



SEMANTICA OPERACIONAL

Formas de comunicación entre las rutinas

Rutinas

- Una **Unidad de Programa** (*función, procedimiento, subprograma.....*)
- Formadas por un **conjunto de sentencias** que representan **acción abstracta**.
- A nivel de diseño **permiten definir una operación creada por el usuario** a semejanza de las operaciones primarias integradas en el lenguaje
- Entonces, permiten **ampliar** a los lenguajes, dan **modularidad, claridad y buen diseño**.
- Los **subprogramas** son el ejemplo más usual y útil presente desde los primeros lenguajes ensambladores **con una llamada explícita y luego retornan** a algún punto de la ejecución (responden al ***esquema call/return***)

Rutinas

- Formas de **Subprogramas**
 - **Procedimientos**
 - **Funciones**

Semánticamente distintos.....

Rutinas

■ Formas de **Subprogramas**

■ **Procedimientos**

- En su concepción más simple, 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.
- Usando procedimientos la solución de **código** es más corta, **comprensible y fácilmente modificable**.
- Permiten al **programador definir y crear nuevas acciones agrupando sentencias**.
- **Pueden no recibir ni devolver ningún valor**.
- Los resultados los produce en **variables no locales** o en **parámetros** que cambian su valor.

Rutinas

■ Formas de **Subprogramas**

■ **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 produce reemplaza a la invocación dentro de la expresión.**
- **Siempre deben retornar un valor.**

Rutinas

■ Conclusiones:

- Cuando se diseña un subprograma **el programador se concentra en el cómo** trabaja dicho **subprograma**.
- Cuando se usa el subprograma se ignorará el cómo y **se concentra en qué permite hacer. La implementación permanece oculta.**
- 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:

- Si las unidades utilizan **variables locales** no hay problema
- Si las variables **no son locales** hay 2 formas:
 - 1. *A través del acceso al ambiente no local*** (entonces hay variables que no son locales, puede llevar a más errores)
 - 2. *A través del uso de parámetros*** (es la forma mejor y más clara)

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

1) A través del **acceso al ambiente no local**

■ **Ambiente no local implícito**

- Es automático
- Utilizando *regla de alcance dinámico*
- Utilizando *regla de alcance estático*

■ **Ambiente común explícito**

- Se definen áreas comunes de código
- El programador debe especificar que comparte
- Cada lenguaje tiene su forma de realizarlo
- Ej: uso de paquetes PACKAGE (TDA) de ADA, variables externas en PL/1, cláusula COMMON en PASCAL, etc.)

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.

Que valores

resultarán?

todos usan x z n

Main llama A2(),

A2() llama a A3(),

A3() llama a A1(),

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

- Var locales (que hay problemas)

- Var No locales

USO AMBIENTE
NO LOCAL

USO PARAMETROS

Subrute
NO local
EXPLICITO

AMBIENTE
común
explícito

Área
estática

Área
DINAMICA

ÁREAS
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** utilizada para **pasar** a un **procedimiento o función**.
 - Están en la parte de la invocación
- ***Parámetro Formal (Parámetro)***: es una **variable** utilizada para **recibir valores de entrada** en una **rutina, subrutina etc.**
 - Se ponen en la parte de la declaración

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

*El **pasaje de parámetros** es mejor, ya que el uso intensivo de **accesos al ambiente no local** puede provocar alguna pérdida de control y que **las variables terminan 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

- Da **distintas posibilidades de compartir cosas** (que veremos más adelante)
- Permite **enviar distintos parámetros en distintas invocaciones** a las rutinas
- **Más flexibilidad**, se pueden transferir **más datos y de diferente tipo en cada llamada**
- Permite **compartir en forma más abstracta**, solo especificamos el **nombre a argumentos y parámetros**) y el **tipo** de cada cosa que se comparta
- Nos adecuamos a las reglas de los lenguajes

Ventajas del Pasaje de Parámetros

- Proporciona **ventajas en protección**: el uso intensivo de **accesos al ambiente no local decrementa la seguridad y legibilidad** de las soluciones ya que *las variables terminan siendo visibles aun donde no es necesario o donde no debería*.
- Proporciona **ventajas en legibilidad**: Permite al programador **encontrar más fácilmente los errores**. Transformo en rutinas o funciones a los que les paso valores, es más fácil depurar y encontrar errores y no chequear cada repetición en el código.
- Esto le **da modificabilidad**, si hay errores uno se focaliza en **qué cosas estoy compartiendo, que argumentos y parámetros estoy utilizando y su tipo**.

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

Parámetros

Parámetros **formales** en declaración

```
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.
```

Parámetros **reales** en invocación

¿Los parámetros formales son variables locales?

¿Qué datos pueden ser los parámetros reales?

Parámetros

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

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

Se declara con una sintaxis particular al lenguaje, y sirve para intercambiar información entre la función que hace la llamada y la función que la recibe.

