Unidad 5

DISEÑO MODULAR: SUBPROGRAMAS

Objetivos

- Construir la solución específica del problema con módulos funcionales interrelacionadas (subprogramas).
- Distinguir entre los dos tipos de subprograma: procedimiento y función y utilizar cada uno de ellos de forma adecuada.
- Establecer la intercomunicación entre subprogramas a través de las variables de enlace.
- Distinguir entre las dos formas existentes de intercomunicación de parámetros: por valor y por referencia o variable, empleando cada una de ellas adecuadamente.
- Escribir correctamente las llamadas a funciones y procedimientos.
- Comprender y usar correctamente la técnica de diseño Top-Down

Contenidos

- Introducción
- Diseño modular: enfoque Top-Down
- Módulos. Diagramas jerárquicos
- Metodología. Documentación
  - Metodología
  - documentación
- Programa principal y subprogramas
  - El programa principal
  - Subprogramas
  - Cohesión y acoplamiento
- Tipos de subprogramas.
  - Según su ubicación.
  - Según el valor de retorno
- Variables de enlace o parámetros
  - Tipos de parámetros
  - Paso de parámetros
- Objetos locales y globales
- Procedimientos y funciones
- Diseño de la interfaz
  - Pasos para el diseño de interfaz
  - ¿Cómo queda el diseño del algoritmo?
- Llamadas a un subprograma desde distintos módulos.

Introducción

- Hasta ahora se han usado subprogramas, tanto en pseudocódigo (***escribir***, ***leer***, ***raíz***), como en Java (***print***, ***read, sqrt***).
- Hemos aprendido a usar módulos en el diseño del pseudocódigo generalizado.
- Ahora aprenderemos a incluir estos módulos como subprogramas en el pseudocódigo detallado.

Realizar diseños modulares?

La importancia del diseño, para frenar las ganas de crear código rápidamente, se resume en una palabra: ***calidad***.

El diseño modular contribuye directamente a ello.

Diseño Modular

- No usar una buena metodología de diseño ocasiona:
  - Disminución de la legibilidad, comprensión y modificabilidad de los programas.
  - Invertir demasiado tiempo en corrección de errores.
  - Imposibilidad de reutilización de código.
  - Documentación nula o insuficiente.
- Con la programación modular se evitan los anteriores problemas.

