PROGRAMACION I

PROGRAMACION MODULAR

Programador Universitario en Informática

FCEyT – UNSE

2022

REGULARIDAD

Aprobar con una nota 50 ptos o más las 2 evaluaciones parciales o sus respectivos recuperatorios.

PROMOCION

Aprobar con 70 ptos o más los 2 parciales

FECHAS DE PARCIALES (Teórico-Prácticos)

Evaluación Previa Parcial Jueves 15/09

PARCIAL 1 - Jueves 29/09

PARCIAL 2 – Martes 15/11

RECUPERATORIOS (Semana del 21/11)

RECORDEMOS

ETAPAS EN LA SOLUCIÓN DE PROBLEMAS CON COMPUTADORA

- ***Análisis del problema***
- ***Diseño del algoritmo***
- ***Codificación del algoritmos***
- ***Verificación***
- ***Documentación***
- ***Mantenimiento***

PROGRAMACION MODULAR

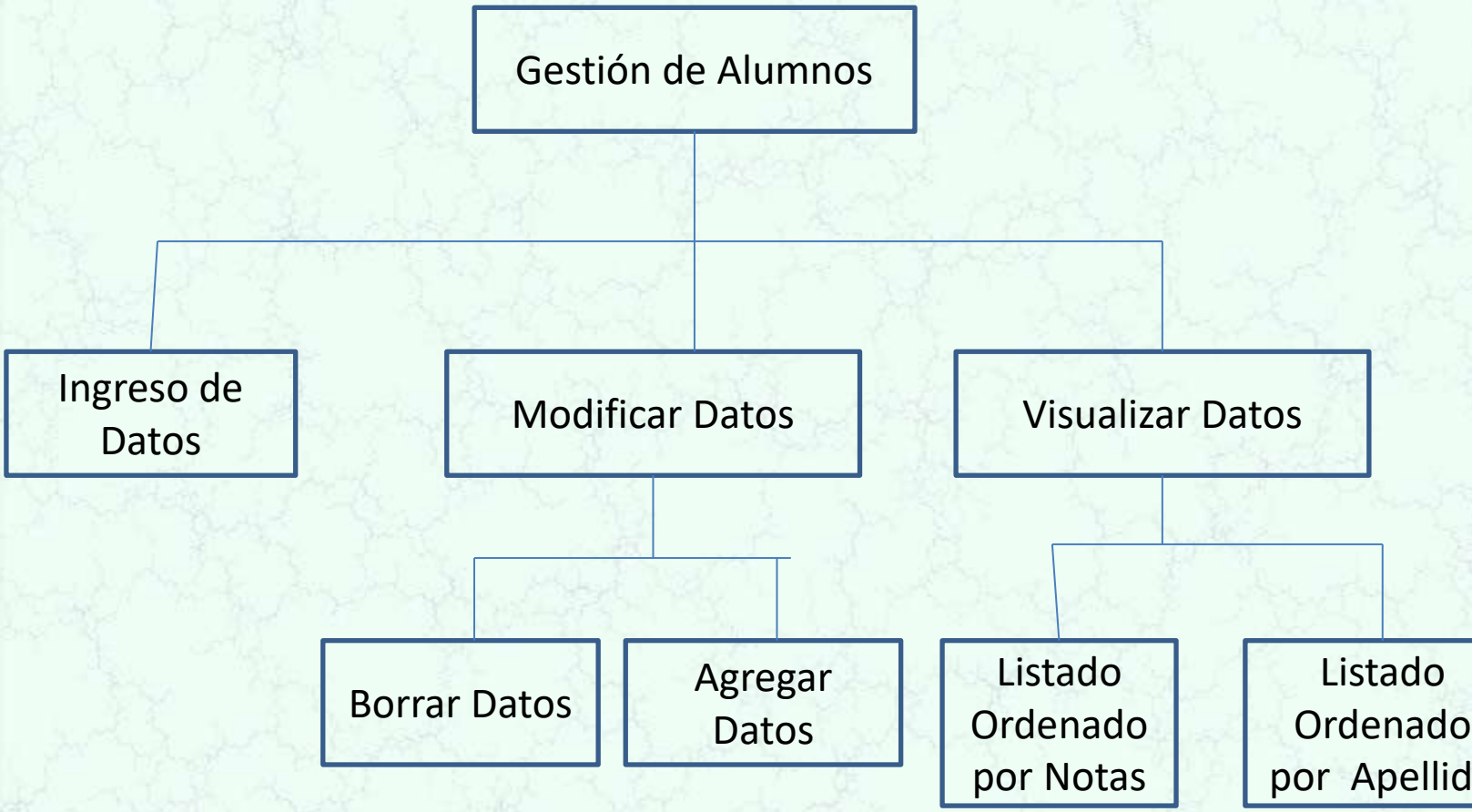
La **programación modular** es un paradigma de **programación** que consiste en dividir un programa en módulos o subprogramas con el fin de hacerlo más legible y manejable.

