TEMA 2

SUBALGORITMOS
FUNCIONES Y PROCEDIMIENTOS

FUNCIONES Y PROCEDIMIENTOS

- ÍNDICE
  - SUBALGORITMOS
  - DEFINICIÓN DE FUNCION Y PROCEDIMIENTO
  - PASO DE PARÁMETROS
  - AMBITO: VARIABLES LOCALES Y GLOBALES
  - EFECTOS LATERALES
  - SUGERENCIAS

SUBALGORITMOS

- Para solucionar un problema complejo se pueden usar varias estrategias:

  - **Divide y vencerás.** Descomposición en partes más sencillas de resolver o “abarcables”.

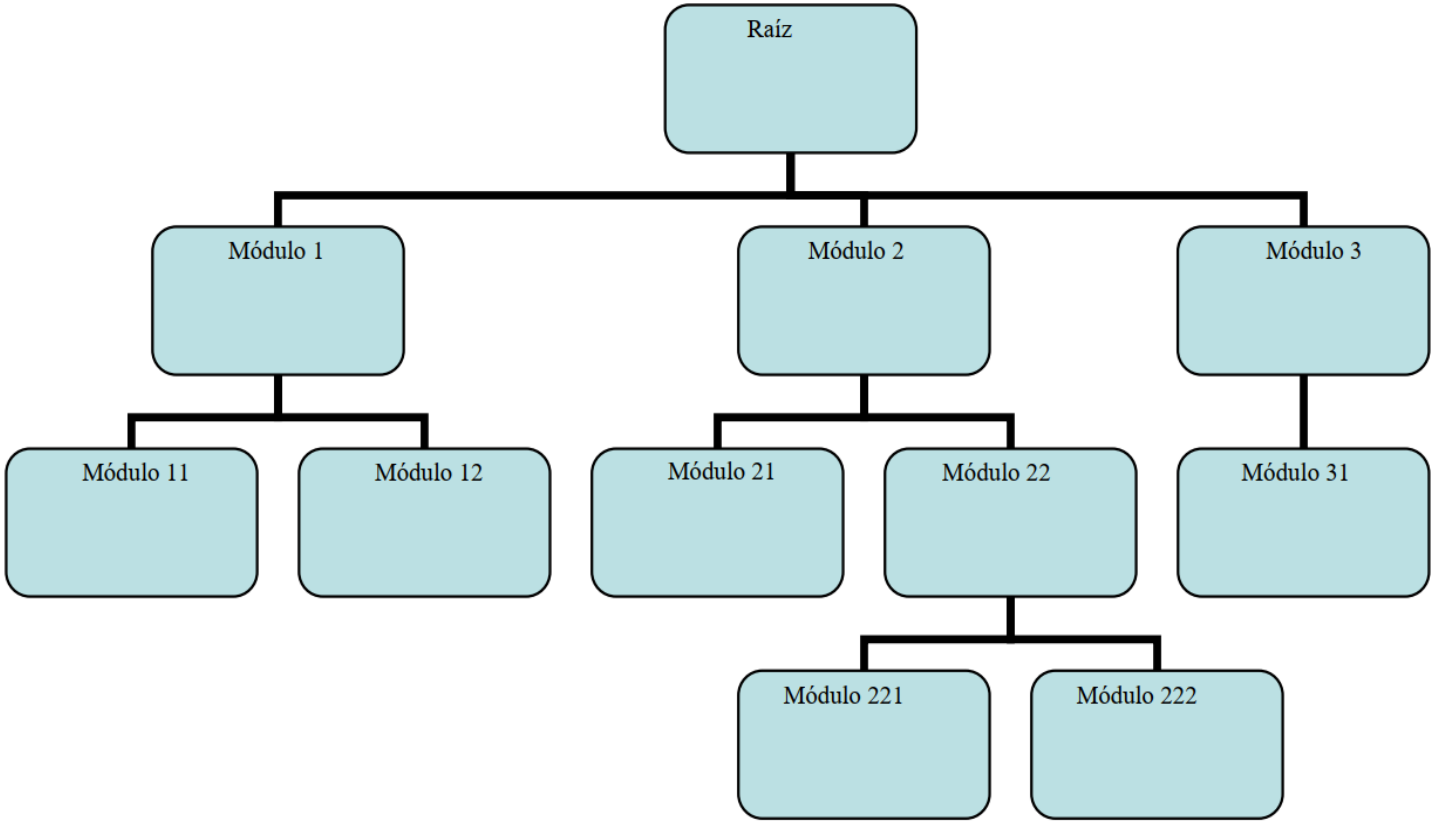
  - Por analogía o similitud. Un problema similar o muy parecido del que conocemos la solución.