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

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

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

Parámetros

- **Evaluación de los parámetros reales y ligadura con los parámetros formales**
 - **Evaluación:**
 - En general **antes de la invocación** primero se **evalúan** los **parámetros reales**, y **luego** se hace la **ligadura**. Se verifica que todo está bien antes de transferir el control a la unidad llamada.
 - **Ligadura:**
 - **Posicional:** Se **corresponden con la posición** que ocupan en la lista de parámetros. **Van en el mismo orden**
 - **Palabra clave o nombre:** 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

Ejemplo:

- en **Phyton light llamo** con ligadura por nombre

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

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

llamo P(4,Z) y recibe P(Z,4)

- **Desventaja**, cuando invocas **hay que conocer 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

- Cada lenguaje es distinto y hay que conocerlos para evitar errores.
 - En **Ada** pueden **mezclarse** ambos **métodos**.
 - En **C++**, **Ada**, **Python**, **JavaScript**, los **parámetros formales** pueden tener **valores por defecto**, **no es necesario listarlos todos en la invocación**.
 - Se realiza una **asignación** de esos valores a los parámetros **en la cabecera**

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

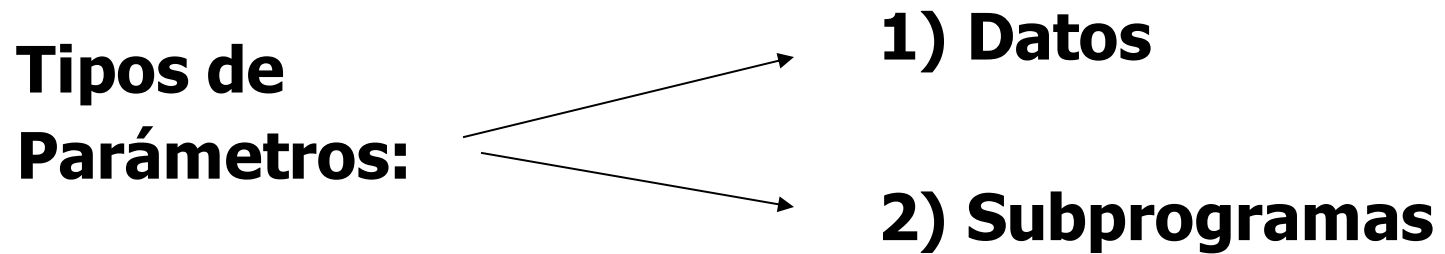
Ej. JavaScript

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



- Cada uno tiene distintos subtipos

Parámetros

1) Datos como Parámetros

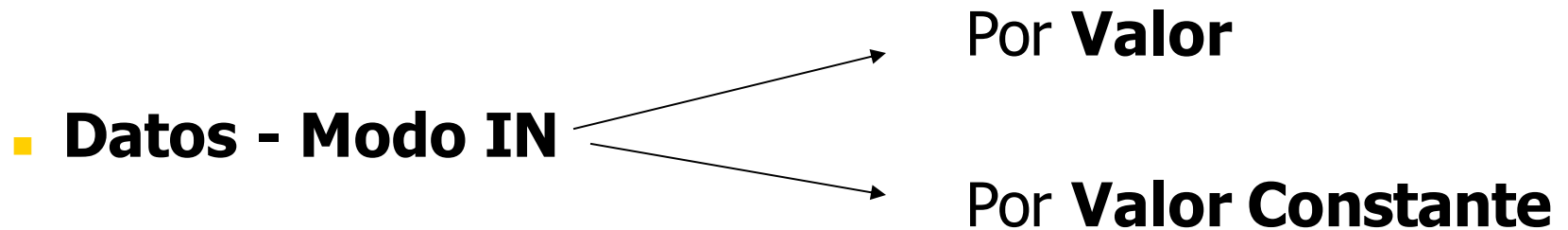
*Hay diferentes **formas de transmitir los parámetros hacia y desde el programa llamado.***

Desde el punto de vista semántico de los parámetros formales pueden ser:

- Datos**
- **Modo IN:** El *parámetro formal* recibe el dato desde el *parámetro real*. (***la conexión es al inicio 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**, 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;
```

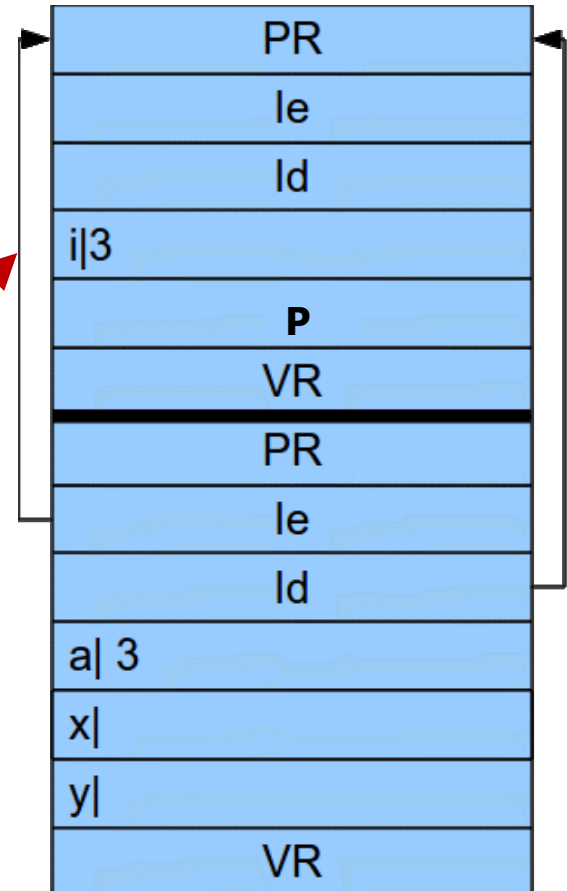
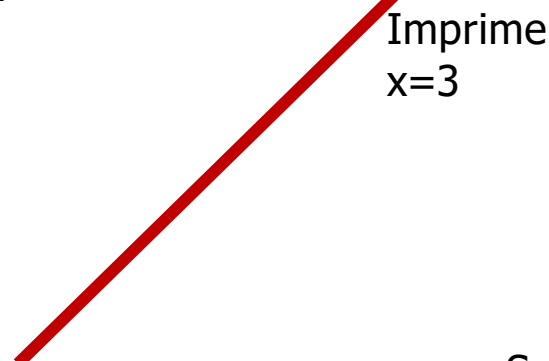
```
Begin
```

```
    i=3;
```

```
    P(i);
```

```
    Print(i);
```

```
End.
```



Se copia al aloca el registro en memoria

Parámetros

Datos - Modo IN - Por valor constante:

- Se **envía un valor**, pero **la rutina receptora no puede modificarlo**, es decir queda con un valor fijo que no se puede cambiar.
- No indica si se realiza o no la copia.
- **La implementación debe verificar** que el **parámetro real no sea modificado**.
- Algunos lenguajes que permiten el modo IN con pasaje por valor constante. **(no todos lo tienen)**

Ejemplos:

ADA define que parámetros modo IN es por valor constante

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:

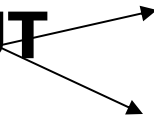
Desventaja:

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**
 - Resultado de funciones**

Conexión al final

Parámetros

• Datos - Modo OUT - Por Resultado:

- El valor del **parámetro formal** se **copia al parámetro real al terminar de ejecutarse la unidad que fue llamada** (rutina).
- El **parámetro formal es una variable local**
- El **parámetro formal** es una variable **sin valor inicial** porque no recibe nada.
Debemos inicializar de alguna forma si el lenguaje no lo hace por defecto. Esto puede traer **problemas**.

Parámetros

• Datos - Modo OUT - Por Resultado:

■ ***Desventajas:***

- **Consume tiempo y espacio** porque **hace copia al final**
- **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** en la ejecución de la unidad llamada

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

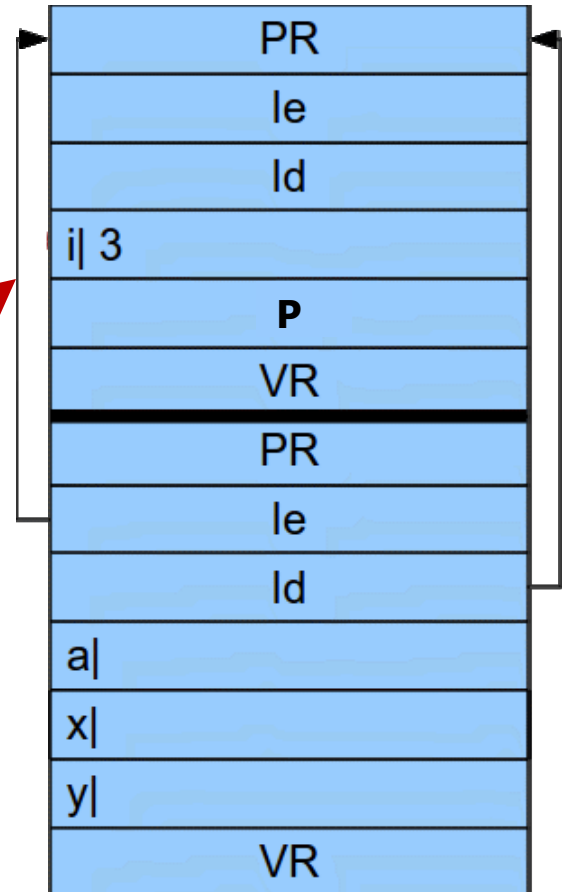
i=3;

P(i);

Print(i);

End.

Imprime 5



Se copia el valor del parámetro al desalocar el registro de memoria 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.
- **Remplaza la invocación en la expresión que contiene el llamado.**
- Puede devolverse de distintas formas según lenguaje:
 - ***return*** como en Python, C, etc.,
 - ***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:

El **parámetro formal** es una **variable local** que **recibe una copia a la entrada** del contenido **del parámetro real** y **a la salida el parámetro real recibe una copia** de lo que tiene el **parámetro formal**.

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 lo 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
```

