



# SEMANTICA OPERACIONAL

## Formas de comunicación entre las rutinas

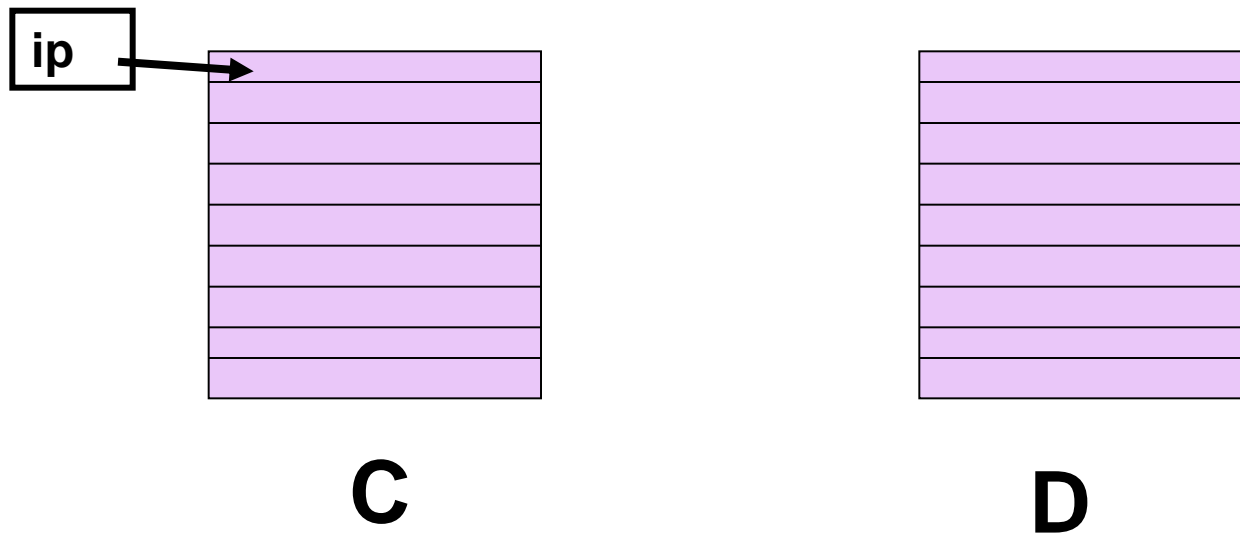
# REPASO CLASE ANTERIOR

Esquemas de ejecución:

- Estático:
  - Todos los registros de activación cargados en la memoria
  - No permite recursión
- Basado en pila:
  - Registros de activación activos
  - Permite recursión
  - Permanente cambio en zona de datos
- Dinámico:
  - Todo el manejo se da en la Heap

# EJECUCIÓN DE LOS LENGUAJES DE PROGRAMACIÓN

procesador



- Casos C1 y C2 sobre esquema Estático
- Casos C3, C4 y C5 sobre esquema Basado en Pila
- Caso C6 sobre esquema Dinámico

# Rutinas

- Conjunto de sentencias que representan **acción abstracta**.
- Representan una **unidad de programa**.
- Amplían el lenguaje
- El ejemplo mas usual y útil presente desde los primeros lenguajes ensambladores son las **subprogramas**, unidades de programa con llamada explicita, que responden al esquema de call/return.
- A nivel de diseño permite definir una operación creada por el usuario a semejanza de las operaciones primarias integradas en el lenguaje



# Rutinas

- Formas de subprogramas
  - Procedimientos
    - Definen **nuevas sentencias** creadas por el usuario.
    - Los resultados los produce en variables no locales o en parámetros que cambian su valor.
    - En general se los llama procedimientos.
  - Funciones
    - Define un **nuevo operador**.
    - Similar a las funciones matemáticas ya que solo producen un valor y no producen efectos laterales.
    - Se las invoca dentro de expresiones y el valor que produce reemplaza a la invocación dentro de la expresión



# Rutinas

## ■ Conclusiones:

- Al diseñar un subprograma el programador se **concentra** en cómo trabaja dicho subprograma.
- Cuando se usa el subprograma se ignorara el **cómo** y se concentra en el **qué** se puede hacer. Es decir 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.