SUBALGORITMOS

- Estrategia “**divide y vencerás**": Consiste en partir el problema en subproblemas más simples o **módulos**. Este paso se repite con cada subproblema hasta que sus subsoluciones sean evidentes (**diseño descendente o top-down**).

- Para solucionar cada subproblema se realizan **subalgoritmos**. La mayoría de los lenguajes de programación permiten programar estos subalgoritmos por medio de **subprogramas.**

SUBALGORITMOS



No tiene por qué haber dependencia jerárquica, son **módulos independientes**, pero que se emplean cuando es necesario.

SUBALGORITMOS

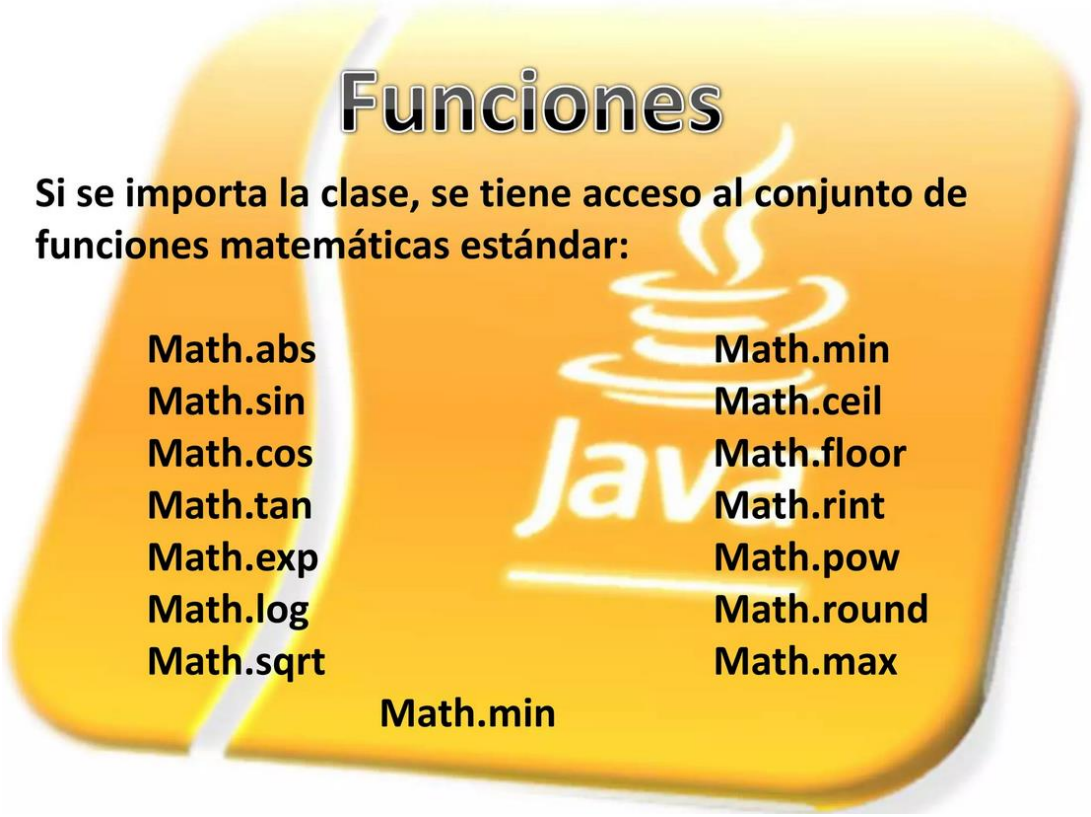
- **¿Cuándo parar el refinamiento?.**

Un refinamiento excesivo podría dar lugar a un número tan grande de módulos que haría poco práctica la descomposición.

Criterio para dejar de descomponer: Cuando el MODULO definido realice una **única tarea**, lo suficientemente **simple y entendible**, y **no** existan **subtareas que requieran descomposición**.

