Programación I: Algoritmos, Datos y Estructuras.

Ing. Ángel Vázquez. Msc.

MODULARIZACIÓN

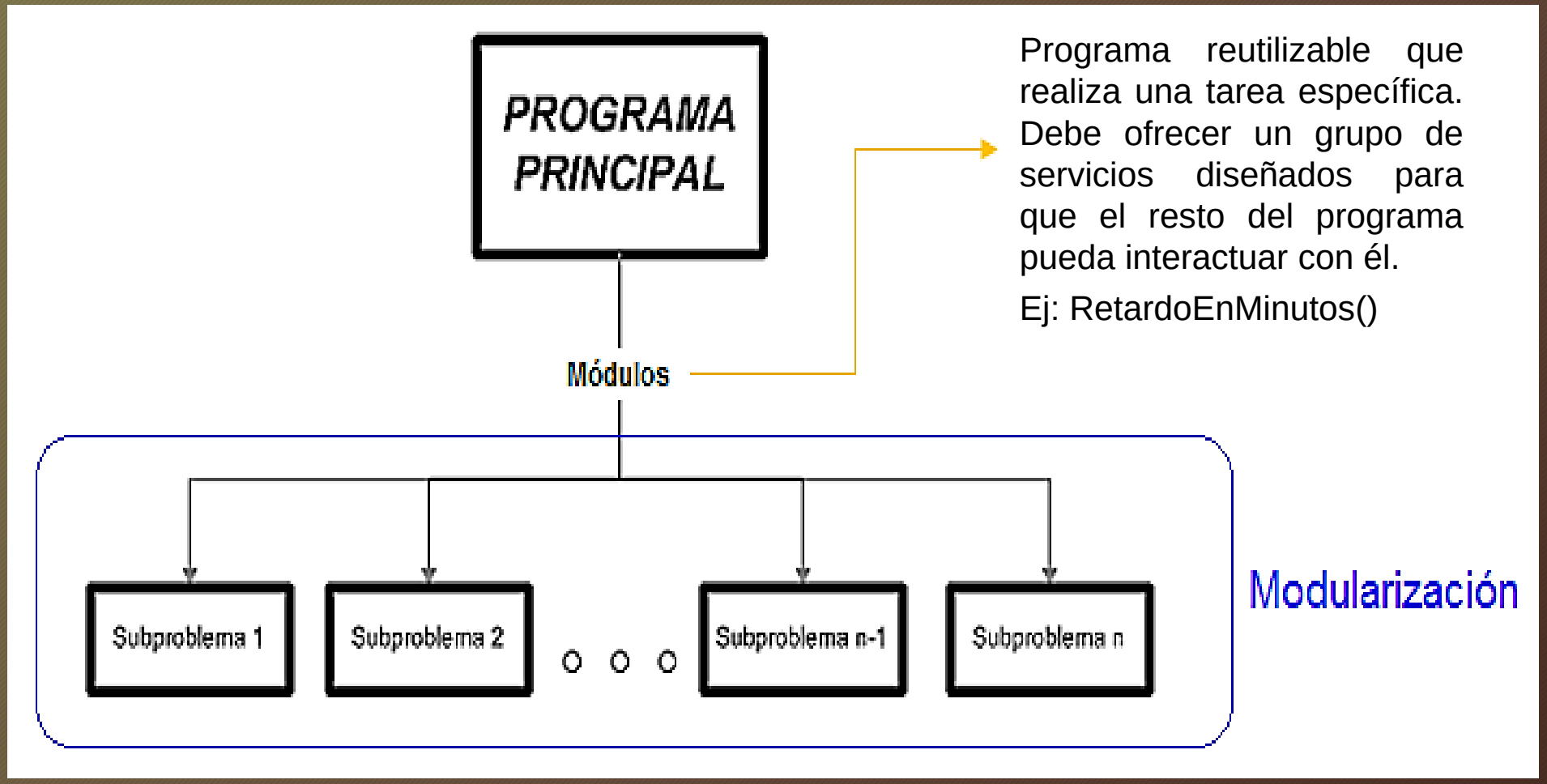
Angelo Pereira

Introducción

Programa complejo → ~~Diagrama de flujo~~ Modularizar

