parámetro real**, debe **evaluarse cada vez que se lo usa**. (ej. Pensar si es un loop)
 - Puede existir **aliasing**
 - **Difícil de implementar** y genera **soluciones confusas** para **el lector y el escritor**.

Parámetros

■ Pasaje de parámetros en algunos lenguajes:

■ C:

- **Por valor**, (si se necesita ***por referencia se usan punteros***).
- permite pasaje **por valor constante**, agregándole ***const***

■ Pascal:

- **Por valor** (por defecto)
- **Por referencia** (opcional: **var**)

■ C++:

- Similar a C
- Más pasaje **por referencia**

■ Java:

- Sólo **copia de valor**. Pero como las variables de tipos **no primitivos** son **todas referencias** a variables anónimas en el HEAP, el paso por valor de una de estas variables constituye en realidad un **paso por referencia** de la variable.
- La **estructura de datos primitiva** es un tipo de estructura de datos que **almacena datos de un solo tipo**. Ejemplos ***son los enteros, los caracteres y los flotantes, etc.***
- La estructura de datos **no primitiva** es un tipo de estructura de datos que **puede almacenar datos de más de un tipo**. Ejemplos ***son los arrays, TAD, listas, etc.***

Parámetros

- **Pasaje de parámetros en algunos lenguajes (continuación):**
 - **PHP:**
 - **Por valor**, (predeterminado).
 - **Por referencia** (&)
 - **RUBY:**
 - **Por valor.** Pero al igual que Python si se pasa es un objeto “**mutable**” (objeto que puede ser modificado), no se hace una copia, sino que se trabaja sobre él.
 - **Python:**
 - Envía objetos que pueden ser “**inmutables**” o “**mutables**” (objeto que pueden ser o no modificados). Si es **inmutable** actuará como **por valor** y, si es **mutable**, ejemplo: listas, no se hace una copia, sino que **se trabaja sobre él**.

