

# Laboratorio de Computación I



# Recursividad



## **Recursividad**

**Recursividad es el proceso de definir algo en términos de sí mismo.**

- **Se dice que una función es recursiva cuando se llama a sí misma.** El lenguaje C permite que una función pueda llamarse a sí misma. A esta característica se le denomina recursividad.
- **La recursividad es distinto de la iteración**
- **La recursividad es una alternativa para ejecutar estructuras de iteración**



## Recursividad -Imágenes



## **Función Recursiva**

Se componen de

**1- Caso Base:** Una solución simple para un caso en particular (puede haber mas de un caso base).

**2- Caso Recursivo:** una solución que involucra utilizar la función original, con parámetros que se acercan mas al caso base. Los pasos que sigue el caso recursivo son los siguientes:

- a. El procedimiento se llama a si mismo.
- b. El problema se resuelve, resolviendo el mismo problema pero de tamaño menor.
- c. La manera en la cual el tamaño del problema disminuye asegura que el caso base eventualmente se alcanzará.

# *Lenguaje de Programación C*

Escribe un programa que calcule el factorial (!) de un entero no negativo.

Ejemplo:

$0! = 1$

$1! = 1$

$2! = 2 \rightarrow 2! = 2 * 1!$

$3! = 6 \rightarrow 3! = 3 * 2!$

$4! = 24 \rightarrow 4! = 4 * 3!$

$5! = 120 \rightarrow 5! = 5 * 4!$

*Iterativo*

```
int factorial (int n) {  
    int fact = 1;  
    for (int i = 1; i <= n; i++)  
        fact *= i;  
    return fact;  
}
```

*Recursivo*

```
int factorial (int n) {  
    if n == 0 return 1;  
    else  
        return factorial (n-1) * n;  
}
```

## Función Recursiva: Ejemplo – Factorial

### Solución

Aquí se muestra la secuencia que toma el factorial

$$N! = \begin{cases} 1 & \text{si } N = 0 \text{ (base)} \\ N * (N - 1)! & \text{si } N > 0 \text{ (recursión)} \end{cases}$$

El razonamiento recursivo tiene dos partes: la **base** y la **regla recursiva de construcción**.

La base NO es recursiva y es el punto de partida como de terminacion de la definicion.

```
#include <stdio.h>
#include <conio.h>
```

```
long int factorial(int n);
```

```
Int main ()
```

```
{
    int valor;
    long int result;
    printf("\nIngresa numero:");
    scanf("%d",&valor);
    if (valor < 0) printf("Error");
    else
    {
        result=factorial(valor);
        printf("\nEl factorial de %d es %ld",valor,result);
    }
}
```

```
getch();
return 0;
```

```
}
long int factorial(int n)
```

```
{
    long int resp;
    if(n==1 || n==0)
        return 1;
    else
        resp=n* factorial(n-1);
    return (resp);
}
```



Diagram illustrating the recursive nature of the factorial function. An arrow points from the recursive call `factorial(n-1)` in the code to a yellow box containing the text: **RECURSIVIDAD - factorial se llama a si misma**.



# ¿Recursión o iteración?

- Ventajas de la Recursión ya conocidas
  - Soluciones simples, claras.
  - Soluciones elegantes.
  - Soluciones a problemas complejos.
- Desventajas de la Recursión: INEFICIENCIA
  - Sobrecarga asociada con las llamadas a subalgoritmos
    - Una simple llamada puede generar un gran numero de llamadas recursivas. (Fact(n) genera n llamadas recursivas)
    - ¿La claridad compensa la sobrecarga?
    - El valor de la recursividad reside en el hecho de que se puede usar para resolver problemas sin fácil solución iterativa.
  - La ineficiencia inherente de algunos algoritmos recursivos.

# ¿Recursión o iteración?

LA RECURSIVIDAD SE DEBE USAR  
CUANDO SEA REALMENTE  
NECESARIA, ES DECIR, CUANDO NO  
EXISTA UNA SOLUCIÓN ITERATIVA  
SIMPLE.