*Qué es **Modularizar**?*

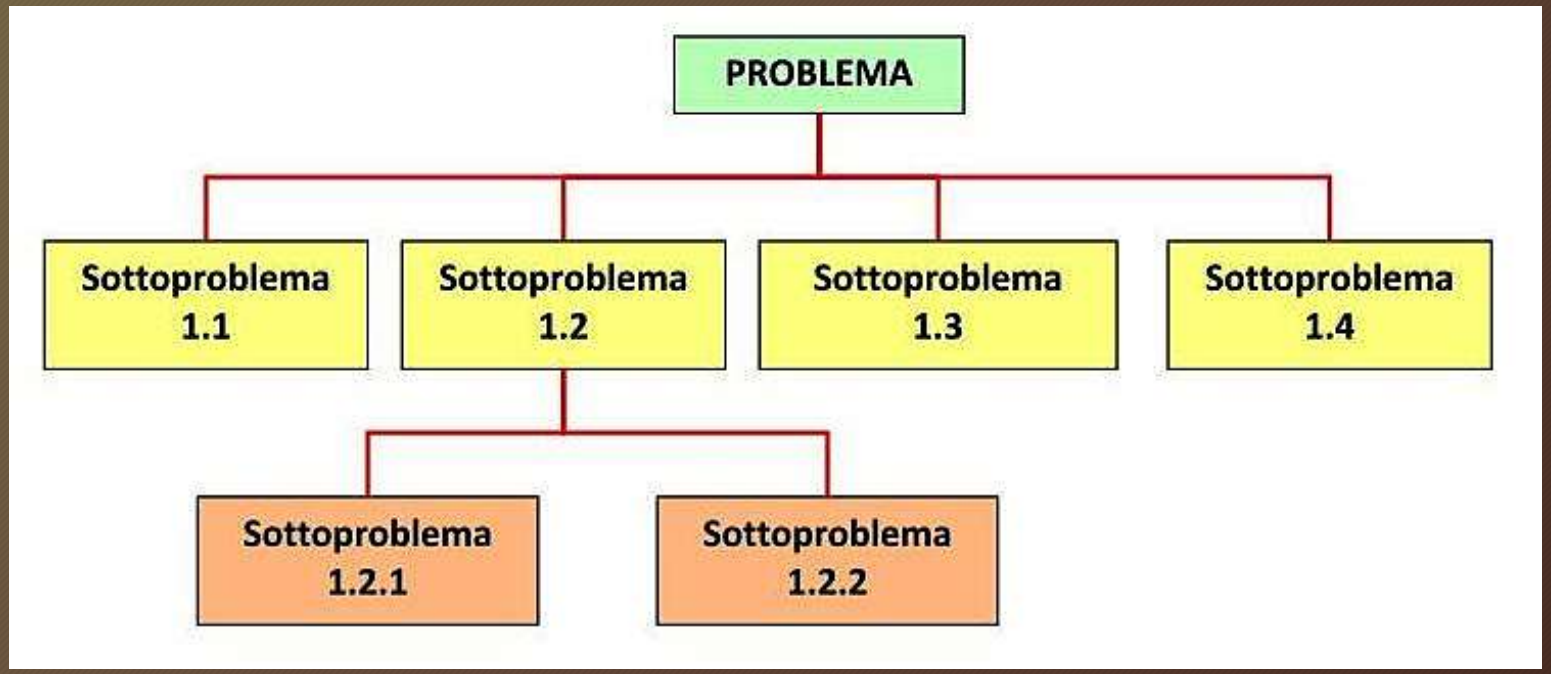
Proceso de dividir/descomponer un programa en diferentes módulos.



Paradigma de la programación modular.