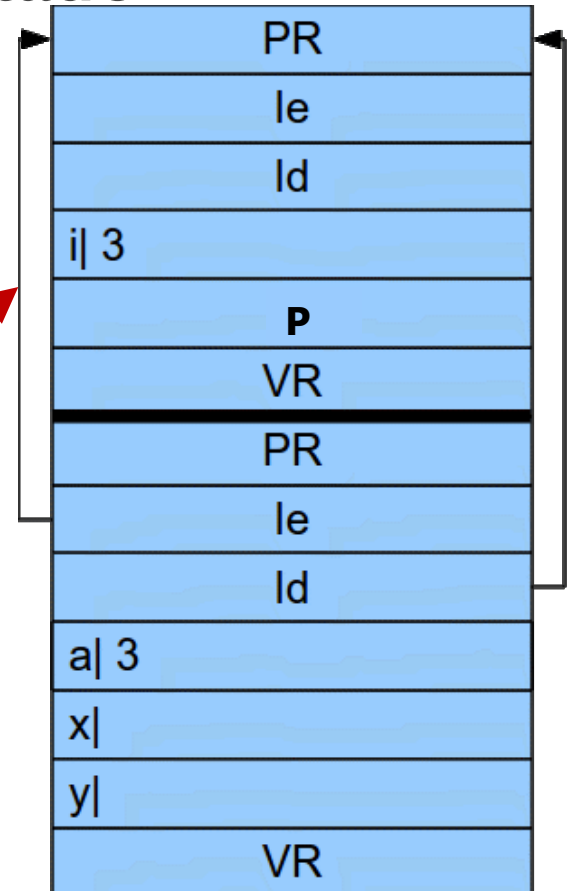
```
    i=3;
```

```
    P(i);
```

```
    Print(i);
```

```
End.
```

Imprime 6

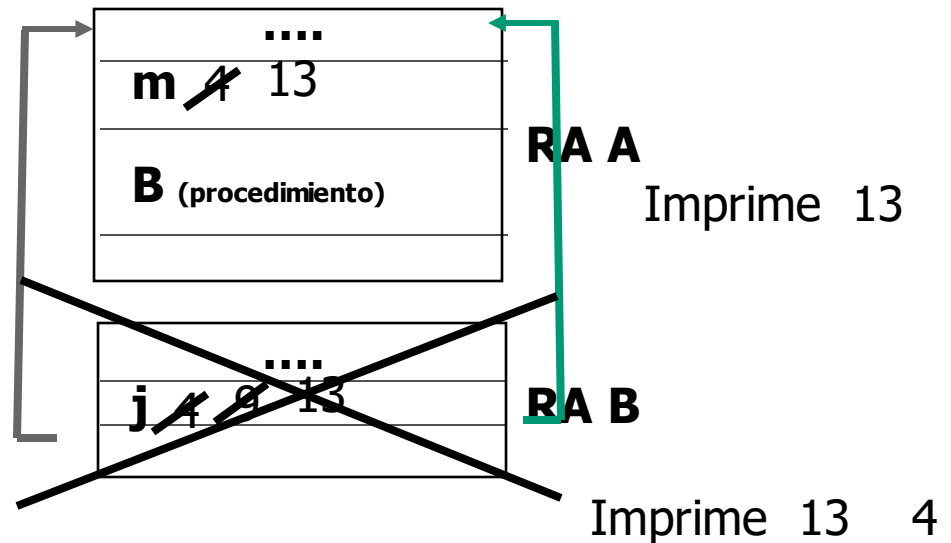


Se copia al allocar el registro y se modifica el parámetro formal al finalizar la ejecución de la rutina.

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;
```



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:

- **No es copia por valor es por referencia, se asocia la dirección (I-valor) del parámetro real al parámetro formal.**
- **Hay una asociación entre el PF y el I-valor del PR (aliasing situation)**
- **La conexión es al inicio y permanece hasta el final**
- **El parámetro formal será una variable local que contiene la dirección al parámetro real de la unidad llamadora que estará entonces en un ambiente no local.**
- **Cada referencia al parámetro formal será a un ambiente no local, entonces cualquier cambio que se realice en el parámetro formal dentro del cuerpo del subprograma quedará registrado en el parámetro real. El cambio será automático.**
- **El parámetro real queda compartido por la unidad llamadora y llamada. Será bidireccional.**

Parámetros

Datos - Modos IN/OUT– Por Referencia:

Desventajas:

- El **acceso al dato** es **más lento por la indirección** a resolver cada vez, pero de **bajo costo de implementar por muchas arquitecturas**.
- Se puede llegar a **modificar el parámetro real inadvertidamente**
- Se generan **alias** y estos **afectan la legibilidad** y por lo tanto la **confiabilidad**, se hace muy **difícil la verificación** de programas y **depurar errores**

Ventajas:

- **Eficiente en tiempo y espacio** en cuanto que **no se realizan copias de los datos**, ayudan en grandes volúmenes de datos

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
```