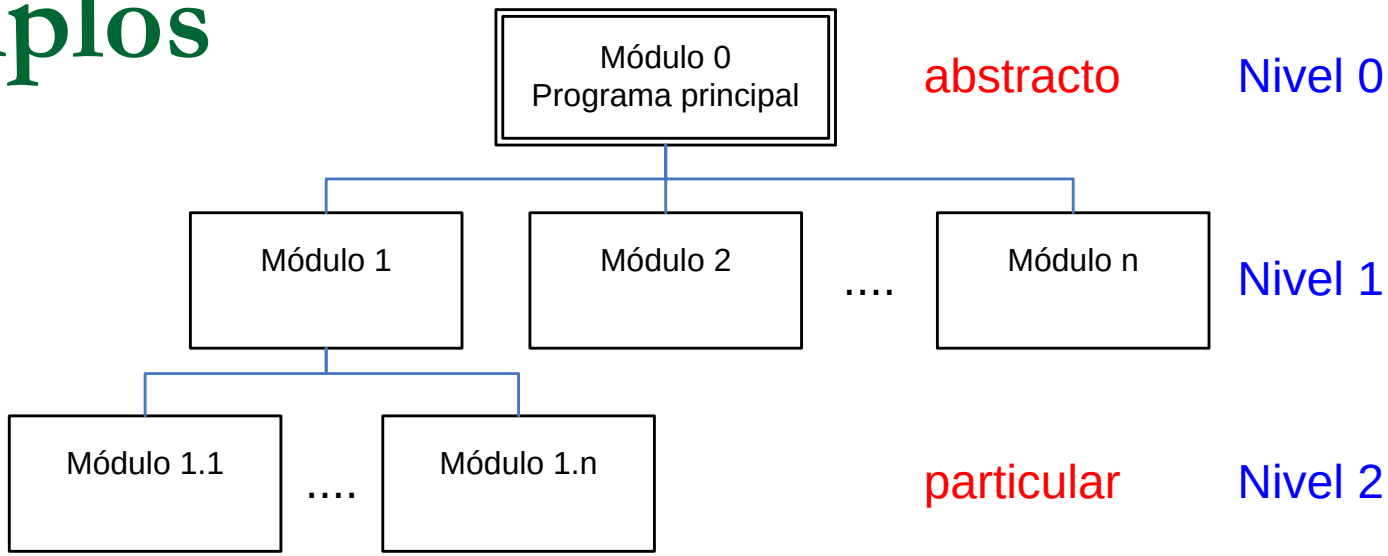
Diseño modular

- **Definición**: Metodología de desarrollo que consiste en construir un problema grande a partir de un conjunto de otros más pequeños, que juntos dan solución al primero.
- Se desea que los componentes sean muy independientes entre ellos e independientes del algoritmo principal, de forma que puedan ser diseñados sin considerar el contexto en el que van a ser usados.
- La técnica utilizada para ello:
  - Enfoque ***Top-Dow***, llamado también ***Diseño descendente*** o ***Refinamiento por pasos sucesivos.***
  - Enfoque **Bottom-Up** (De abajo a arriba, construcción de algoritmos a partir de componentes prefabricados . Se usará en MOO).

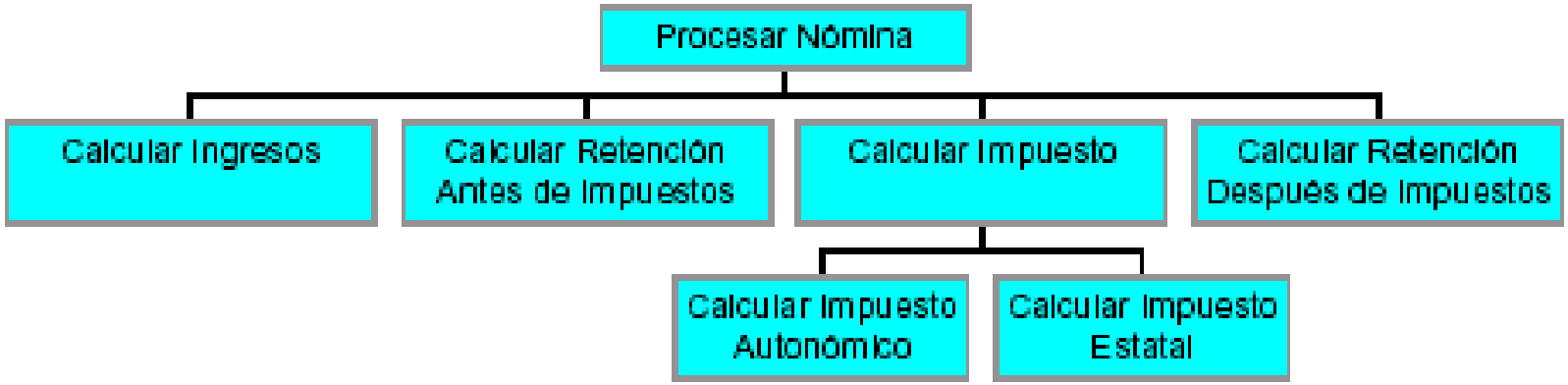
Módulos, subprogramas y Diagramas

- Al dividir un problema en otros más sencillos, se crea una estructura jerárquica de módulos llamados ***módulos funcionales***.
- La representación gráfica de esta estructura se conoce como ***Diagrama de Estructura de Módulos*** **(DEM).**
- El **DEM** contiene los distintos niveles de refinamiento.
- ***Subprograma*** es un algoritmo que puede ser diseñado y codificado independientemente del contexto en el que va a ser usado.
- Los módulos que se conviertan en subprogramas formarán el ***Diagrama de Descomposición funcional*** **(DDF).**

Ejemplos



Algoritmo de proceso de nóminas



Ventajas del Diseño Modular

- Simplifica el diseño.
- Permite la programación aislada (***Encapsulamiento***) .
- Permite aplicar la ***estrategia de Escarlata O'Hara*.**
- Aporta conocimiento sobre ***qué*** hace el módulo y no ***cómo*** lo hace (***Abstracción***).
- Permite la compilación separada
- Agiliza la depuración de programas al permitir que cada módulo pueda depurarse de manera independiente.