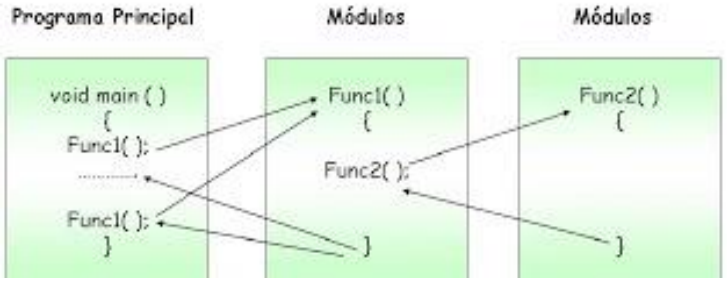
SUBALGORITMOS

- Ejemplos de módulos que realizan una **única tarea**, lo suficientemente **simple y entendible**, y **no** existen **subtareas que requieren descomposición:**



SUBALGORITMOS

- El problema principal se soluciona con el algoritmo principal (el primer paso de la división).
- El programa principal correspondiente se encarga de ensamblar (**llamar o invocar**) correctamente los distintos subprogramas y de **intercambiar información** entre ellos.



- Esta característica permite que los algoritmos sean **simples**, **modulares** y **reutilizables**.

SUBALGORITMOS

- Problemas al escribir programas:
  - código fuente repetido
  - falta de estructuración del código fuente
- Solución: subprogramas
  - pueden ser utilizados desde distintos puntos de un programa (evitan la repetición de código)
  - un subprograma soluciona una parte del problema inicial (facilita la estructuración)
  - son independientes: se pueden escribir y verificar sin preocuparse de detalles de otros subprogramas.
  - facilitan la localización de errores.

SUBALGORITMOS

El uso apropiado de la modularización tiene importantes ventajas:

- **Facilita el desarrollo del software**, dado que los subproblemas con más fáciles de resolver.
- **Facilita la reutilización del software**, dado que un módulo se puede usar en muchos programas.
- **Facilita el mantenimiento**. Se puede profundizar en las pruebas de cada módulo más de lo que se hace en un programa mayor.

SUBALGORITMOS

Dos conceptos importantes:

- **ACOPLAMIENTO:** Se refiere al grado de interdependencia entre módulos. Es preciso minimizar el acoplamiento (interdependencia) entre módulos. Relaciones externas.
- **COHESIÓN:** Fortaleza interna de un módulo, lo fuertemente (estrictamente) relacionadas que están entre sí las partes de un módulo. Relaciones internas.

Los módulos de un programa deben estar débilmente acoplados y fuertemente cohesionados.

Ejemplo librería Math (Java)

SUBALGORITMOS

- **MÓDULOS COMPRENSIBLES:** Para facilitar los cambios, el mantenimiento y la reutilización de módulos, cada módulo debe ser comprensible de forma aislada.

  Para ello, es bueno que posea:

  - **Independencia funcional.**
  - **IDENTIFICACIÓN:** Nombre debe ser adecuado y descriptivo.
  - **DOCUMENTACIÓN**, debe aclarar todos los detalles de diseño e implementación que no queden de manifiesto en el propio código.
  - **SIMPLICIDAD**, las soluciones sencillas son siempre las mejores.

SUBALGORITMOS

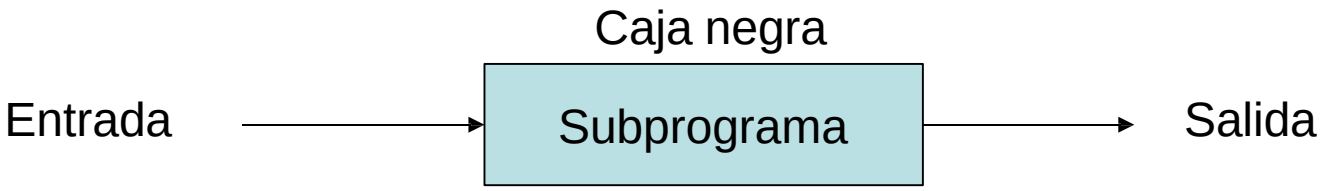
- Tipos de subprogramas:
  - **FUNCIONES:** Es un subprograma que toma uno o más valores denominados **argumentos** y produce un valor denominado **resultado**.

    Es similar a una función matemática.

  - **PROCEDIMIENTOS:** Es un subprograma que ejecuta un proceso especifico y devuelve varios (más de uno) o ningún resultado. Se usan cuando las funciones no son apropiadas.

SUBPROGRAMAS

- Existen dos puntos de vista:
  - El **programador del subprograma**. Se encargará de diseñarlo y declararlo (DECLARACIÓN). Es imprescindible que tenga claro:
    - Datos de entrada exteriores al subprograma, datos que va a devolver, tarea a realizar y cómo va realizarla.
  - El **programa que usa** el subprograma. Se encargará de llamarlo (INVOCACIÓN). Sólo tiene que tener claro:
    - Datos tiene que darle y qué le va a devolver (modelo de la caja negra).