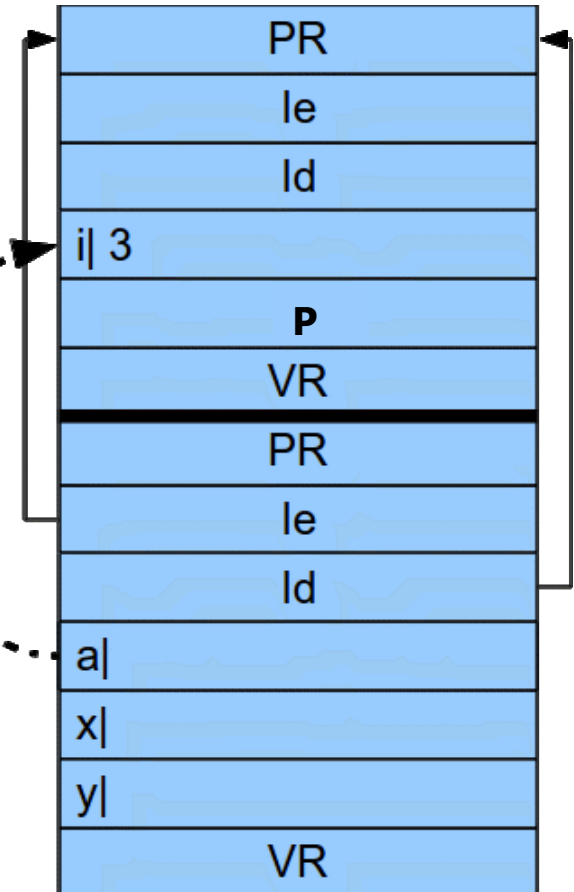
```
    i=3;
```

```
    P(i);
```

```
    Print(i);
```

```
End.
```

Imprime 6



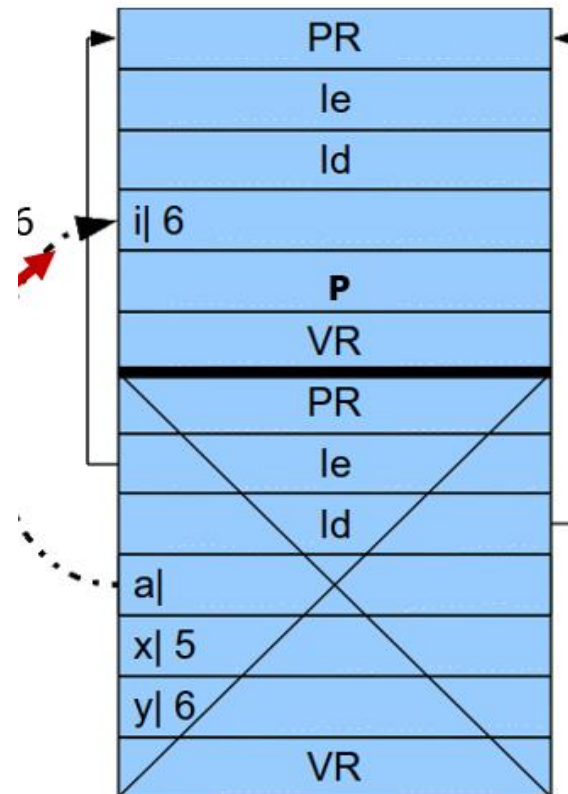
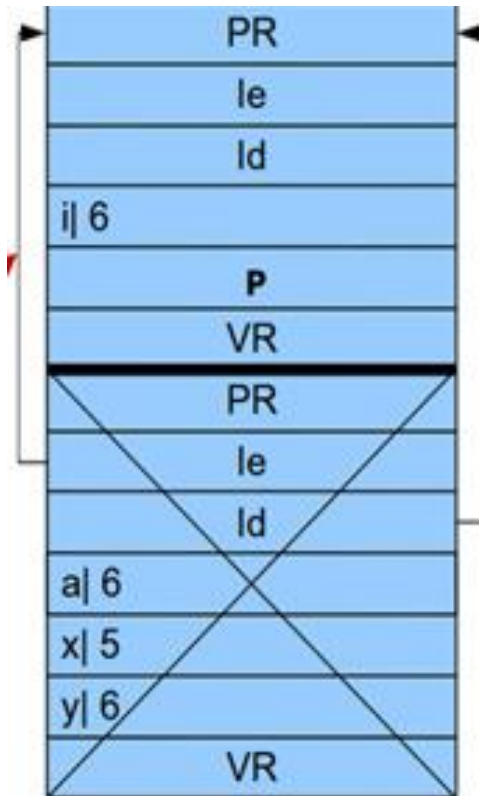
Se trabaja directamente sobre la variable referenciada