Mas Ventajas del Diseño Modular

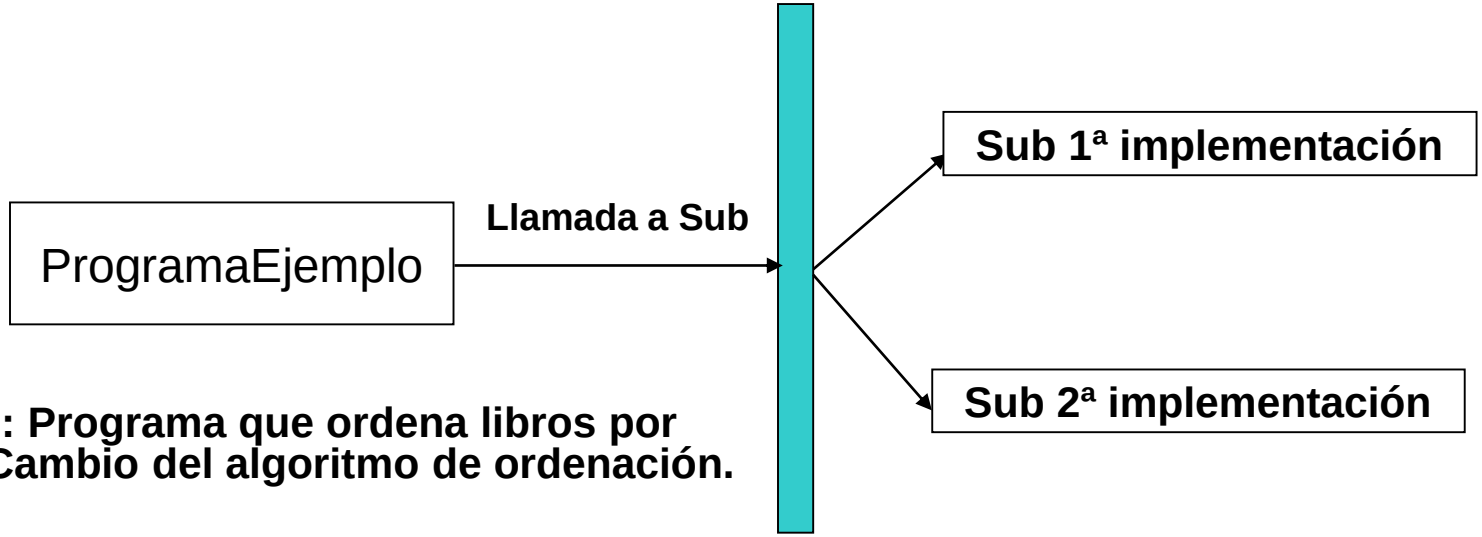
- Reutilización del módulo en otro contexto.
- Facilita el trabajo en equipo, al permitir que la resolución del problema la lleven a cabo varios programadores.
- Contribuye a la comprensión y legibilidad de los algoritmos.
- Reduce el tiempo y coste del desarrollo de software por la reutilización de módulos ya existentes.
- Favorece el mantenimiento y modificación de los módulos ya diseñados y, por consiguiente, de los programas.

Encapsulamiento y Abstracción

- ***Encapsulamiento***: característica que separa la implementación del uso de un subprograma.

  Si el método que soluciona una tarea A cambia su forma de solucionarla, cualquier programa que lo usa no se ve afectado.

  Por ejemplo, un *ProgramaEjemplo* que usa un subprograma *Sub* no se verá afectado si el código de *Sub* varía pasando de una 1ª implementación a otra 2ª implementación.
- ***Abstracción***: Sabemos qué hace no cómo ni con qué herramientas.



Ejemplo: Programa que ordena libros por autor || Cambio del algoritmo de ordenación.

Aplicando la técnica

- Pasos a seguir:
  - Pasar un tiempo pensando en el problema globalmente.
  - Escribir los pasos principales, especificando la estructura principal del programa con los módulos que contiene ➔ Da lugar al Pseudocódigo generalizado.
  - Examinar cada uno de los pasos anteriores completando los detalles.
  - Si no se sabe realizar una determinada tarea, se le da un nombre y se pasa a la realización de la tarea siguiente, más tarde será atendida (***estrategia de Escarlata O'Hara***).
  - Este proceso continúa en tantos niveles como sea necesario, hasta completar el diseño.