La programación modular está basada en la técnica de diseño descendente (**Top Down**) , que como ya vimos consiste en dividir el problema original en diversos subproblemas o Módulos que se pueden resolver por separado, para después recomponer los resultados y obtener la solución al problema.

Cada módulo es independiente, ejecutan una única actividad o tarea, los módulos se analizan, codifican y ponen a punto por separado.

La descomposición de un programa en módulos independientes más simples se conoce también como método de ***"divide y vencerás"*** . Se diseña cada módulo con independencia de los demás hasta llegar a la descomposición final del problema en módulos en forma jerárquica.

Ejemplo


Gestión de Alumnos
Ingreso de Datos
Modificar Datos
Visualizar Datos
Borrar Datos
Agregar Datos
Listado Ordenado por Notas
Listado Ordenado por Apellid

El problema se soluciona con el correspondiente Programa Principal, y los subproblemas mediante subprogramas denominados **Procedimientos o Funciones.**

Se dice que el programa principal ***llama o invoca*** al subprograma. El subprograma ejecuta una tarea especifica, y a continuación ***devuelve el control*** al programa que lo convocó. (Figura1)
Un subprograma puede llamar a su vez a sus propios subprogramas (Figura2)

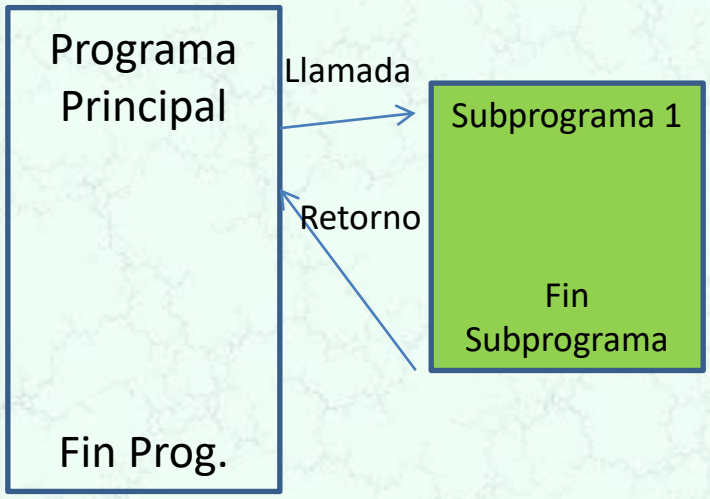


Figura 1

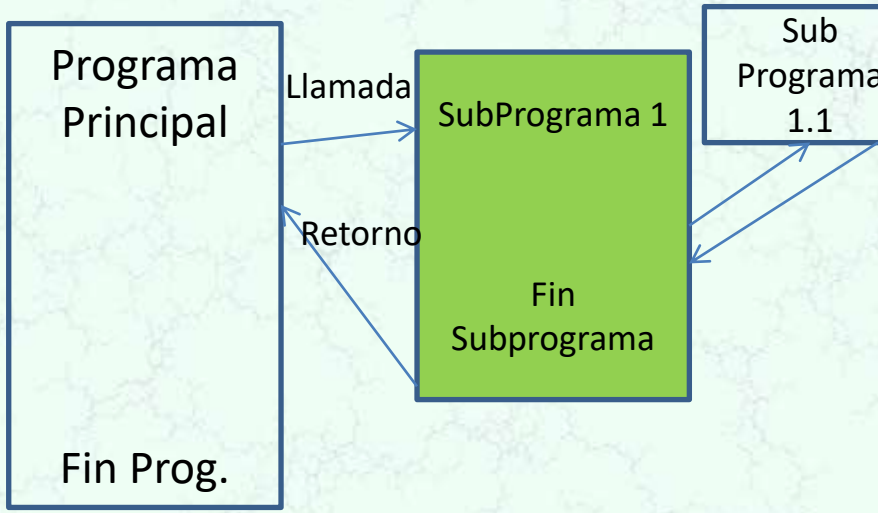
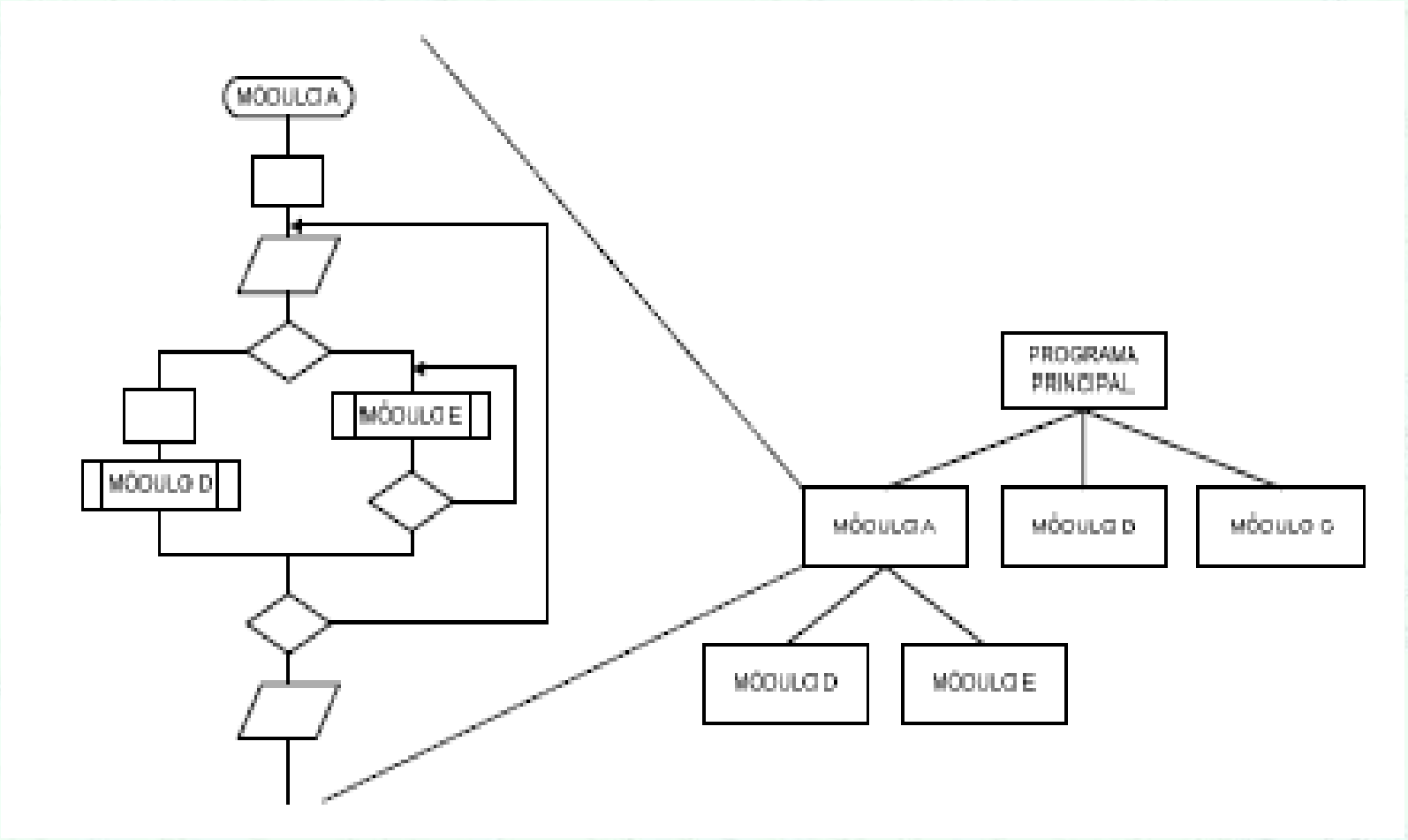


Figura 2

Un programa modular estaría compuesto de :

- ✓Un programa principal, encargado de coordinar la ejecución
- ✓Una serie de módulos que resolverían cada una de las tareas correctas del problema



VENTAJAS DE LA PROGRAMACION MODULAR

- Facilidad para aprehender el problema.
- División del trabajo entre un equipo de programadores.
  - Si los módulos son independientes, cada programador del equipo de desarrollo puede encargarse de uno.
  - El jefe del proyecto integrará los distintos módulos en la aplicación principal.
- Facilidad de mantenimiento y corrección de errores.
  - Si cada módulo cumple una tarea completa es más fácil detectar donde se produce un error.
  - Si se necesita realizar una mejora, solo habrá que modificar un módulo.
- Reutilización del código.
  - Un módulo que realice una tarea determinada podrá utilizarse en otro programa que precise de la misma tarea.