Parámetros

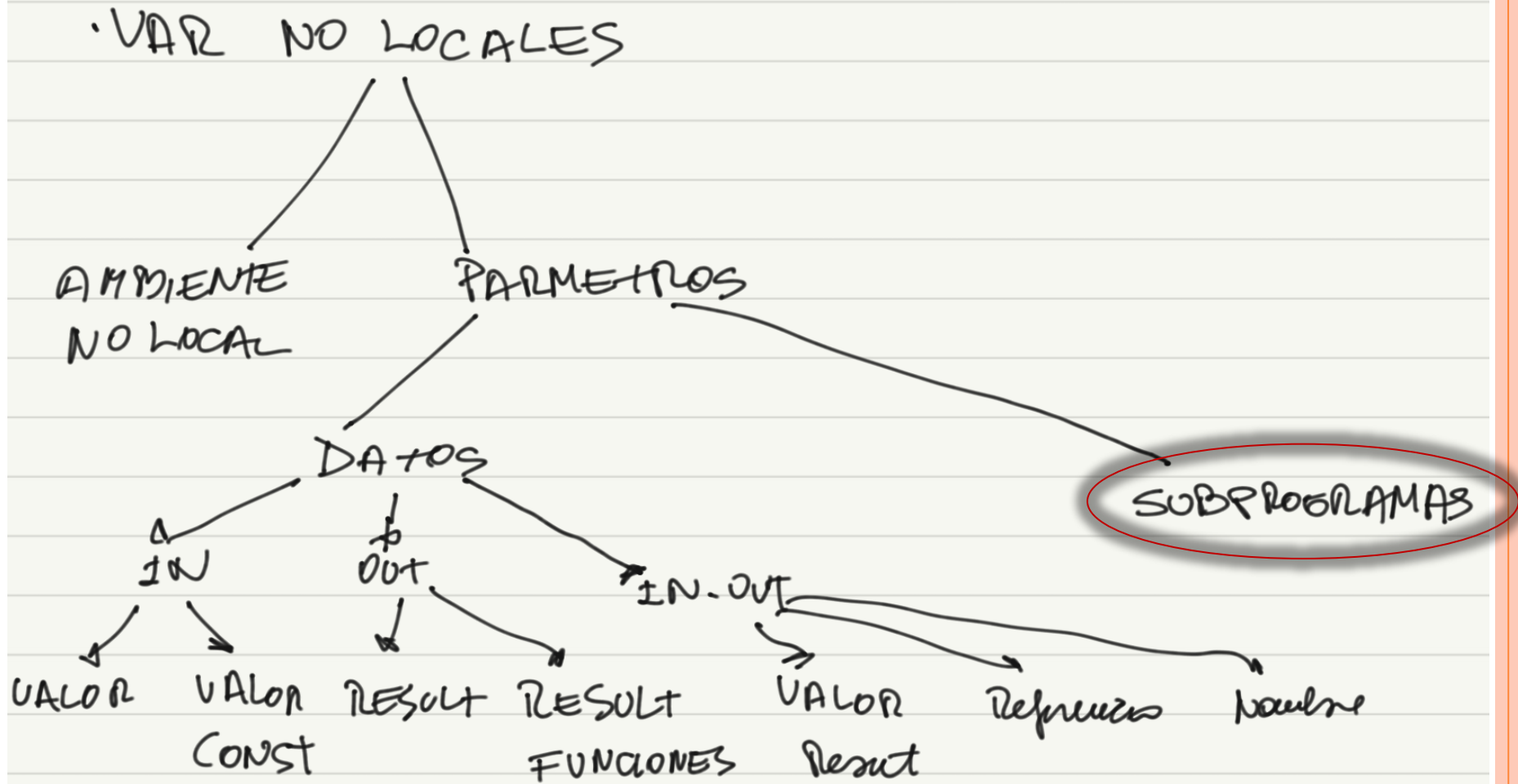
- **Pasaje de parámetros en algunos lenguajes (continuación):**
 - **ADA:** usa varios
 - **Por copia modo IN** (por defecto)
 - **Por resultado modo OUT**
 - **IN-OUT.**
 - Para los tipos de datos **primitivos** indica que es por **valor-resultado**
 - Para los tipos **no primitivos**, y datos compuestos (arreglos, registros) se hace por **referencia**

Los tipos primitivos de datos proveídos por Ada son seis

1. **Integer** : Tipo que abarca los enteros positivos y negativos Natural : Este tipo de dato contiene números positivos más el cero.
2. **Positive**: Solamente permite números positivos.
3. **Float**: Tipo que almacena cualquier número real
4. **Char** : Guarda un carácter.
5. **String**: Almacena un número definido de caracteres .

- Particularidad de ADA:
 - **En las funciones** solo se permite el paso por **copia de valor**, lo cual **evita** parcialmente la posibilidad de **efectos colaterales**.

Formas de conectar y "compartir datos" entre diferentes Unidades de Programa:



Parámetros

2) Parámetro Subprogramas:

1. Rutinas que reciben funciones como parámetros

- **Bastante común** en la mayoría de los lenguajes

2. Rutinas que devuelven funciones cómo resultado

- Mecanismo **fundamental** en **lenguajes funcionales**.
- **Poco común** en otros lenguajes.

Parámetros

2) Subprogramas como Parámetro:

Hay **situaciones** en que es conveniente o necesario **poder usar nombres de subprogramas como parámetro** para **ejecutar alguna acción**.

Ejemplos:

- Un **programa** que trabaja con **un arreglo al que se le debe aplicar distintos procesos de ordenamiento**. El **nombre del procedimiento que ordena sería un parámetro más**.
- Un **lenguaje** que **no proporciona manejo de excepciones**, el **nombre de la rutina** que haría las veces del **manejador** podría **ser un parámetro subprograma**.
- En **general** se usa con **funciones matemáticas**

Parámetros

2) Subprogramas como Parámetro:

Rutinas que reciben funciones como parámetros

Caso: Se pasa una función $f()$ como parámetro a otra función $g()$

- ¿qué entorno se utiliza cuando $f()$ accede a variables no locales?
- ¿si hay variables con el mismo nombre que están en diferentes entornos?
- ¿si hay entornos anidados? **¿cuál usa?**

Algunas cuestiones que se deben resolver

- chequeo de tipos de subprogramas
- subprogramas anidados
- ¿Cuál es el entorno de referencia?