Módulo ⇨ Subprograma

- Para elegir qué módulos se convierten en subprogramas se debe evaluar:
  - Si favorece o no la legibilidad y comprensión del problema global.
  - Las perspectivas de reutilización.
  - **El coste de *interfaz*** (forma de comunicación). Se determina en función de:
    - Establecimiento de variables de enlace (vías de comunicación).
    - Requerimientos de memoria.
    - Aumento de tiempo de ejecución.

Ver ejemplo pag. 4

Metodología de diseño

1) **Análisis**: , Entradas, salidas, requisitos, Estudio de bucles (Como hasta ahora).
2) **Solución General**: Reflejará la estructura general del programa y aparecerán los nombres de los módulos que contenga (Como hasta ahora).
3) **Definición de módulos**: Escribir el código correspondiente de los módulos y realizar la traza de cada uno de ellos (Como hasta ahora).(*)
4) **R**ealizar el **D**iagrama de **E**structura de **M**ódulos (**Nuevo**).
5) **D**ecidir los módulos que se van a convertir en subprogramas. De todos éstos, se estudiará su **interfaz** (**Nuevo**).
6) **R**ealizar el **pseudocódigo detallado** del algoritmo (se hará en Java en la Unidad siguiente), donde aparecerá el código correspondiente a los módulos no convertidos en subprogramas y **las llamadas** a los que sí se han convertido (Un poquito diferente a lo de ahora).
7) **R**ealizar el **D**iagrama de **D**escomposición **F**uncional (**Nuevo**).
8) **E**jecutar la traza, enlazando programa principal y subprogramas (**Nuevo**).

Documentación

Ver ejercicio pag. 7

A la documentación utilizada hasta ahora se añadirán descripciones y especificaciones de cada módulo, y otra documentación anexa.

- **Documentación externa** que contiene:
  - **Especificaciones**: ¿qué hace cada módulo? ¿en qué contexto?.
    Para los módulos que vayan a convertirse en subprogramas, deberá detallarse su **interfaz**.
  - **Desarrollo**. Breve historia de las modificaciones sufridas por el programa.
  - **Esquema gráfico** del diseño descendente si lo hubiera (DEM y/o DDF).
- **Documentación interna,** código autodocumentado:
  - Comentarios en la llamada a subprogramas.
  - Comentarios en los subprogramas incluyendo:
    - Etiqueta de cada subprograma con su nombre.
    - **Interfaz** (delante de la definición de cada subprograma).
    - Comentarios para explicar el propósito de variables de enlace.

Interfaz:
Proceso que realiza.
Precondiciones.
Entradas.
Salidas.
Postcondiciones.