FUNCIONES

- Las funciones son diseñadas para realizar una tarea especifica, tomando unos valores de entrada (argumentos) y devolviendo un único valor.

- **DECLARACIÓN en PseInt**:

```
Funcion {variable de retorno} <- <nombre>
(lista_parámetros Por Valor/Por Referencia)
   [declaraciones locales]
   acciones //cuerpo de la función
FinFuncion
```

  - **{variable de retorno}:** Nombre de la variable que tiene el resultado a devolver.
  - **<nombre>:** Nombre de la función.
  - **(lista_parametros)**: Separados por comas. Si no se indica, los arrays se pasan por referencia, las demás expresiones por valor.

FUNCIONES

- Ejemplo:

```
Proceso EjemploFuncion
    Definir radio, valorArea Como Real;

    Escribir "Escribe el radio: ";
    Leer radio;

    valorArea <- areaCirculo(radio);
    Escribir "Area: ", valorArea;
FinProceso

Funcion area <- areaCirculo(radioCirculo)
    Definir area Como Real;
    area <- 3.1416 * (radioCirculo ^2);
FinFuncion
```

FUNCIONES

- **INVOCACIÓN A UNA FUNCIÓN.**
  - La forma de llamar a una función es la siguiente:

```
nombre (lista_parámetros_actuales)
```

`nombre`: Es el nombre de la función.

`lista_parámetros_actuales`: Son expresiones que dan valor a los parámetros de la función.

Estos parámetros tienen que ***coincidir*** en ***número***, ***orden*** y ***tipo*** con los parámetros formales (los de la cabecera de la función).

Todos los parámetros son de entrada.

FUNCIONES

- **INVOCACIÓN A UNA FUNCIÓN**.

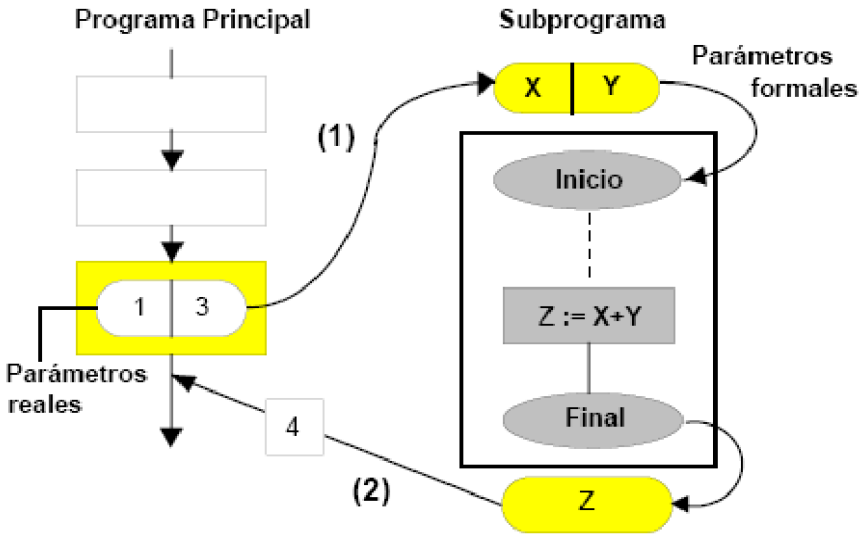
  Ejemplos:

  ```
  x←tangente(30)
  Escribir tangente(2*cos(45))
  preciofinal←descuento(precioinicial,30)
  ```

- NOTAS:
  - Siempre se realiza la llamada dentro de una expresión.
  - La variable a la que se asigna el valor de la función debe tener el **mismo tipo** que el resultado de la función. Ej. x debe ser real.

FUNCIONES

- **INVOCACIÓN A UNA FUNCIÓN**.



FUNCIONAMIENTO (I):

– Al hacer la llamada y ceder el control a la función, se asocia (asigna el valor) de cada **parámetro real** a cada **parámetro formal** asociado, siempre por orden de aparición y de izquierda a derecha, por lo que siempre que no coincidan los tipos y el número de parámetros formales y reales, se produce un error.

FUNCIONES

- **INVOCACIÓN A UNA FUNCIÓN**.