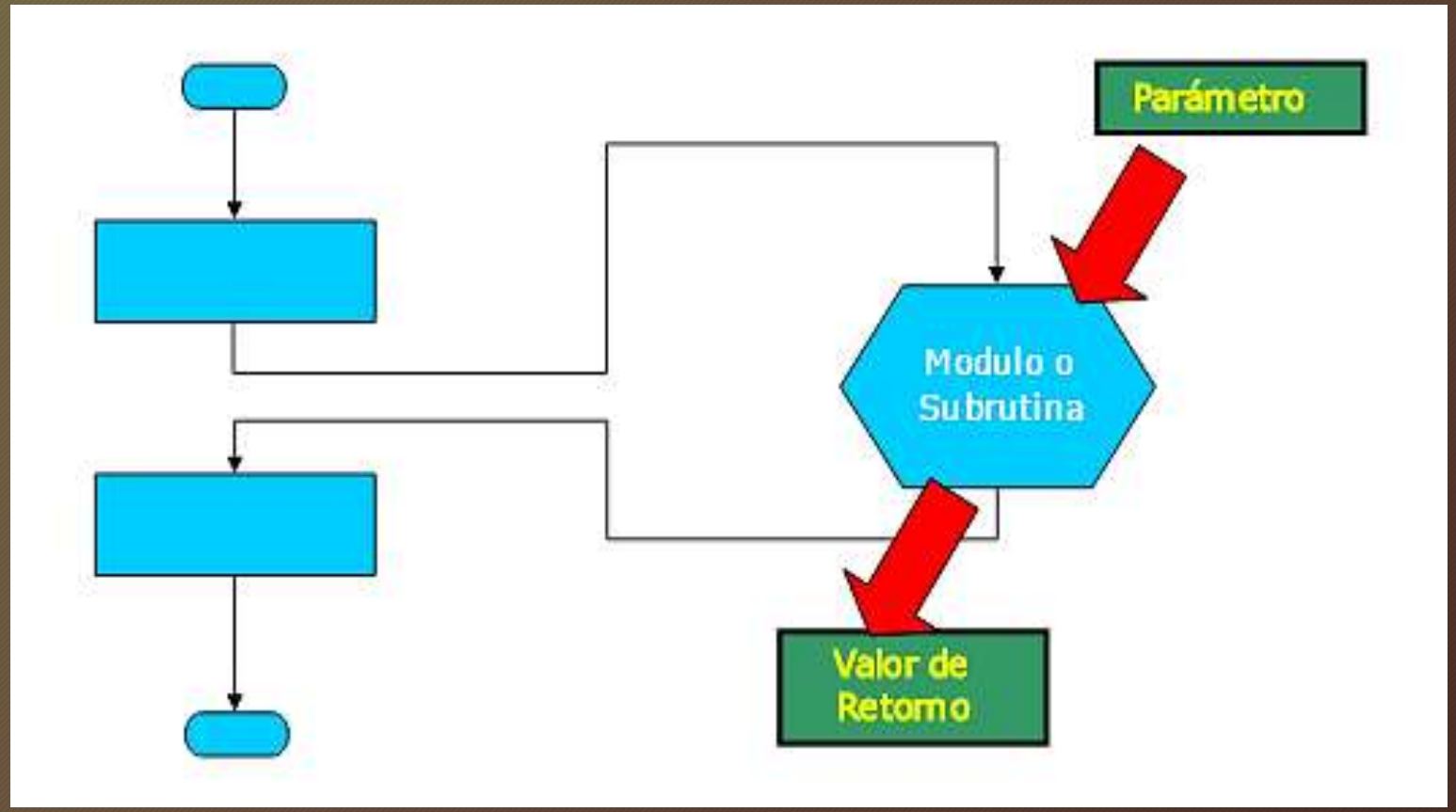
Y si la tarea asignada a un módulo es demasiado compleja?