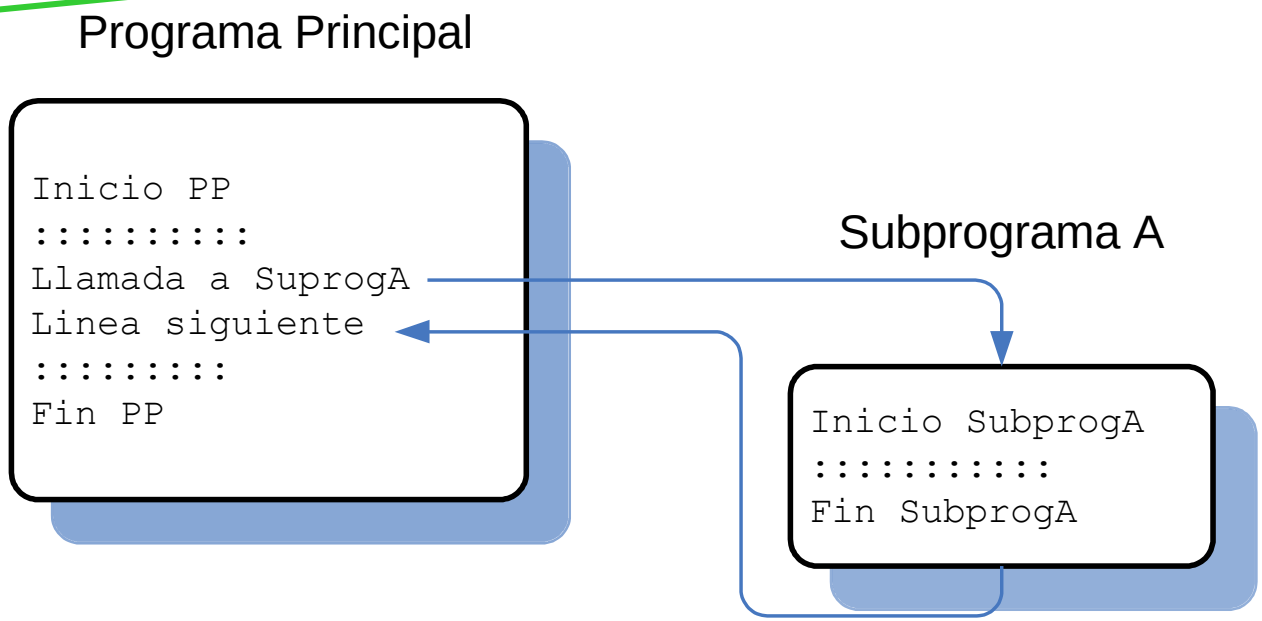
Programa principal

- Describe la semántica de la solución completa del problema.
- Consta principalmente de:
  - **definiciones de tipos,**
  - **declaraciones de objetos,**
  - **arquitectura de primer nivel:**

- estructuras de control básicas

- llamadas a subprogramas

¡¡Una llamada a subprograma pasa el control de flujo al subprograma, ejecuta su código y vuelve al PP!!.



Subprogramas

- **Un subprograma es una estructura de control de flujo**.
- La llamada a un subprograma es una instrucción que transfiere el control al subprograma.
- Un subprograma se ejecuta cuando es llamado por el programa principal o por otro subprograma.
- La estructura de un subprograma es la de un programa principal sencillo, con algunas diferencias en el encabezado y en la finalización.
- Un subprograma puede ser llamado cuantas veces sea necesario, tanto por el programa principal, como por otros subprogramas.

Indicadores de calidad

Para evaluar el diseño modular se usan indicadores de calidad como la *cohesión* y el *acoplamiento*.

Ambos influyen en la calidad y en el coste del producto resultante.

Cohesión → HallarMayorMenor ( )

- **Definición: medida del grado de identificación de un módulo con una funcionalidad o tarea concreta.**
- **Estudia la relación existente entre todos los componentes de un módulo. Si están altamente relacionados están muy cohesionados.**
- **Existe una escala para medir la coherencia (Stevens, buscar en Internet).**
- **Un subprograma altamente coherente debe realizar una única tarea.**

Indicadores de calidad

Acoplamiento ⟶ Ordenar( )

- **Definición: medida del grado de interdependencia de los módulos de un programa.**
- **Los módulos deben ser absolutamente independientes. (*)**
- **Si un módulo necesita algo de otro deben comunicarse a través de las variables de enlace.**

Criterios de Modularización

- No existe un procedimiento formal para descomponer un problema en módulos.
- Algunos criterios a seguir:
  - Minimizar el acoplamiento (los módulos deben ser tan independientes entre sí como sea posible).
  - Maximizar la cohesión (Un módulo debe realizar una única tarea).
    - Como criterio para establecer el grado de cohesión según Stevens: ***"Escribir una frase que describa el propósito del módulo y examinarla; si la frase no contiene un objeto específico sencillo a continuación del verbo lo más normal es que estemos en la banda baja de cohesión"***

      Por ejemplo, Hallar el factorial de un número (alta cohesión), Calcular el volumen y área de la esfera e imprimirlos en pantalla (baja cohesión).

Tipos de subprogramas

- **Según su ubicación**:
  - Subprogramas internos:
    - Generalmente se declaran y/o se definen físicamente antes del cuerpo del programa principal.
    - Las llamadas se hacen por su nombre.
  - Subprogramas externos:
    - Su declaración y definición (código) figuran físicamente separados del programa principal.
    - Pueden compilarse separadamente e, incluso, codificarse en un lenguaje distinto al del programa que lo usa.
    - Las llamadas se hacen por su nombre.
    - Generalmente, no se dispone de su código fuente.