  FUNCIONAMIENTO (y II):

  – Si todo ha ido bien, se ejecutan las acciones de la función hasta que lleguemos a una de tipo *devolver <expresión>* que pondrá fin a la ejecución.*

  – Se le asocia al nombre de la función el valor retornado y se devuelve el control al subprograma que hizo la llamada, pero sustituyendo el nombre de la función por el valor devuelto.

* En algunos lenguajes de programación se espera encontrar un `return`.

En PSeInt no hay `devolver`, cuando llega a `FinFuncion` se devuelve el control al subprograma que hizo la llamada.

FUNCIONES

- **EJEMPLOS:**

- Las funciones matemáticas en **Java** vienen definidas en la clase *Math*:

  - Seno de un ángulo: sin(double a)
  - Coseno de un ángulo: cos(double a)
  - Tangente de un ángulo: tan(double a)

- En **Python** hay que importar la librería *math*:

```
import math
a=math.sin(1)
b=math.cos(1)
c=math.tan(1)
print(a)      ##Imprime: 0.841
print(b)      ##Imprime: 0.54
print(c)      ##Imprime: 1.557
```

PROCEDIMIENTOS

- Son subprogramas que realizan una tarea especifica y que pueden devolver **más de un resultado (0, 1 o n).**
- **DECLARACIÓN en PseInt**:

```
Funcion <nombre> (lista_parámetros Por Valor/
/Por Referencia)
  [declaraciones locales]
acciones //cuerpo del procedimiento
FinFuncion
```

- El significado de cada elemento es el mismo que en las funciones.
- **Ningún valor está asociado con el nombre del procedimiento;** por consiguiente, **no puede ocurrir en una expresión**.

PROCEDIMIENTOS

- Ejemplo:

```
Funcion incrementar (numero Por Referencia)
    numero ← numero+1
FinFuncion
```

PROCEDIMIENTOS

- **INVOCACIÓN A UN PROCEDIMIENTO**.
  - La forma de llamar a un procedimiento es:

```
nombre (lista_parámetros_actuales)
```

nombre: Es el nombre del procedimiento.

lista_parámetros_actuales: Son expresiones o variables.
Coinciden en número, orden y tipo con los parámetros formales.

Ej: `leerVector(a,b)`

`incrementar(c)`

- NOTAS: Siempre se realiza la llamada en una línea, como si fuera una acción.

Los **parámetros pueden ser de ENTRADA, de SALIDA o de E/S.**

PROCEDIMIENTOS

- **INVOCACIÓN A UN PROCEDIMIENTO**.

  FUNCIONAMIENTO (I):

  – Se cede el control al procedimiento al que se llama y se sustituye **cada parámetro formal** de la definición **por el parámetro actual** o real de la llamada asociado a él. La sustitución se hace de izquierda a derecha por orden de colocación.

    Tienen que existir el **mismo número de parámetros** formales que reales, y el **tipo** tiene que **coincidir** con el del parámetro formal asociado. Si no se cumple alguna de estas condiciones se produce un error.

PROCEDIMIENTOS

- **INVOCACIÓN A UN PROCEDIMIENTO**.

  FUNCIONAMIENTO (II):

  – Si la asociación ha sido correcta comienzan a ejecutarse las acciones del procedimiento hasta llegar a la última acción.

    Al terminar se vuelven a asociar los **parámetros formales que devuelven los resultados** a los parámetros reales asociados en la llamada, es decir, de esta manera **algunos** de los **parámetros reales de la llamada ya contendrán los resultados del procedimiento**.

PROCEDIMIENTOS

- **INVOCACIÓN A UN PROCEDIMIENTO**.

  FUNCIONAMIENTO (y III):

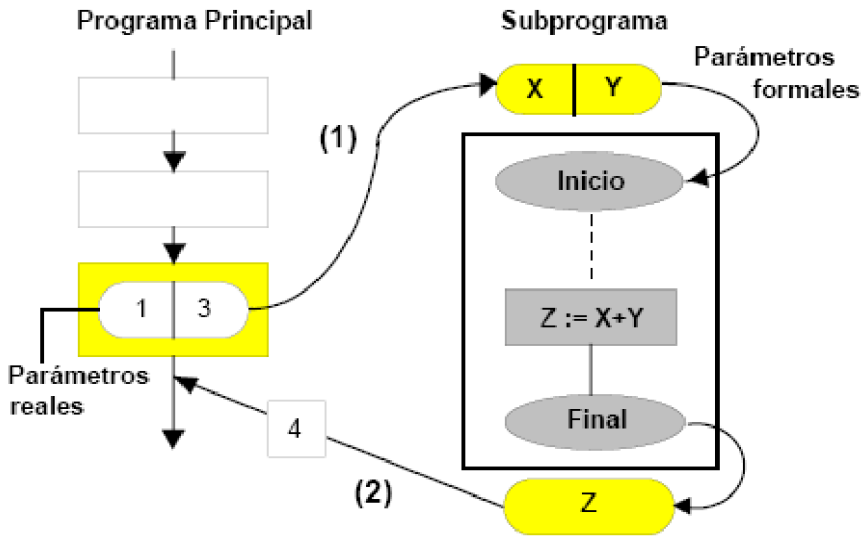
  – Finalmente se cede el **control a la siguiente acción a la que se hace la llamada**, pero teniendo en cuenta que en esta acción y en las siguientes se puede usar ya los parámetros reales en los que se devolvieron los resultados del procedimiento, para trabajar con ellos.