Dividir en Subtareas

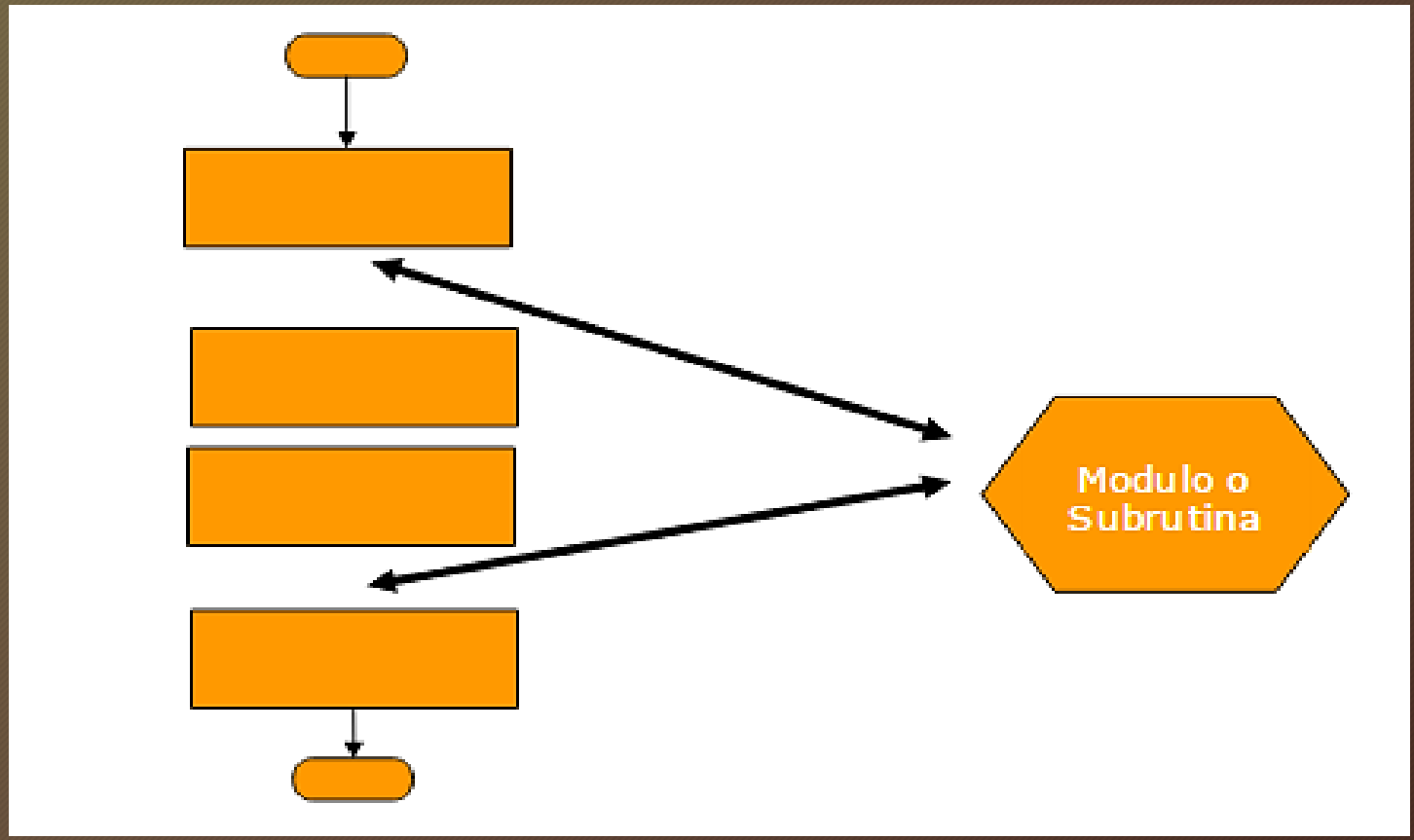


Diseño Top Down

Los módulos pueden recibir valores llamados parámetros y/o devolver resultados (*valor de retorno*).



Una subrutina puede ser invocada/llamada desde distintos lugares del programa.



Como los módulos pueden resolverse independientemente de los demás, las soluciones deben combinarse para resolver el problema original.

$$\cos x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n)!} x^{2n} \qquad \text{para todo } x$$

- Función Potencia
- Función División
- Función Factorial

Proceso de la Modularización

E*structuración y separación* de las tareas que realiza en métodos (*herramienta que ayuda a modularizar un programa*).

Fragmento de entidad o código auto-contenido que ejecuta una tarea específica.

Java: Método
C++: Función
Basic: Subrutina o procedimiento

En general, en cualquier lenguaje de programación (bajo o alto nivel), estos métodos se conocen como *Subrutinas*.

Ventajas de la Modularización

- Facilita su proceso de diseño, implementación y operación.
- Reduce el tamaño del código.
- Facilita la compresión (cada módulo puede ser estudiado separadamente).
- Facilita la corrección de errores y el mantenimiento del mismo (localización rápida).
- La modificación de un módulo no afecta a los demás.
- Reutilización del código. (Ej: retardos)

Modularización aplicada en la Vida Real

Jugar un Videojuego

Ejemplo .– Algoritmo para encontrar el mayor de tres números.