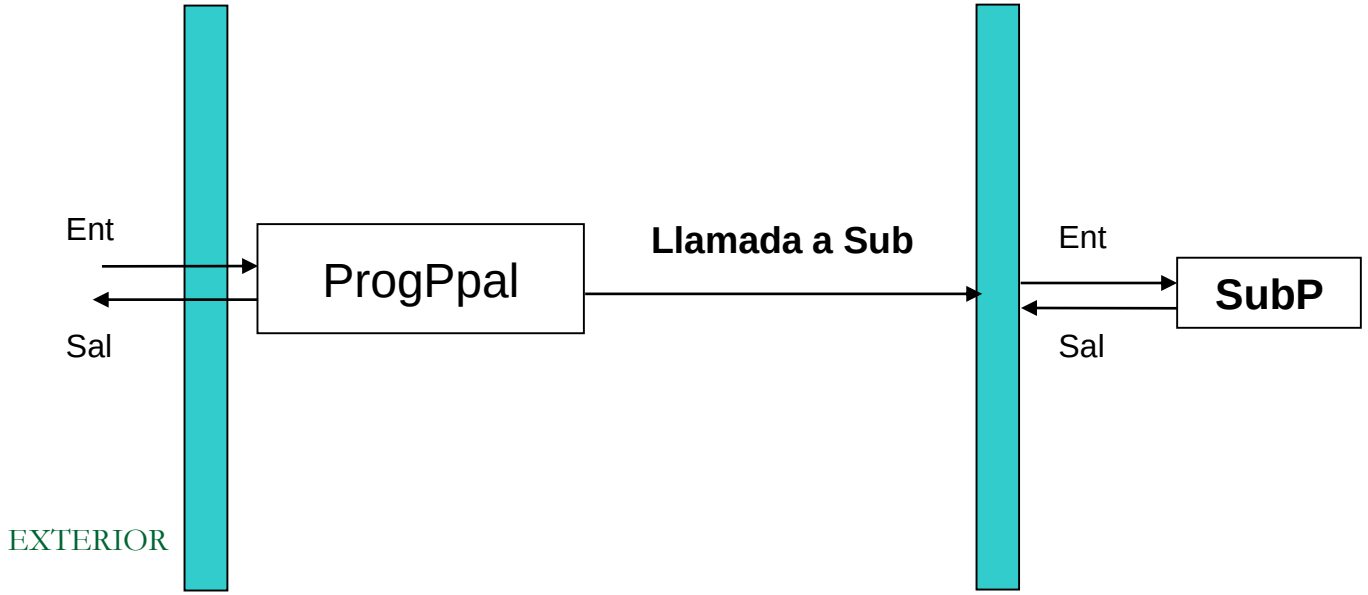
Tipos de subprogramas

- **Según el valor devuelto o valor de retorno**:
  - Procedimientos:
    - No tienen valor de retorno, es decir, no tienen valor asociado al nombre, por lo que no puede aparecer a la derecha de un símbolo de asignación.
  - Funciones:
    - Tienen valor de retorno, es decir, tienen valor asociado a su nombre, por lo no puede aparecer a la izquierda de un símbolo de asignación.

En Java todos los subprogramas se conocen como funciones, aunque incorpora herramientas para implementar procedimientos (Capítulo 6).

Variables de enlace o parámetros

- **El programa principal**
  - Sus datos de entrada vienen de las unidades de entrada.
  - Los resultados son enviados a las unidades de salida.
- **El subprograma**
  - Sus datos de entrada vienen del PP o del subprograma que lo llama.
  - Los resultados son enviados al PP o al subprograma que lo llama.