Criterios para la Descomposición Modular

- Es necesario un compromiso entre el tamaño de los módulos y la complejidad de la aplicación.
  - Si un programa de descompone en demasiadas unidades, decrece la efectividad
  - Cuando el número de módulos se incrementa, decrece el esfuerzo para realizarlos, pero aumenta el esfuerzo de integración y la carga en memoria.
- **Independencia funcional**

  Un módulo debe dividirse hasta que se consiga un nivel mínimo aceptable de independencia funcional.

  La independencia funcional de mide a través de dos criterios:

  Cohesión
  - ✓ Mide de relación entre las partes internas de un módulo.
  - ✓ Todas deben estar encaminadas a realizar una única función

  Acoplamiento
  - ✓ Mide la relación del módulo con el resto de los módulos
  - ✓ Debe comunicarse lo menos posible
  - ✓ Pocas veces se conseguirá un acoplamiento nulo
- **Finalmente un módulo debe tener mucha Cohesión y poco Acoplamiento**

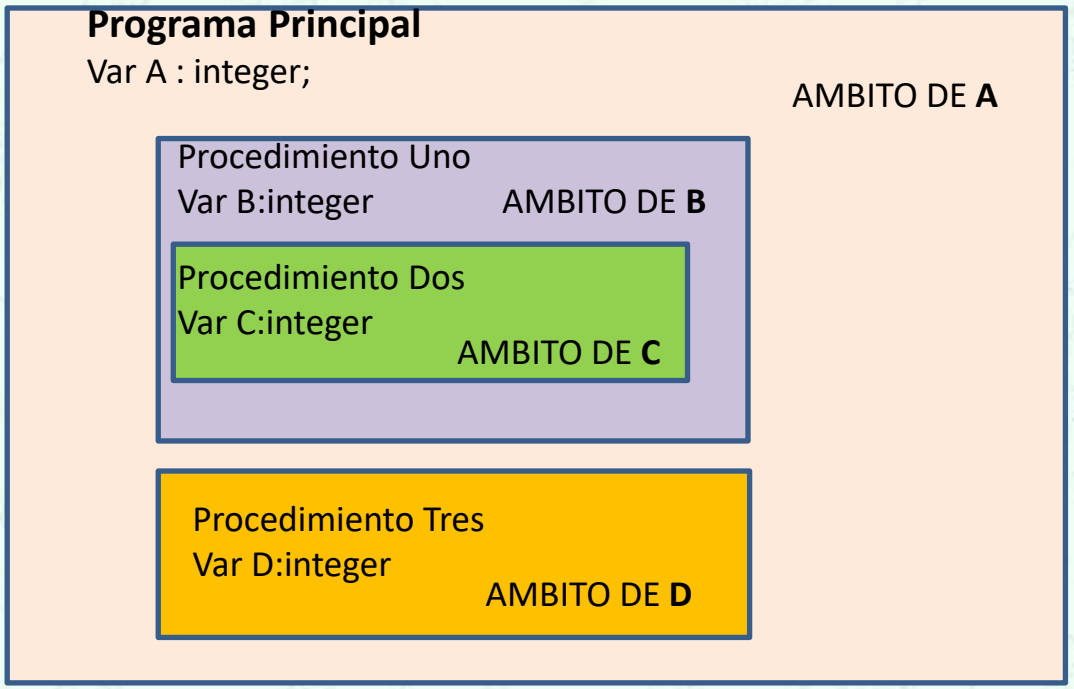
AMBITO DE LAS VARIABLES

VARIABLES LOCALES Y GLOBALES

Las variables que aparecen en un algoritmo o programa se dividen atendiendo a s
ámbito en locales y globales.

- **Variables locales:** son aquellas cuyo ámbito de actuación se reduce a la funció
  o procedimiento donde están definidas. Fuera de su función o procedimient
  estas variables no serán conocidas.
- **Variables globales:** son aquellas definidas por el algoritmo o program
  principal, y potencialmente, su ámbito de actuación se extiende a todos lo
  procedimientos y funciones del algoritmo, siempre que no se hayan (re)definid

en el interior de alguna de ellas, bien variables locales o bien como parámetros,
con el mismo nombre.



