

# Informática II

Recursividad

# ¿Qué es recursividad?

- ▮ Que puede repetirse.
- ▮ La recursividad es un concepto fundamental en matemáticas y en computación.
- ▮ Es una alternativa diferente para implementar estructuras de repetición (ciclos). Los módulos se hacen llamadas recursivas.
- ▮ Se puede usar en toda situación en la cual la solución pueda ser expresada como una secuencia de movimientos, pasos o transformaciones gobernadas por un conjunto de reglas no ambiguas.

# Función recursiva

Se compone de:

- ▮ **Caso base:** una solución simple para un caso particular (puede haber más de un caso base). Es importante determinar un caso base, es decir un punto en el cual existe una condición por la cual no se requiera volver a llamar a la misma función.
- ▮ **Caso recursivo:** una solución que involucra volver a utilizar la función original, con parámetros que se acercan más al caso base. Los pasos que sigue el caso recursivo son los siguientes:
  1. La función se llama a sí misma
  2. El problema se resuelve, tratando el mismo problema pero de tamaño menor
  3. La manera en la cual el tamaño del problema disminuye asegura que el caso base eventualmente se alcanzará.

# Ejemplo: Calculo del factorial

▢ Calcule el factorial (!) de un entero no negativo.

0!	=1	=1	=1
1!	=1 * 0!	=1 * 1	=1
2!	=2 * 1!	=2 * 1	=2
3!	=3 * 2!	=3 * 2 * 1	=6
4!	=4 * 3!	=4 * 3 * 2 * 1	=24
5!	=5 * 4!	=5 * 4 * 3 * 2 * 1	=120
N!	N * (N - 1)!		

▢ El factorial de un número es la multiplicación de cada número desde 1 hasta ese número, entonces es muy sencillo crear un ciclo de 1 hasta el número pedido para hacer el cálculo.

# Factorial (sin recursividad)

```
1 #include<iostream>
2 #include<locale.h>
3
4 using namespace std;
5
6
7 int main(){
8     int i,num,resultado=1;
9     cout << "Calculo del factorial." << endl;
10    cin >> num;
11
12    for(i=1; i<=num; i++)
13    {
14        resultado = resultado * i;
15    }
16    cout << "El factorial de "<<num<<" es: " << resultado;
17    return 0;
18 }
19 }
```

Uso de la  
estructura  
repetitiva FOR

# Factorial (con recursividad)

Para el calculo de N!

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

The diagram illustrates the recursive definition of the factorial function. It features a large green curly brace on the left side of the equation. The first case, '1 si N = 0', is enclosed in a dashed blue box, with a blue oval callout to its right labeled 'Caso base'. The second case, 'N \* (N - 1)! si N > 0', is enclosed in a dashed orange box, with an orange oval callout to its right labeled 'Caso Recursivo'.

# Factorial (con recursividad)

```
1  #include<iostream>
2  #include<locale.h>
3
4  using namespace std;
5
6  int calcularFactorial(int);
7
8  int main(){
9
10     int num=0;
11
12     cout<<"Introduce un numero para calcular el factorial: ";
13     cin>>num;
14     cout<<"El factorial de "<<num<<" es: "<<calcularFactorial(num);
15     return 0;
16
17 }
18
19 int calcularFactorial(int num)
20 {
21     if(num==0)
22         return 1;
23
24     else
25         return num * calcularFactorial(num-1);
26 }
27
28 }
```

Caso base

Se llama así misma

Caso Recursivo

# ¿Por qué usar recursividad?

- ▮ Son mas cercanos a la descripción matemática.
- ▮ Permiten simplificar el código.
- ▮ Generalmente mas fáciles de analizar.
- ▮ Se adaptan mejor a las estructuras de datos recursivas.
- ▮ Los algoritmos recursivos ofrecen soluciones estructuradas, modulares y elegantemente simples.

# ¿Cuándo no usar recursividad?

- ▮ Cuando las funciones usen arreglos largos.
- ▮ Cuando las funciones cambian de manera impredecible.
- ▮ Cuando las iteraciones sean la mejor opción.



# Ejemplo: Calculo de la Serie de Fibonacci

□ Los valores son:

0	1	1	2	3	5	8	13	21
n1	n2	n3	n4	n5	n6	n7	n8	n9

Elementos definidos al comienzo

□ Comienza con un 0, luego un 1 y a partir de ahí cada término de la serie suma los 2 anteriores. Fórmula recursiva:

$$\text{fib}(n) = \text{fib}(n - 1) + \text{fib}(n - 2)$$

$$\text{fib}(3) = \text{fib}(3-1) + \text{fib}(3-2)$$

$$= \text{fib}(2) + \text{fib}(1)$$

$$= 1 + 0$$

# Ejemplo: Calculo de la Serie de Fibonacci

```
1  #include<iostream>
2  #include<locale.h>
3
4  using namespace std;
5
6  #define n1 0
7  #define n2 1
8
9  int calcularFibonacci(int);
10
11 int main(){
12     int num=0, i;
13     cout<<"Introduce cuantos numeros de la serie Fibonacci desea ver: ";
14     cin>>num;
15     for(i=1;i<=num;i++)
16     {
17         cout<<calcularFibonacci(i)<<" ";
18     }
19
20     return 0;
21 }
22
23
24 int calcularFibonacci(int num)
25 {
26     if (num==1)
27         return n1;
28     else if (num==2)
29         return n2;
30
31     else
32         return calcularFibonacci(num-1) + calcularFibonacci(num-2);
33 }
34 }
```

Casos bases

Caso  
Recursivo

# Recursión vs iteración

## Repetición

- ✓ Iteración: ciclo explícito (se expresa claramente)
- ✓ Recursión: repetidas invocaciones a una función

## Terminación

- ✓ Iteración: el ciclo termina o la condición del ciclo falla
- ✓ Recursión: se reconoce el o los casos bases

En ambos casos podemos tener ciclos infinitos