Algoritmo

```
Algoritmo numero_mayor

    Escribir("Ingrese 3 números: ")
    Leer(a, b, c)
    mayor = max(a, b, c)
    Escribir("El número mayor es ", mayor)

Fin Algoritmo
```

Método

```
Numero max(Numero num1, Numero num2, Numero num3)
    num_max = num1

    Si num2 > num_max Entonces
        num_max = num2
    Fin Si

       Si num3 > num_max Entonces
        num_max = num3
    Fin Si

    Retornar num_max
Fin Metodo
```

Métodos y Lenguajes de Programación

Lenguajes de programación → *librería de métodos*

Conjunto de procedimientos que permiten llevar a cabo cálculos matemáticos comunes, manipulación de texto y caracteres, operaciones de E/S, operaciones con bases de datos, operaciones de redes de computadoras, procesamiento de archivos, entre otras tareas.

“Un desarrollador puede escribir sus propios métodos según lo requiera el procedimiento diseñado para la solución del problema”

Declaración de un Método

Dado que podrían existir funciones declaradas de manera análoga que cumplen labores completamente diferentes o funciones incorrectamente declaradas cumpliendo tareas similares.

- Permite una rápida localización a futuro.
- Veloz identificación de la función específica que éste realiza dentro de un programa.
- Le permite al desarrollador *reutilizar* de manera coherente y sobretodo eficiente de código.

Tipo de dato de retorno

Nombre del método

Lista de parámetros

Encabezado del método

Tipo de Dato de Retorno

- Indica el resultado del método que será utilizado por el código en el que fue invocado.
- Ej: mostrará si es número entero, decimal, cadena de texto, etc.

Un método puede retornar sólo un o ningún valor

Si no retorna ningún valor:

- El método se limita a cumplir con una tarea específica sin que haya necesidad de que retorne un valor en específico, y no dará el tipo de valor de retorno.

Del ejemplo anterior, el tipo de retorno del método **max** indica que devolverá un valor numérico, posiblemente incluyendo una parte decimal.

Método

```
Numero max(Numero num1, Numero num2, Numero num3)
    num_max = num1

    Si num2 > num_max Entonces
        num_max = num2
    Fin Si

       Si num3 > num_max Entonces
        num_max = num3
    Fin Si

    Retornar num_max
Fin Metodo
```

Nombre del Método

Nombre que se utilizará para llamar o invocar al método que se está declarando.

- Debe expresar claramente la tarea que cumple (describir todo lo que hace el subprograma).
- Utilizar letras minúsculas y/o MAYÚSCULAS, números y _.
- Si es difícil encontrar un nombre conciso, dicho método intenta ejecutar demasiadas tareas.
  - ✓ Subdividir el método en partes más pequeñas que cumplan funciones específicas y fáciles de definir.

¡Prohibido!

- Utilizar caracteres especiales como (tildes, ñ, -, llaves, corchetes, ni empezar con número.

() [] {} ´ - 2 ñ

Lista de Parámetros

- Argumentos, datos de entrada (Información adicional) que recibe el método para operar con ellos en el momento de su invocación.
- Están delimitada por paréntesis.
- Cada parámetro es suministrado como una variable que puede ser referenciada y utilizada en el cuerpo del método.
- Puede contener uno o más parámetros separados por comas, o no contener parámetros según el diseño del método.
- Si la lista de parámetros está vacía, significa que el método declarado no requiere ninguna información adicional para ejecutar su tarea.

Del ejemplo anterior, la lista de parámetros del método max está conformada por tres variables numéricas *(que posiblemente incluirán una parte decimal)* llamadas num1, num2 y num3.

Método

```
Numero max(Numero num1, Numero num2, Numero num3)
    num_max = num1

    Si num2 > num_max Entonces
        num_max = num2
    Fin Si

       Si num3 > num_max Entonces
        num_max = num3
    Fin Si

    Retornar num_max
Fin Metodo
```

El Cuerpo de un Método

- Definido por un conjunto de *(una o más)* instrucciones que se escriben y ejecutan dentro del método para cumplir con una tarea específica.
- La combinación declaración y cuerpo del método conforman una ***entidad o código auto-contenido.***

Información y operaciones que se ejecutan dentro del método que son sólo conocidas y relevantes para tal método debido a que ninguna otra parte del programa puede acceder a ellas de forma directa.