VARIABLES LOCALES Y GLOBALES

El uso de las variables locales presenta la ventaja importante de construir módulos altamente independientes, donde las comunicaciones necesarias desde los programas que los invoquen deben hacerse imprescindiblemente mediante el pasaje de parámetros. De esta manera se evitan posibles efectos laterales producidos por operaciones sobre datos dentro de un módulo. Es por ello que siempre se aconsejará limitar el uso de variables globales. ***Cualquier información requerida por un subprograma será pasada como parámetro.***

En definitiva podemos decir que desarrollar módulos con **Independencia *Funcional***, requiere gran uso de variables locales y mínima comunicación externa mediante parámetros es una buena práctica de programación, que favorece la reutilización y el mantenimiento

Ejemplo:

```
Programa Variables Globales Locales;
Var
A, X, Y : entero;

Entero Función FF (N : entero);
Var
        X : entero;
begin
        A = 5
        X = 12
      FF = N+A
Return;
(*cuerpo principal del programa*)
Begin
         X = 5;
         A = 10;
         Y = FF (X)
        Mostrar  “Resultados: “,   X, A, Y
end.
```

Un módulo puede implementarse a través de una Función o Procedimiento

FUNCIONES

Una función es un un conjunto de instrucciones, con un nombre asociado, que cumple las siguientes características:

- Tiene uno o más parámetros de entrada.
- Todos los valores de entrada son necesarios y suficientes para determinar el **único** valor de salida.

Su sintaxis es la siguiente:

Tipo_de_salida/ resultado* FUNCION nombre_funcion (*lista de parámetros formales)
(Declaraciones locales)
INICIO
instrucción 1
instrucción 2
....
RETURN **(*expresión){ de Tipo_de_salida }***
FIN_FUNCION

Ejemplo de una Función

String FUNCION Signo (*x: real);*

VARIABLES
S: string;
INICIO
Si (x = 0) entonces S= "nulo"
sino Si (x > 0) entonces S = "positivo"
sino S = "negativo"
FinSi
FinSi

RETURN **(S *)***
FIN_FUNCION

Programa Números
Var
Z: real;
A: string;
Inicio
Mostrar("Ingrese un número real");
Leer (Z);
A = signo(Z);
Mostrar("El número ingresado es: ", A);

Fin Programa

FUNCIONES DE BIBLIOTECA/ INTERNAS: PERMITEN REALIZAR UNA OPERACIÓN CON SÓLO UNA LLAMADA A LA FUNCIÓN, SIN NECESIDAD DE ESCRIBIR SU CÓDIGO FUENTE.

EJEMPLO: FUNCIONES NUMÉRICAS

- **FUNCIONES MATEMÁTICAS**
  - ✓ **abs (x) DEVUELVE EL VALOR ABSOLUTO DE X.**
  - ✓ **Sqrt (x) DEVUELVE LA RAIZ CUADRADA DE X**
- **FUNCIONES TRIGONOMÉTRICAS**
- **FUNCIONES LOGARÍTMICAS**

FUNCIONES DEFINIDAS POR EL USUARIO: SON LAS DEFINIDAS POR EL PROGRAMADOR PARA IMPLEMENATAR ALGUN MÓDULO O SUBPROGRAMA DISEÑADO

PROCEDIMIENTOS

Son conjuntos de instrucciones con un nombre asociado, al igual que las funciones, que realiza una tarea específica y puede retornar ***ninguno, uno o varios valores como respuesta***

Su sintaxis es la siguiente:

PROCEDIMIENTO nombre_*procedimiento (lista_de_parámetros_formales)*
CONSTANTES
...
TIPOS
...
VARIABLES
...
INICIO
instrucción 1
instrucción 2
....
FIN_PROCEDIMIENTO

Al no soportar el retorno, al contrario que las funciones, para devolver valores se usan parámetros de **salida** o de **entrada/salida.**

PARÁMETROS Y VARIABLES

- En la declaración de un subalgoritmo se incluyen los parámetros a través de los cuales se suministran los datos de entrada.
- En el momento de la llamada se produce un intercambio de información entre l parámetros del subalgoritmo (parámetros formales) y las variables del algoritmo princip (parámetros actuales/ reales).
- Además de los parámetros, pueden existir variables internas que son de ámbito local subalgoritmo.