Para este paso de información se usan los parámetros

Tipos de parámetros

- Parámetros formales (*):
  - son **variables** u otros **objetos de datos locales** de los subprogramas.
  - Se utilizan para la recepción de los datos que llegan con la llamada o para la emisión de resultados.
  - Aparecen declarados en la cabecera del subprograma entre paréntesis
  - Son fijos para cada subprograma.
- Parámetros reales o actuales:

```
Potencia (base,exponente)
Inicio
.....
.....
.....
FinP
```

Tipos de parámetros

- Parámetros reales o actuales:
  - Son **variables** o **constantes** u otros **objetos locales** (*), enviados en cada llamada desde el programa o subprograma que hace la llamada.
  - Éstos parámetros aparecen listados (escritos) en la llamada.

```
//Ejemplo 1
PROGRAMA PRINCIPAL
Inicio
.....
Potencia (bas,exp) //locales del PP
 .....
FinPP
```

```
//Ejemplo 2
MODULO_M (...)
Inicio
 .....
Potencia (bas,3)
 .....
FinMM
```

Correspondencia

- Entre parámetros reales y formales existe una correspondencia en cuanto al:
  - **tipo de dato**:

    Los tipos de datos de cada parámetro deberán ser iguales en la llamada y en la cabecera del subprograma llamado.
  - **Orden**:

    El flujo de comunicación se realiza en orden, del primer parámetro real al primero formal, del segundo real al segundo formal, etc..
  - **Número**:

    La cantidad de parámetros de llamada y cabecera del subprograma llamado debe ser la misma.

Paso de parámetros

- **Por valor**:
  - ***Se utiliza para suministrar datos de entrada*** a un subprograma.
  - El valor del parámetro real o actual ***se copia*** en el parámetro formal correspondiente.
  - Por tanto, el parámetro real no podrá ser modificado por el subprograma.
  - Por lo que el paso por valor ***evita que se produzcan cambios no intencionados en los parámetros actuales*** o reales
- **Paso por referencia**:

Paso de parámetros

- **Paso por referencia o *por variable***:
  - ***Se usa indistintamente para suministrar datos de entrada y recibir datos de salida***.
  - ***El subprograma recibe la referencia en memoria del parámetro actual* en el correspondiente parámetro formal**.
  - El subprograma utiliza la variable pasada como si fuera propia.
  - Por tanto, ***puede modificarla a conveniencia*** .
  - El paso por referencia debe evitarse en la medida de lo posible, puesto que puede producir efectos laterales, es decir, cambios indeseables en los parámetros reales. Esto ocurre en ME no así en MOO por su propia implementación.
  - Cuando un parámetro es pasado por referencia se pondrá delante de su declaración en la lista de parámetros y en el programa principal la palabra referencia o, simplemente, ***ref*.**

Parámetros: Resumen

Tipo de parámetro	Utilización
Actual o Real	Se encuentra en una sentencia de llamada a un subprograma. **Es pasado** por valor o por referencia a un parámetro formal.
Formal por valor	Se encuentra en la lista de parámetros en la cabecera de un subprograma y **recibe** una copia del valor almacenado en el parámetro actual correspondiente de la llamada al subprograma.
Formal por Referencia	Se encuentra en la lista de parámetros en la cabecera de un subprograma y **recibe** la referencia del parámetro actual correspondiente de la llamada al subprograma.

Procedimientos

- Son subprogramas internos o externos.
- ***Sin valor asociado al nombre*** (no significa que no dispongan de datos de salida al módulo que lo llama).
- Si son internos la declaración (o **prototipo**) y/o definición se hará en el mismo archivo del programa que lo usa.
- Si son externos su declaración (o **prototipo**) y/o definición se encuentra en un archivo, llamado en Java librería, que debe incluirse en el programa que lo use.

Procedimientos

- Formato de declaración:

  nombreProcedimiento (tipos_datos_parámetros)

- Formato de la llamada:

  nombreProcedimiento (lista_parámetros_reales)

- ¿Qué se diseña como procedimiento?: Los módulos que no estén obligados a devolver un valor explícito asociado a su nombre al programa principal o al subprograma que lo llama.

Funciones

- Son subprogramas internos o externos.
- ***Con valor asociado al nombre***.
- Si son internas la declaración (o **prototipo**) y/o definición se hará en el mismo archivo del programa que lo usa.
- Si son externas su declaración (o **prototipo**) y/o definición se encuentra en un archivo, llamado en Java librería, que debe incluirse en el programa que lo use.

Funciones

- Formato de declaración:

  tipo nombreFunción (tipos_datos_parámetros)

- Formato de la llamada:

  nombreVariable = nombreFunción (Lista_parámetros_reales)

- ¿Qué se diseña como función?: módulos que realicen cálculos y que produzcan unos resultados basados en ellos que son necesarios para el programa principal o subprograma que lo usa.