Parámetros

- **Ambiente de referencia para las referencias no locales dentro del cuerpo del subprograma pasado como parámetro.**

- Debe **determinarse** cuál es el **ambiente de referencia no local correcto para un subprograma** que se ha **invocado** y que ha sido **pasado como parámetro**.

Cuando un procedimiento que se pasó como parámetro se ejecuta, puede tener problemas con referencias que hay dentro del procedimiento.

- **En general:** si hay **una referencia a una variable que no pertenece a él y no es local** entonces **surge el problema a dónde va a buscarla** porque este **es un procedimiento que se mandó como parámetro**
 - Si el lenguaje sigue la **cadena estática** para buscar su referencia **se pregunta dónde está contenido este procedimiento**
 - Si el lenguaje sigue la **cadena dinámica** entonces **se pregunta quién llamó a este procedimiento**.

Parámetros

- **Ambiente de referencia para las referencias no locales dentro del cuerpo del subprograma pasado como parámetro.**

Que pasa si se dan estas condiciones:

- 1. Se pasan funciones como argumentos*
- 2. Esas funciones tienen referencias a variables definidas fuera de ellas (variables libres)*
- 3. Hay funciones anidadas.*

¿Cómo saber a donde ir a buscar las variables?

Hay 3 reglas de alcance y binding de parámetros

Parámetros

- Ambiente de referencia para las referencias no locales dentro del cuerpo del subprograma pasado como parámetro.

Tres reglas:

- Ligadura shallow (*superficial*)
- Ligadura deep (*profunda*)
- Ligadura ad-hoc

Parámetros – Ejemplo en JS

```
function sub1() {  
  var x;  
  function sub2() {  
    alert(x); //Creates a dialog box with the value of x  
  };  
  function sub3() {  
    var x;  
    x = 3;  
    sub4(sub2);  
  };  
  function sub4(subx) {  
    var x;  
    x = 4;  
    subx();  
  };  
  x = 1;  
  sub3();  
};  
sub1();
```

**Consideremos la ejecución de sub2
cuando se llama en sub4.
Cual es el valor de X?**

Shallow/Superficial: Busca donde *se ejecutará lanzará* la *función pasada como parámetro* **function**

sub4 (subx)

Mira la Unidad donde está declarado el subprograma que contiene a la función cómo parámetro formal (subprograma que lo invoca):
El ambiente de referencia de esa ejecución es el de **sub4**, por lo que la referencia a x en sub2 está vinculada a la x local en sub4, y **la salida del programa es 4.**

Deep/Profunda: Se busca en el entorno donde esta *definida/declarada la función pasada como parámetro* **function sub2()**

Mira la unidad donde está declarado el subprograma parámetro.
El entorno de referencia de la ejecución de **sub2** es el de **sub1**, entonces la referencia a x en sub2 está vinculada a la x local en sub1, y **la salida es 1.**

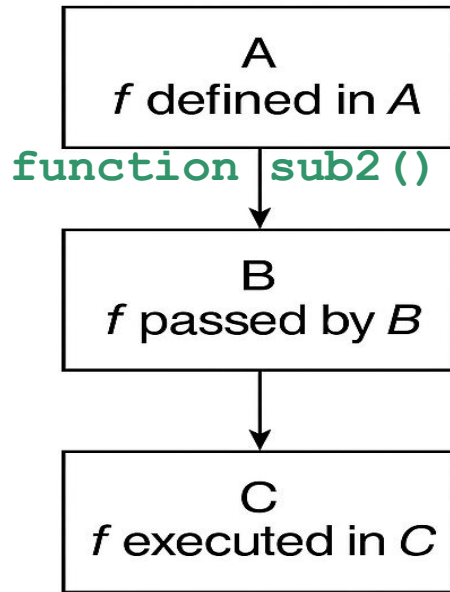
Ad hoc: Se busca en el entorno *donde se pasa la función cómo parámetro* **sub4 (sub2)**