Ejemplo

```
Algoritmo Prueba_Función

Función real Cuadrado (x)
  variable real x
  variable real y

  y=x*x;
  devolver y
fin Función

  variable real y, z

  Leer(z)
  y = Cuadrado(z)
  Escribir("El cuadrado de ", z, "es", y)

fin Algoritmo
```

Parámetro formal

Variable local a la función

Parámetro real

Paso de parámetros

- Existen dos modos de paso de parámetros:
  - **Por valor**: el contenido de los parámetros reales se copia a los parámetros formales. Las modificaciones internas al subalgoritmo no afectan a la variable externa. Pueden verse como **parámetros de entrada**.
  - **Por referencia**: se pasa una referencia a la dirección de memoria del parámetro real. Si se modifica el parámetro formal se modifica la variable externa. Pueden verse como **parámetros de salida o de entrada/salida**.
- En la lista de parámetros, el paso por referencia se indica anteponiendo **ref** al parámetro.

Ejemplo

```
Algoritmo Prueba_Procedimiento

Procedimiento Pot_23(x, ref x2, ref x3)
 variable real x, x2, x3

 x2 = x*x,  x3 = x2*x
 x = 0
 retornar
fin Procedimiento

 variablesreal y, y2, y3
 y = 7, y2 = 0, y3 = 0

 Pot_23(y, y2, y3)
 Escribir("V=", y, "V^2=", y2, "V^3=", y3)

fin Algoritmo
```

Paso por valor

Paso por referencia

Modificación de variables interna al procedimiento

Inicialización

Ejemplo - continuación

```
Algoritmo Prueba_Procedimiento

Procedimiento Pot_23(x, ref x2, ref x3)
 variable real x, x2, x3

 x2 = x*x,  x3 = x2*x
 x = 0
 retornar
fin Procedimiento

 variablesreal y, y2, y3
 y = 7, y2 = 0, y3 = 0

 Pot_23(y, y2, y3)
 Escribir("V=", y, "V^2=", y2, "V^3=", y3)

fin Algoritmo
```

Paso por valor

Paso por referencia

Estado de la memoria antes de la llamada

y	7
y2	0
y3	0

Ejemplo - continuación

```
Algoritmo Prueba_Procedimiento

Procedimiento Pot_23(x, ref x2, ref x3)
 variable real x, x2, x3

 x2 = x*x,  x3 = x2*x
 x = 0
 retornar
fin Procedimiento

 variablesreal y, y2,
 y = 7, y2 = 0, y3 = 0

 Pot_23(y, y2, y3)
 Escribir("V=", y, "V^2=", y2, "V^3=", y3)

fin Algoritmo
```

Paso por valor

Paso por referencia

x2 y x3 hacen referencia a la misma dirección que y2 e y3, respectivamente

Estado de la memoria al entrar en el procedimiento

y	7
y2, x2	0
y3, x3	0
x	7

x es una variable distinta sobre la que se copia el valor de y

Ejemplo - continuación

```
Algoritmo Prueba_Procedimiento

Procedimiento Pot_23(x, ref x2, ref x3)
  variable real x, x2, x3

  x2 = x*x,  x3 = x2*x
  x = 0
  retornar
fin Procedimiento

  variablesreal y, y2, y3
  y = 7, y2 = 0, y3 = 0

  Pot_23(y, y2, y3)
  Escribir("V=", y, "V^2=", y2, "V^3=", y3)

fin Algoritmo
```

Paso por valor

Paso por referencia

Estado de la memoria tras ejecutar el cuerpo del procedimiento

y	7
y2, x2	49
y3, x3	343
x	0

Ejemplo - continuación

```
Algoritmo Prueba_Procedimiento

Procedimiento Pot_23(x, ref x2, ref x3)
 variable real x, x2, x3

 x2 = x*x,  x3 = x2*x
 x = 0
 retornar
fin Procedimiento

 variablesreal y, y2, y3
 y = 7, y2 = 0, y3 = 0

 Pot_23(y, y2, y3)
 Escribir("V=", y, "V^2=", y2, "V^3=", y3)

fin Algoritmo
```

Paso por valor

Paso por referencia

y no ha cambiado

Estado de la memoria al volver al programa principal

y	7
y2	49
y3	343

y2 e y3 se han modificado

MATERIAL A CONSULTAR PARA ESTA UNIDAD

Capítulo 6 – Procedimientos y Funciones
(Libro: Fundamentos de Programación
Luis Joyanes Aguilar)

Capítulo 4 – Procedimientos Y Funciones. Parámetros.
(Libro: Algoritmos, datos y programas- Armando E. D
Giusti)

Capítulo 8 .Ptos. (8.3.4), (8.3.5), (8.3.6) y (8.3.7),
(Libro: Ingeniería de Software. Un enfoque práctico- Rog
Pressman)