Por Referencia también es conocido por Variable

Parámetros – Comparación

Por Valor/Resultado

Por Referencia

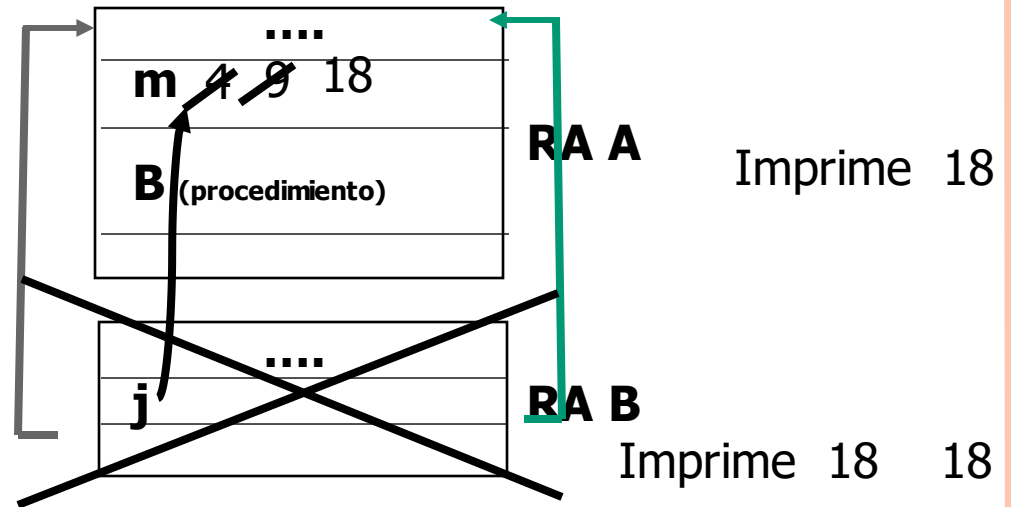


Comparación de ambas modalidades:
Mismo resultado
Semánticamente distinto

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;
```



Parámetros

Datos - Modo IN/OUT - Por Nombre:

- El **parámetro formal** es **sustituido textualmente por una expresión del parámetro real más un puntero al entorno del parámetro real**. (se maneja una estructura aparte que resuelve esto, pero no se verá en detalle en la materia)
- Se establece **la ligadura entre parámetro formal y parámetro real** en el momento de la **invocación**, pero la **"ligadura de valor"** se **difiere** hasta el momento en que se lo utiliza (la dirección se resuelve en **ejecución**). ***Distinto a por referencia***
- El objetivo es otorgar **evolución de valor diferida**.
- Si el dato a compartir es:
 - Un único valor se comporta exactamente igual que el pasaje por referencia.
 - Si es una constante es equivalente a por valor.
 - Si es un elemento de un arreglo puede cambiar el suscripto entre las distintas referencias
 - Si es una expresión se evalúa cada vez

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
```

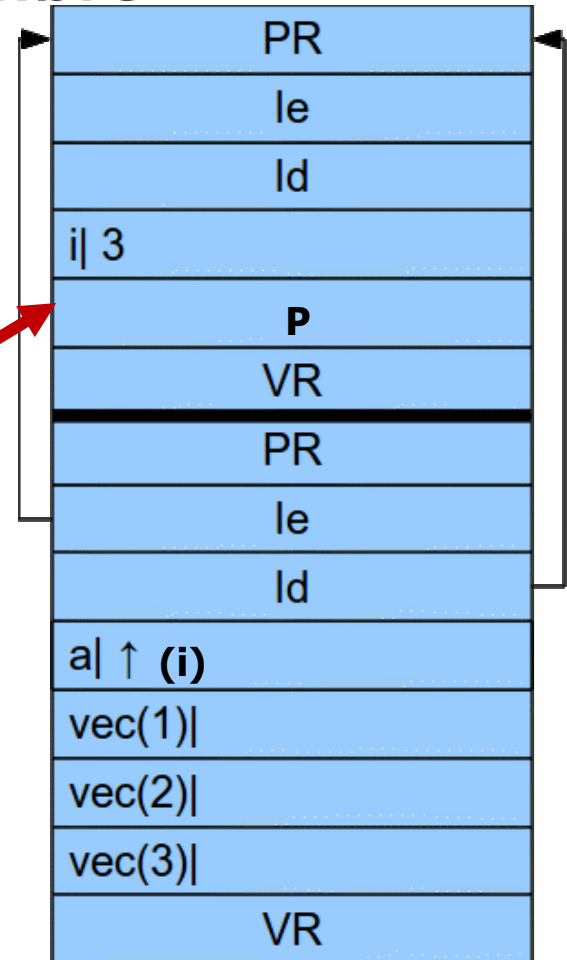
```
    i=3;
```

```
    P(i);
```

```
    Print(i);
```

```
End.
```

Imprime
2



El parámetro formal es sustituido textualmente apuntando al parámetro real. 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;