Ejercicio nº 7

PARÁMETROS

- Los parámetros o argumentos son la forma correcta de **intercambiar información** entre el programa principal y el subprograma.

- Clasificación de parámetros, según donde se encuentren en el programa:
  - Parámetros **actuales** o **reales**.
  - Parámetros **formales**.



PARÁMETROS

- Parámetros **actuales** o **reales**: Si los parámetros están en la **llamada** al subprograma. Sus valores se copiarán en los parámetros formales.

- Parámetros **formales**: Si los parámetros están en la **declaración (cabecera) del subprograma**. Su contenido lo recibe al realizar la llamada a la función de los parámetros reales. Los parámetros formales son variables locales dentro de la función.

PARÁMETROS

- Parámetros reales o parámetros formales:
  - Cuando invocamos a un subprograma **NO** es necesario utilizar una **variable con el mismo nombre** del parámetro de definición del subprograma. Lo que se pasan son los valores.
  - Sí es necesario que los **valores** sean de **tipos compatible.**

PARÁMETROS

- Tipos de parámetros, según el flujo de información:
  - Parámetros de **entrada (E)** al subprograma. Si se modifica el valor del parámetro formal, el valor del real no cambia.
  - Parámetros de **salida (S)** al subprograma. No tiene valor inicial en el procedimiento. El valor del formal se pasa al valor del real.
  - Parámetros de **entrada/salida (E/S). El valor inicial se tiene en cuenta.** El valor del formal se pasa al valor del real.

PARÁMETROS

Siempre debemos verificar 3 cosas en la correspondencia entre parámetros actuales y formales:

- Cantidad de parámetros
- Conformidad de tipos de los parámetros
- Orden de los parámetros

PASO DE PARÁMETROS

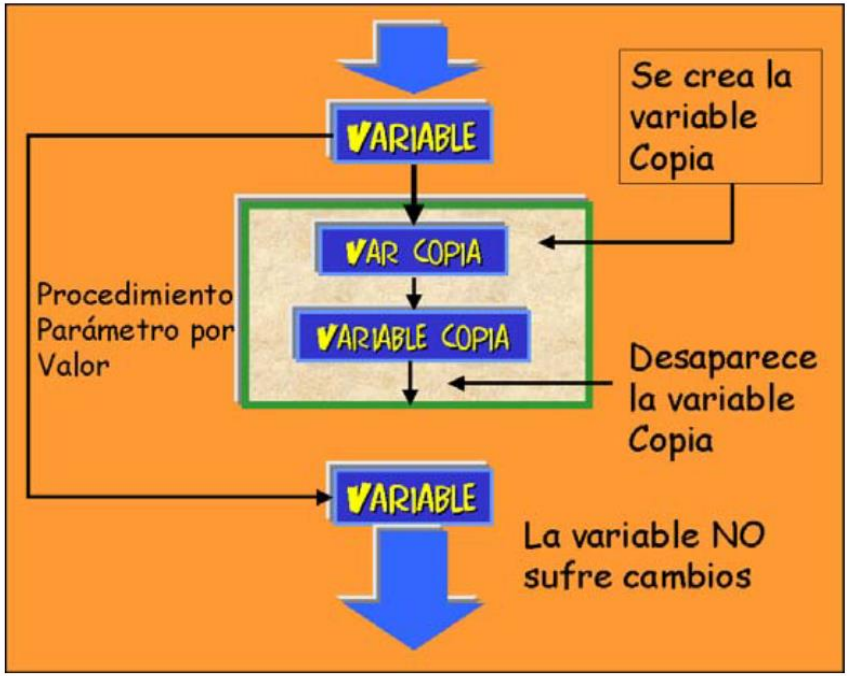
- Existen dos métodos de paso de parámetros:
  - **Paso por valor**: Se usa cuando los parámetros son de entrada.
  - **Paso por referencia o por variable**: Se utiliza para parámetros de salida o de entrada/salida.