Claramente podemos observar la tarea específica que cumple el método max dentro del algoritmo *(encontrar el mayor de tres números)* y como las operaciones que se ejecutan dentro del mismo son únicamente conocidas por el método.

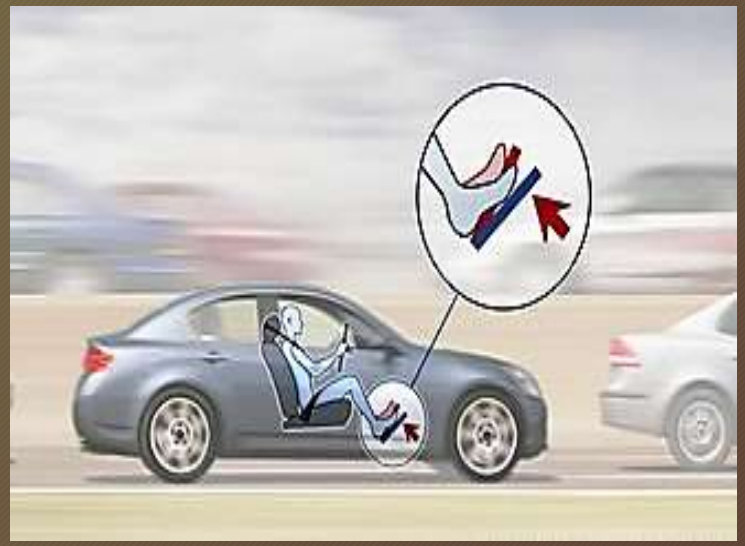
Método

```
Numero max(Numero num1, Numero num2, Numero num3)
    num_max = num1

    Si num2 > num_max Entonces
        num_max = num2
    Fin Si

       Si num3 > num_max Entonces
        num_max = num3
    Fin Si

    Retornar num_max
Fin Metodo
```

Ejemplo de la vida real

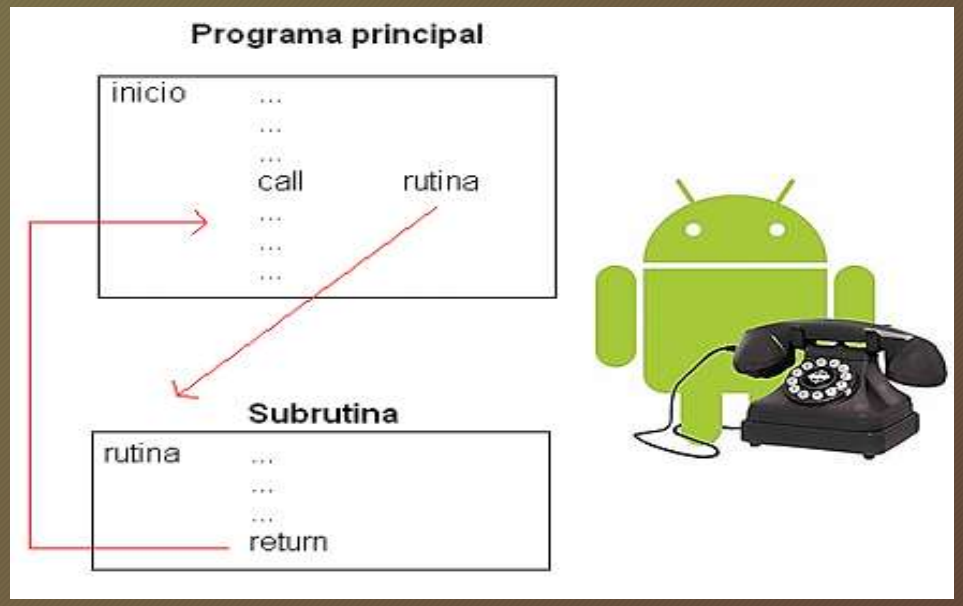
Aceleración de un vehículo

- La forma de invocar este método sería ejerciendo presión sobre el pedal de aceleración. Un ejemplo de parámetro (argumentos) en este caso sería:



- Cantidad de aceleración que se requiere.
- Información adicional que es proporcionada por la persona que presiona el acelerador (ligeramente o con fuerza).