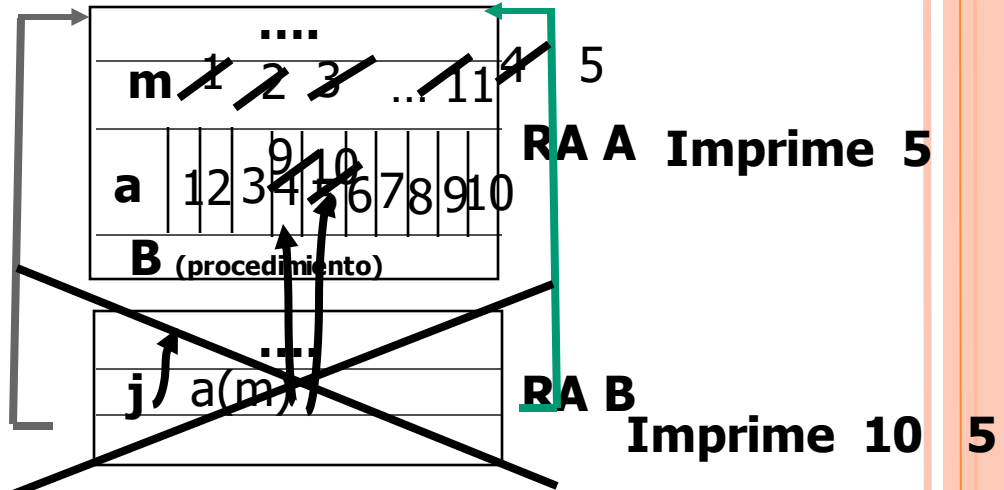
begin

```
for m=1 to 10 begin    a[m]=m;  end;
```

m:=4; B(a[m]);

```
write (m);
```

end;



**Es fundamental preguntarse
¿qué significa "a(m)" en el
registro de activación del
"llamante". Este es el secreto del
parámetro por nombre.
Calcula la dirección durante la
ejecución**

Parámetros

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

- **Thunks**
- Para **implementarlo** 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** que 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 (continuación):

- **Desventajas y ventajas:**
 - Es un método más **flexible** porque **extiende el alcance del parámetro real**, pero esto mismo **puede llevar a errores**.
 - **Posee evaluación diferida al ejecutar**
 - Es más **lento** ya que debe **evaluarse cada vez que se lo usa**. (ej si es un loop se evalúa cada vez)
 - Es **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**. La estructura de datos **no primitiva** es un tipo de estructura de datos que **puede almacenar datos de más de un tipo**. Ejemplos de ***estructuras de datos primitivas son los enteros, los caracteres y los flotadores.***

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 **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**.

VAR NO LOCALES

AMBIENTE
NO LOCAL

PARAMETROS

DATOS

SUBPROGRAMAS

IN

OUT

IN-OUT

VALOR

VALOR

CONST

RESULT

RESULT

FUNCIONES

VALOR

Result

Referencias

Nombre

Parámetros

2) Subprogramas como Parámetro:

- En algunas situaciones es conveniente o necesario poder manejar los **nombres de los subprogramas como parámetros** para ejecutar alguna acción.
- **No** lo incorporan **todos los lenguajes**.
- Es un tema que puede traer confusión.

Parámetros

2) Subprogramas como Parámetro:

Ejemplos:

- Un programa que trabaja con un arreglo al que se le debe aplicar un proceso de ordenamiento. Ese proceso no siempre es el mismo. Sería natural que el nombre del procedimiento que ordena sea un parámetro más.
- Un lenguaje que no proporcionan manejo de excepciones, el nombre de la rutina que haría las veces del manejador, podría ser un parámetro subprograma.

Ada no contempla los **subprogramas como valores**. Pero utiliza **unidades genéricas** en su defecto