PASO DE PARÁMETROS

- **Paso por valor**. Funcionamiento:
  - se calcula el valor de los parámetros reales en la llamada (evaluando las expresiones correspondientes).
  - una **copia** de dicho valor se asigna a los parámetros formales del subprograma.
  - el subprograma opera sobre esta copia.
  - al finalizar el subprograma **se pierde su estado de cómputo local**, y cualquier cambio hecho en el parámetro formal NO quedará reflejado en el parámetro real.

PASO DE PARÁMETROS

- **Paso por valor**.
- **Restricciones**:
  - Los parámetros reales pueden ser **expresiones, variables o constantes**.
  - Los parámetros formales y reales han de ser de tipo **asignación-compatibles**

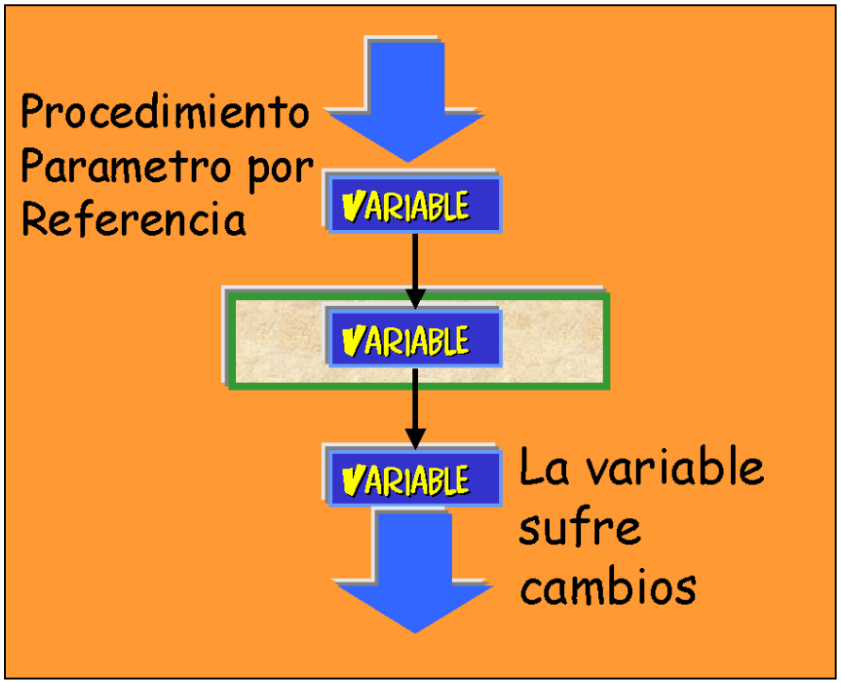


PASO DE PARÁMETROS

- **Paso por referencia o por variable.**
  Funcionamiento:
  - Los parámetros reales sustituyen directamente a los parámetros formales (es decir, los parámetros formales son sinónimos de los parámetros reales).
  - el subprograma va modificando dichos parámetros.
  - aunque al finalizar el subprograma se pierde su estado de cómputo local, cualquier cambio hecho en el parámetro formal **SI** quedará reflejado en el parámetro real.

PASO DE PARÁMETROS

- **Paso por referencia o por variable.**
- **Restricciones**:
  - sólo se permiten **variables** como parámetros reales
  - Los parámetros formales y reales han de ser de tipos **idénticos.**



Ejercicio paso por valor y paso por referencia

Ejercicio nº 8 (para casa)

Ejercicio nº 9

TIPOS DE VARIABLES

- **VARIABLES GLOBALES:**
  - se declaran en el bloque del algoritmo/programa principal

- **VARIABLES LOCALES A UN SUBPROGRAMA:**
  - se declaran en el bloque del mismo subprograma