# Parámetros

- Formas de compartir datos entre diferentes unidades:
  - A través del acceso al ambiente no local
  - A través del uso de **parámetros**



# Parámetros

- A través del acceso al ambiente no local
  - **Ambiente común explícito**
  - **Ambiente no local implícito**
    - Utilizando regla de alcance dinámico
    - Utilizando regla de alcance estático





# Parámetros

**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.**



# Parámetros

- Pasaje de Parámetros

¿Qué ventajas tiene respecto forma de compartir accediendo al ambiente no local?



# Parámetros

- Pasaje de Parámetros
  - El pasaje de parámetros es el mas flexible y permite la transferencia de diferentes datos en cada llamada.
  - Proporciona ventajas en **legibilidad y modificabilidad.**
  - Nos permiten compartir los datos en forma abstracta ya que indican con precisión qué es exactamente lo que se comparte



# Parámetros

Parámetros **formales**

```
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**

¿Los parámetros formales son variables locales?  
¿Qué datos pueden ser los parámetros reales?



# Parámetros

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

- **Evaluación:**

- En general en el momento de la invocación primero se evalúa los parámetros reales, y luego se hace la ligadura antes de transferir el control a la unidad llamada.

- **Ligadura:**

- **Posicional:** Se corresponden con la posición que ocupan en la lista
- **Palabra clave o nombre:** Se corresponden con el nombre por lo tanto pueden estar colocados indistintamente en la lista.

En Ada pueden mezclarse ambos métodos.

En C++, Ada, Python los parámetros formales pueden tener valores por defecto, con lo cual a veces no es necesario listarlos todos en la invocación



# Parámetros

- **Clases de parámetros: Datos y Subprograma**

- **Parámetros datos**

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

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

- **Modo IN:** El parámetro formal recibe el dato desde el parámetro real
    - **Modo out:** El parámetro formal envía el dato al parámetro real
    - **Modo IN/OUT:** El parámetro formal recibe el dato del parámetro real y el parámetro formal le envía el dato al parámetro real



# Parámetros

## ■ Modo IN

Por **Valor**

Por **Valor Constante**

### Por Valor:

- El valor del parámetro real se usa para inicializar el correspondiente parámetro real al invocar la unidad.
- Se transfiere el dato real.
- En este caso el parámetro formal actúa como una variable local de la unidad llamada.

*Desventaja:* consume el tiempo para hacer la copia y el almacenamiento para duplicar el dato.

*Ventaja:* protege los datos de la unidad llamadora, el parámetro real no se modifica.



# Parámetros

Ejemplo:

**Procedure A;**

Var x: integer;

**Procedure B( y:integer);**

**Begin**

y:= y + 1;

Write (x,y);

**End;**

**Begin**

x:=9;

B(x);

Write (x);

**End;**





# Parámetros

## Por valor constante:

- No indica si se realiza o no la copia, lo que establece es que la implementación **debe verificar** que el **parámetro real no sea modificado**.
- Ejemplo: parámetros IN de ADA.

*Desventaja:* requiere realizar mas trabajo para implementar los controles.

*Ventaja:* protege los datos de la unidad llamadora, el parámetro real no se modifica

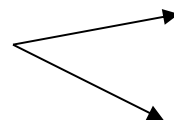
### Ejemplo en C/C++

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



# Parámetros

## ■ Modo OUT



Por **Resultado**

### **Por Resultado:**

**Resultado de funciones**

- El valor del parámetro formal se copia al parámetro real al terminar de ejecutarse la unidad llamada.
  - El parámetro formal es una variable local, sin valor inicial.
- ### ■ *Desventaja:*
- Consume tiempo y espacio
  - Si se repiten los parámetros reales los resultados pueden ser diferentes.
  - Se debe tener en cuenta el momento en que se evalúa el parámetro real
- ### ■ Ventaja:
- protege los datos de la unidad llamadora, el parámetro real no se modifica en la ejecución de la unidad llamada



# Parámetros

## Por resultado de funciones:

- El resultado de una función puede devolverse con el ***return*** como en Python, C, etc., o como en Pascal en el ***nombre de la función*** (ultimo valor asignado) que se considera como una variable local.

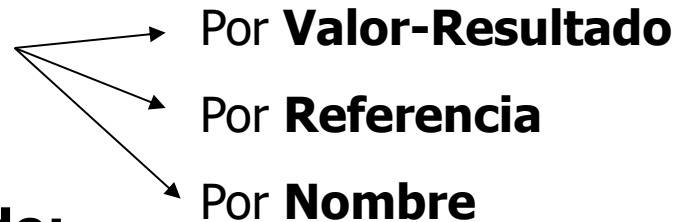
Ej: En C	En Pascal
int f1(int m);	Function
F1(m:integer):integer;	
{....	begin
<b>return(m)</b>	<b>F1:=m + 5;</b>
}	end;

- Dicho resultado reemplaza la invocación en la expresión que contiene el llamado.



# Parámetros

## ■ Modo IN/OUT



### Por Valor/Resultado:

- Copia a la entrada y a la salida de la activación de la unidad llamadora.
- El parámetro formal es una variable local que se 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.
- Cada referencia al parámetro formal es una referencia local.
- Tiene las desventajas y las ventajas de ambos.



# Parámetros

Ejemplo:

**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;**



# Parámetros

## Por Referencia:

- Se transfiere la dirección del parámetro real al parámetro formal.
- El parámetro formal será una variable local a la unidad llamadora que contiene la dirección en el ambiente no local.
- Cada referencia al parámetro formal será a un ambiente no local. Esto significa que cualquier cambio que se realice en el parámetro formal dentro del cuerpo del subprograma quedará registrado en el parámetro real.
- El parámetro real es compartido por la unidad llamada.

### *Desventajas:*

- El acceso al dato es mas lento por la indirección
- Se pueden modificar el parámetro real inadvertidamente
- Se pueden generar alias y estos afectan la legibilidad y por lo tanto la confiabilidad, hacen muy difícil la verificación de programas.

### *Ventaja:*

- Eficiente en tiempo y espacio. No se realizan copias del dato



# Parámetros

Ejemplo:

```
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

## Por Nombre:

- El parámetro formal es sustituido textualmente por el parámetro real.

Es decir 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.**

- El objetivo es otorgar mayor flexibilidad a través de esta 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

Ejemplo:

```
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;
```



# Parámetros

## Por Nombre (continuación):

- Para implementarlo se utilizan los thunks que son procedimientos sin nombre. Cada aparición del parámetro formal se reemplaza en el cuerpo de la unidad llamado por un invocación a un thunks que en el momento de la ejecución activara al procedimiento que evaluará el parámetro real en el ambiente apropiado.
- Es un método mas flexible pero mas lento, ya que debe **evaluarse cada vez que se lo usa**.
- Es difícil de implementar y genera soluciones confusas para el lector y el escritor.



# Parámetros

## ■ Pasaje de parámetros en funciones

- Las funciones no deberían producir efectos laterales. Es decir no deben alterar el valor de ningún dato (local ni no local) solo producir un resultado.
- Los parámetros formales deberían ser siempre modo IN. (Ej. ADA)
- Ortogonalidad. Los resultados deberían poder ser de cualquier tipo. (Ej: Ada, ortogonal; Pascal solo permite escalares)



# Parámetros

## ■ Pasaje de parámetros en algunos lenguajes:

### ■ C:

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

### ■ Pascal:

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

### ■ C++:

- Igual que C más pasaje por referencia

### ■ Java:

- El único mecanismo contemplado es el paso por **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.