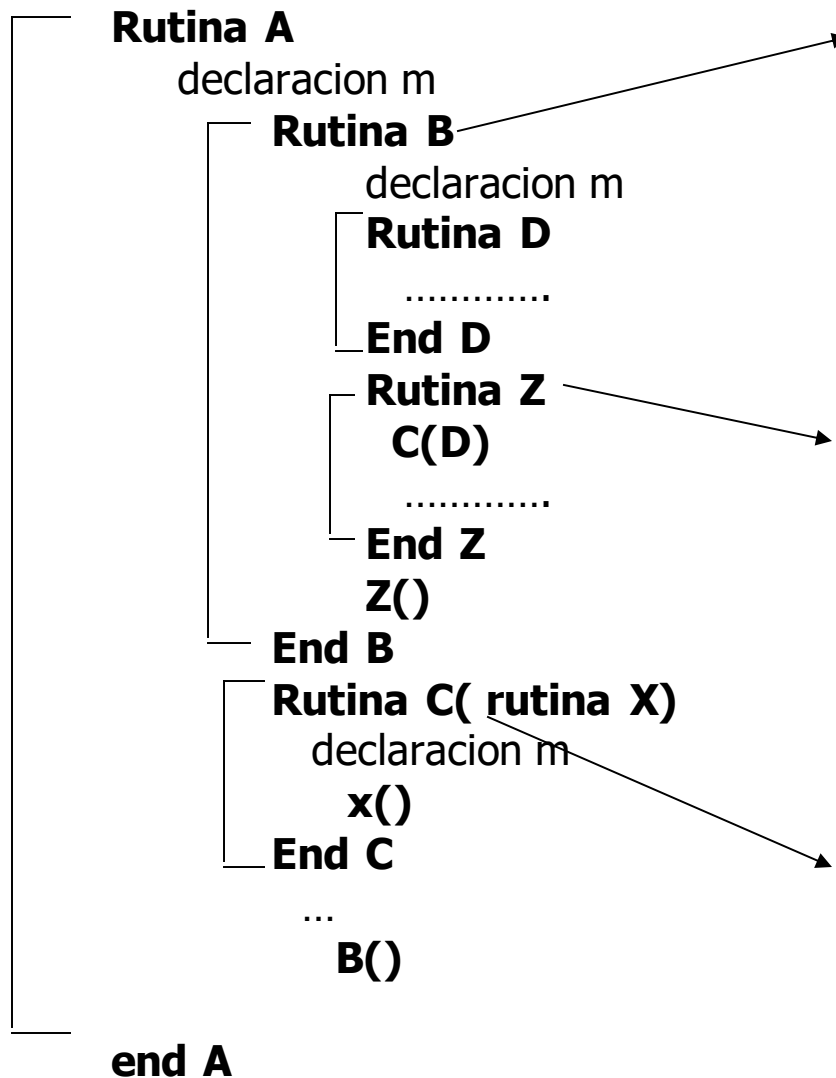
Pascal permite que una **referencia a un procedimiento** sea pasada a un subprograma

C permite pasaje de **funciones como parámetros**.

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.
 - Hay varias opciones:
 - Ligadura **shallow o superficial**: El ambiente de referencia, es el del subprograma que **tiene** el parámetro formal subprograma. Ejemplo: SNOBOL.
 - Ligadura **deep o profunda**: El ambiente es el del subprograma donde **esta declarado** el subprograma usado como parámetro real. Se utiliza en los lenguajes con alcance estático y estructura de bloque.
 - El ambiente del subprograma donde se encuentra el **llamado** a la unidad que tiene un parámetro subprograma. Poco natural. (ad hoc)

Parámetros



Ambiente para el caso de "Profundo". Unidad donde está **declarado** el subprograma **parámetro real: Procedimiento B (deep)**

El ambiente del subprograma donde se encuentra **el llamado** a la unidad que tiene un parámetro subprograma **Procedimiento Z (ad hoc)**

Ambiente para el caso de "Superficial". Unidad donde está el **parámetro formal: Procedimiento C (Shallow)**

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.

Shallow: el entorno 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.

Profunda: 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: el ambiente de referencia es el de la x local al llamado en sub3, y la salida es 3.

Parámetros

Consideraciones para su implementación.

- En algunos casos, el subprograma que declara un subprograma también pasa ese mismo subprograma como parámetro. En esos casos, alcance profundo y alcance ad hoc son lo mismo.
- El enlace ad hoc nunca se ha utilizado porque, uno podría suponer que el entorno en el que el procedimiento aparece como parámetro no tiene conexión natural con el subprograma pasado.
- El enlace superficial no es apropiado para lenguajes con alcance estático con subprogramas anidados. El problema es que el receptor puede no estar en el entorno estático del que envía el parámetro, lo que hace que sea muy poco natural que el procedimiento enviado tenga acceso a las variables del receptor.
- Es más natural que el entorno de referencia esté determinado por la posición léxica de su definición. Por lo tanto, es más lógico que se utilice el alcance profundo.

Parámetros

- Unidades genéricas
 - Una unidad genérica es una unidad que puede instanciarse con parámetros formales de distinto tipo.
 - Por ejemplo una unidad genérica que ordena elementos de un arreglo, podrá instanciarse para elementos enteros, flotantes, etc.
 - Como pueden usarse diferentes instancias con diferentes subprogramas proveen la funcionalidad del parámetro subprograma.

generic

type Elemento is private;
package Conjuntos is
type Conjunto is private;

function Vacio return Conjunto;

procedure Inserta (E : Elemento; C : in out Conjunto);

procedure Extrae (E : Elemento; C : in out Conjunto);

-- pertenencia

function ">" (E: Elemento; C: Conjunto) return Boolean;

function "+" (X,Y : Conjunto) return Conjunto; -- Unión

function "*" (X,Y : Conjunto) return Conjunto; -- Intersección

function "-" (X,Y : Conjunto) return Conjunto; -- Diferencia

function "<" (X,Y : Conjunto) return Boolean; -- Inclusión

No_Cabe : exception;

private

Max_Elementos : constant Integer:=100;

type Conjunto is ...;

end Conjuntos;

package Conjuntos_Reales is new Conjuntos (Float);
package Conjuntos_Enteros is new Conjuntos (Integer);

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.**