El ambiente de referencia es el del subprograma donde se encuentra el llamado a la unidad que tiene un parámetro de tipo Subprograma. El ambiente de referencia es el de la x local **en sub3**, y **la salida es 3.**

Parámetros

- Ambiente de referencia para las referencias no locales dentro del cuerpo del subprograma pasado como parámetro.

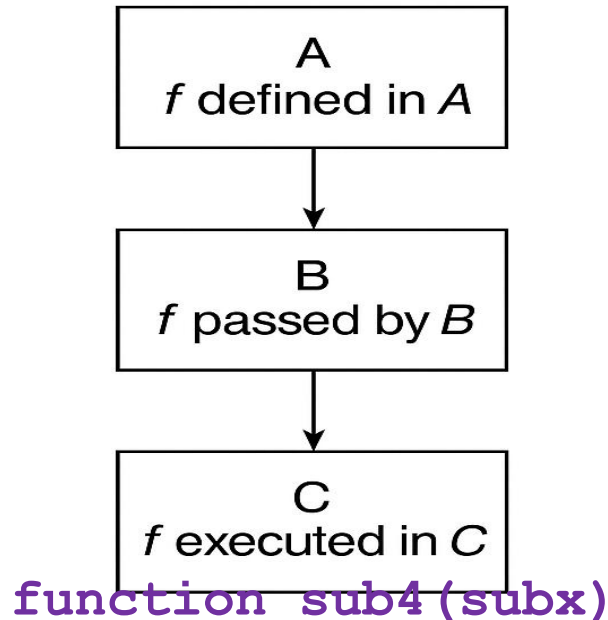
Deep binding



Referencing
environment: A

**Parecido a donde
está contenida,
pero Deep Binding
no es alcance
estático**

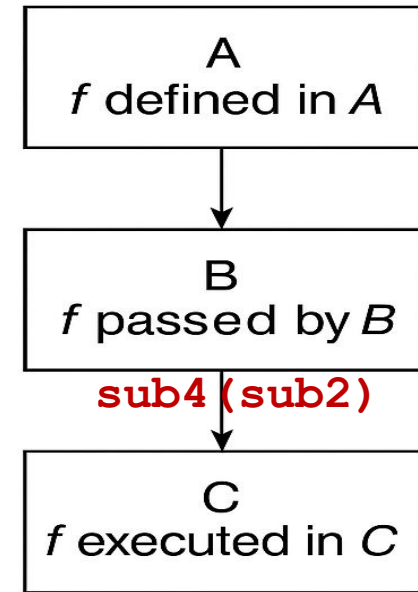
Shallow binding



Referencing
environment: C

**Parecido a quién
me llamó, pero
Shallow Binding
no es alcance
dinámico**

Ad hoc binding



Referencing
environment: B



Parámetros

2) Subprogramas como Parámetro:

Rutinas que reciben funciones como parámetros

- **No** lo incorporan **todos los lenguajes**
- **Algunos** ofrecen **otras soluciones sintácticas y semánticas**

Pascal

- permite que la **referencia a un procedimiento sea pasada a un subprograma**

C

- **permite punteros a funciones, no permite funciones anidadas**

Ada

- **No contempla los subprogramas como valores.**
- Utiliza **unidades genéricas** en su defecto (se verá más adelante)

Bibliografía

- **GHEZZI C. - JAZAYERI M.: Programming language concepts. John Wiley and Sons. (1998) 3er. Ed**
- **SEBESTA: Concepts of Programming languages. Benjamin/Cumming. (2010) 9a. Ed.**
- **David A. Watt, William Findlay: Programming language design concepts (2004)**
- **Maurizio Gabbrielli and Simone Martini: Programming Languages: Principles and Paradigms (2010)**