### ■ Python:

- Envía objetos que pueden ser "**inmutables**" o "**mutables**". 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):

### ■ PHP:

- Por valor, (predeterminado).
- Por referencia (&)

### ■ RUBY:

- Por valor. Pero al igual que Python si se pasa es un objeto “**mutable**”, no se hace una copia sino que se trabaja sobre él.

### ■ ADA:

- Por copia **IN** (por defecto)
- Por resultado **OUT**
- **IN OUT.**
  - Para los tipos **primitivos** indica por **valor-resultado**,
  - Para los tipos **no primitivos**, datos compuestos (arreglos, registro) se hace por **referencia**
- Particularidad de ADA:
  - **En las funciones** solo se permite el paso por copia de **valor**, lo cual **evita** parcialmente la posibilidad de **efectos laterales**.



# Parámetros

## ■ Subprogramas como parámetro:

- En algunas situaciones es conveniente poder manejar como parámetros los nombres de los subprogramas.

**Ejemplo:** Tenemos 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.**

**Ada** no contempla los subprogramas como valores. Utiliza unidades genéricas.

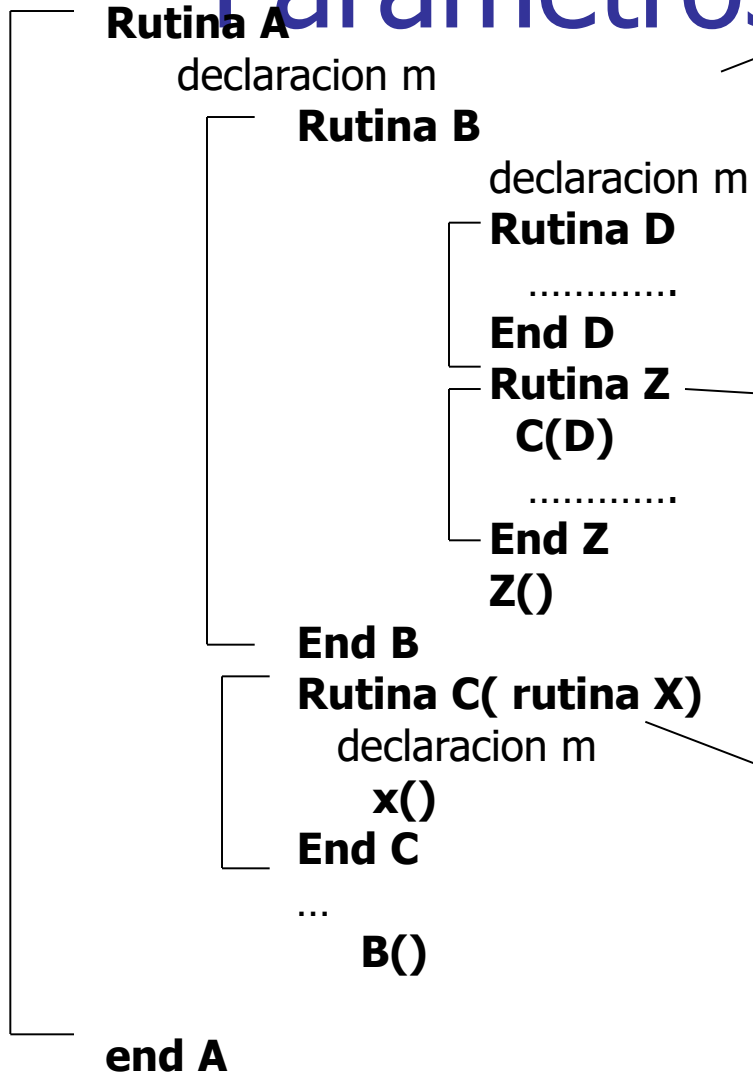
**Pascal** permite que una referencia a un procedimiento sea pasada a un subprograma

**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.

# Parámetros




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

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

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



# Parámetros

- Unidades genéricas
    - Una unidad genérica es una unidad que puede instanciarse con parámetros formales de distinto tipo. Como todas las unidades instanciadas tienen el mismo nombre, da la sensación que una unidad pudiera ejecutarse con parámetros 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);**