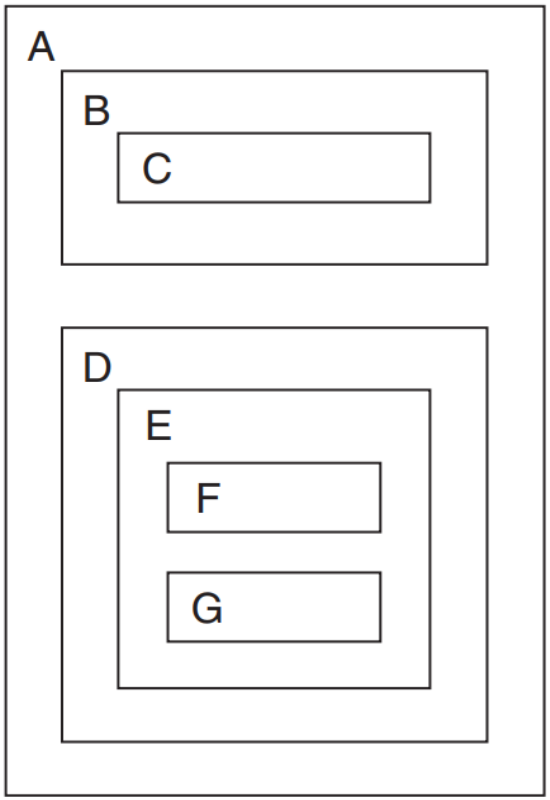
- **VAR. NO-LOCALES A UN SUBPROGRAMA:**
  - se declaran en un bloque exterior al subprograma

AMBITO

- **ÁMBITO** (o visibilidad) de un identificador (variables, constantes, procedimientos, etc.):
  - Parte del programa/algoritmo en que un objeto se define.
  - Bloque en el que el identificador puede ser utilizado.
  - Las **variables globales*** son visibles en todo el programa y subprogramas, excepto en subprogramas que un identificador local tenga el mismo nombre.
  - Las **locales** son visibles en sus propios subprogramas. Este valor no es **accesible** a otros programas, es decir, no pueden utilizar este valor.

NOTA: En PSeInt no se permiten variables globales.

AMBITO

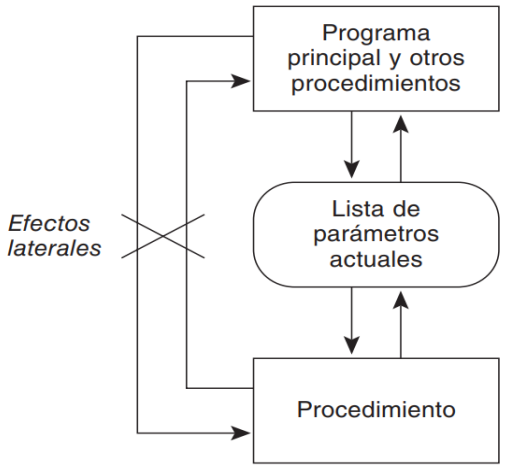


Variables definidas en	Accesibles desde
A	A, B, C, D, E, F, G
B	B, C
C	C
D	D, E, F, G
E	E, F, G
F	F
G	G

EFECTOS LATERALES

- Se producen cuando un subprograma **influye** directamente en el estado de cómputo de **otros** (es decir, **sin** que estos efectos sean producidos por el **paso de parámetros**).

- Es decir, los efectos laterales se producen al modificar variables **globales** o **no- locales** a un subprograma.



EFECTOS LATERALES

- Aunque dichos cambios son sintácticamente correctos, **quedan absolutamente <u>prohibidos</u>** porque:
  - **disminuyen** la **reusabilidad** del subprograma.
  - **dificultan** la **depuración** y **verificación** del programa.

- Para cambiar el valor de variables globales o no locales debemos pasarlas como <u>parámetros por referencia.</u>

SUBPROGRAMAS

- Sugerencias:
  - Antes de desarrollar un subprograma se debe tener claro **para qué** servirá.
  - Cada subprograma deberá ejecutar **una sola tarea**, pero **hacerla bien**.
  - Ningún subprograma deberá **afectar a ningún otro subprograma**. Usar variables locales.
  - Cada subprograma deberá comportarse como una **caja negra**: recibir entradas, efectuar un proceso y entregar resultados.

SUBPROGRAMAS

- Sugerencias:
  - La **comunicación** entre subprogramas se realiza **mediante parámetros**.
  - Si al tratar de realizar una tarea, el subprograma se extiende, se considerará **la división del subprograma**.
  - Cualquier **tarea** que se efectúa **más de una vez** en un programa, debe ser un subprograma.

SUBPROGRAMAS

- Sugerencias:
  - Los subprogramas deben **ser lo más generales posibles**, es decir, servir para el mayor número de entradas.
  - Los subprogramas deben ser **pequeños**, con un tamaño máximo de 20 líneas, aproximadamente, para facilitar su depuración.

Ejercicio nº 10 y 11