Diseño de interfaz

- La interfaz * está formada por:
  - Nombre del subprograma.
  - Tipo de dato que devuelve el subprograma.
  - Descripción de los parámetros entre paréntesis.
  - Características de los parámetros

- Para realizar la documentación interna de la función escribiremos delante del código de la función la interfaz:
  - Prototipo.
  - Breve comentario de lo que realiza.
  - Precondiciones
  - Entradas
  - Salidas
  - Entradas/salidas
  - postcondiciones

Para esto aplicaremos el siguiente método de trabajo 

Interfaz: método de trabajo

Desde el punto de vista del subprog.	Tipos de datos	Cómo se pasan
Necesidades	Datos de entrada	Parámetros pasados por **valor**. Excepto algunos objetos por propia naturaleza, (esto depende del lenguaje) por ejemplo, los arrays.
Devoluciones	Datos de salida	Parámetros pasados por **referencia** si el subprograma es un **procedimiento**, o **valor devuelto** asociado al nombre del subprograma si es una **función**.
Necesidades/ Devoluciones	Datos de entrada/salida	Parámetros pasados por **referencia**.

Interfaz: método de trabajo

- Restricciones:
  - Después de estudiado lo anterior se pensará si los parámetros de entrada, si los hubiera, deben cumplir alguna restricción.
  - Las restricciones, generarán código,
    - Pueden convertirse en precondiciones. En este caso, será el programa principal el responsable de su verificación.
    - Pueden NO convertirse en precondiciones. En cuyo caso, será el subprograma el responsable de su verificación.

¿Funciones o Procedimientos?

- Si el subprograma no tiene salidas o tiene más de una, es aconsejable utilizar un procedimiento.
- Si sólo tiene una salida, es recomendable usar una función.
- Si tiene una sola salida, que además es entrada, es un parámetro de entrada/salida y, por tanto, debe pasarse por referencia, se usará un procedimiento.
- Si para el diseño del subprograma ambos tipos son aceptables, se puede utilizar aquella que más guste al diseñador o seguir la política de la empresa.

Pasos para escribir un subprograma

- Si tuviéramos que realizar subprogramas independientemente del contexto en el que van a ser utilizados ¿cuáles serían los pasos a seguir?

1. Definir el problema que el subprograma ha de resolver.
2. Darle un nombre no ambiguo al subprograma (autodocumentado).
3. Estudiar la interfaz y escribir el prototipo del subprograma.
4. Decidir cómo se va a probar el funcionamiento del subprograma.
5. Buscar el algoritmo más adecuado para resolver el problema.
6. Si el algoritmo es sencillo basta con escribir los pasos principales como comentarios y a continuación rellenar el código correspondiente a cada comentario.
7. Si el algoritmo es complejo, se realizará el pseudocódigo generalizado y después se codificará en el lenguaje correspondiente.
8. Revisar, pasar la traza, al código.
9. Repetir los pasos anteriores hasta conseguir el diseño perfecto.

Recordamos:

- Definir cuidadosamente la interfaz de comunicación.
- Declarar los procedimientos y funciones antes de que se utilicen en una llamada.
- Comprobar los tipos de datos de los parámetros formales y reales.
- Verificar el emparejamiento de parámetros formales y reales, tanto en número como en orden.
- Verificar la llamada según sea procedimiento o función:
  - El nombre de una función deberá aparecer a la derecha de un símbolo de asignación.
  - El nombre de un procedimiento no puede aparecer a la derecha de un símbolo de asignación.

¿Cómo queda el diseño del algoritmo?

- **Análisis:**
  - **Todo lo incluido hasta ahora. +**
  - **Programa principal generalizado. +**
  - **Estudio y diseño de la interfaz de cada subprograma. +**
  - **Diagrama jerárquico de estructura de módulos y/o de descomposición funcional.**
- **Definiciones de subprogramas.**
- **Traza y revisión.**

Ver pág. 29

Ejercicio pág. 30

"Divide las dificultades que observes en tantas partes como sea posible, para su mejor solución".

DESCARTES, René

(Filósofo y matemático francés, (1596-1650))