Invocación de un Método



- Acción de llamar a un método para que éste se ejecute y poder acceder a su funcionalidad.
- Cuando es invocado es necesario proporcionarle un valor (argumentos) para cada uno de los parámetros del método.

- Es realizado desde el interior de otro método y cuando el método invocado termine su ejecución, puede devolver o retornar un resultado.

Ejemplos de la vida real

Gestiones bancarias

Asumiendo la existencia de un método para el registro de un depósito de dinero en una cuenta bancaria, es probable que cuando se invoque a este método se le deba proporcionar el *número de cuenta* y el *monto* del depósito como argumentos.

Ejemplos de la vida real

Gestión jerárquica de una empresa manufacturera.

- El jefe departamental *(representaría el método que invoca*) solicita a un obrero *(el método que es invocado)* realizar una tarea e informar su resultado luego de que la haya terminado *(retorno)*.

- El jefe no necesita saber la forma en que el obrero ejecutó la tarea asignada, ya que el obrero pudo haber llamado por ejemplo a otros obreros *(invocar otros métodos) sin que el jefe se haya enterado.*

Es decir…

En PROGRAMACIÓN, existe una relación jerárquica entre los diferentes métodos que componen un programa, de tal forma que el programa divide sus tareas entre varios métodos, en donde, cada método se encarga de realizar subtareas y de sus detalles de implementación propios.

Almacenamiento en memoria de los Argumentos

- Un argumento puede consistir en valor concreto o en una variable (su nombre puede ser el mismo o diferente al nombre del parámetro.

Esto se debe a que tanto el *argumento* como el correspondiente *parámetro* se almacenan como *variables separadas* en la *memoria de la computadora*. Ver fig.

¡Importante!

El orden, el número y el tipo de dato de los *valores* o *variables* proporcionados como *argumentos* en la invocación de un método, deben corresponder exactamente a la lista de parámetros en la declaración del método.

```
;***************** PROGRAMA PRINCIPAL ****************************

PRINCIPAL:

        movlw   01
        movwf   PORTB       ;Enciende el Led
        call    TIEMPO
        movlw   00
        movwf   PORTB       ;Apaga el Led
        call    TIEMPO
        goto    PRINCIPAL

;***************** SUBRUTINA DE RETRAZO DE TIEMPO ***************

TIEMPO:     clrf    RETRA1      ;
            clrf    RETRA2

BUCLE:      decfsz  RETRA1,1
            goto    BUCLE
            decfsz  RETRA2,1
            goto    BUCLE
            return
        END
```

Resumen

- Reducir la complejidad del programa (“*Divide y vencerás”).*
- Cohesión y acoplamiento (principios usados para medir si los módulos de un diseño fueron bien divididos).
- Pasos para escribir un Subprograma:
  - ✓ Definir el problema que el subprograma ha de resolver.
  - ✓ Darle un nombre no ambiguo al subprograma.
  - ✓ Decidir cómo se puede probar el funcionamiento del subprograma.

Resumen

- ✓ Escribir la declaración del subprograma (cabecera de la función).
- ✓ Buscar el algoritmo más adecuado para resolver el problema.
- ✓ Escribir los pasos principales del algoritmo como comentarios.
- ✓ Rellenar el código correspondiente a cada comentario.
- ✓ Revisar mentalmente cada fragmento de código.
- ✓ Repetir los pasos anteriores hasta quedar completamente satisfecho.

Bibliografía

- Plataforma MOOC. Universidad de Cuenca: UDC003 Fundamentos de Programación I – Parte II.
- Diseño Estructurado de Algoritmos. *P. L. I. Flores Valerio Carlos Augusto.*
- Curso de programación y Análisis de Software. *Arias, Ángel*; *Durango, Alicia*. 2da edición. Pág. 231.
- http://slideplayer.es/slide/4011953/
- http://elvex.ugr.es/decsai/c/apuntes/modularization.pdf

GRACIAS