

GRUPO 3

Prolog 3

# Teoría de Lenguajes de Programación

Sara Acevedo Maya

[saacevedom@unal.edu.co](mailto:saacevedom@unal.edu.co)

Universidad Nacional de Colombia



UNIVERSIDAD  
**NACIONAL**  
DE COLOMBIA

# Contenido

01/ Recursividad  
02/ Tail Recursion  
03/ Listas



UNIVERSIDAD  
**NACIONAL**  
DE COLOMBIA



# 01 - Recursividad



UNIVERSIDAD  
**NACIONAL**  
DE COLOMBIA



# Recursividad

La recursividad es cuando una función se llama a sí misma para resolver un problema.

De una forma mas técnica, se puede decir que es la forma en la cual se especifica un proceso basado en su propia definición.



UNIVERSIDAD  
**NACIONAL**  
DE COLOMBIA



# EJEMPLO

**masgrande(elefante, caballo).**  
**masgrande(caballo, perro).**  
**masgrande(perro, raton).**  
**masgrande(raton, hormiga).** ←

**muchomasgrande(A, B) :- masgrande(A, B).**  
**muchomasgrande(A, C) :- masgrande(A, B), masgrande(B, C).**  
**muchomasgrande(A, D) :- masgrande(A, B), masgrande(B, C), masgrande(C, D).**  
**muchomasgrande(A, E) :-**  
**masgrande(A, B), masgrande(B, C), masgrande(C, D), masgrande(D, E).** ←





# EJEMPLO

**masgrande(elefante, caballo).**  
**masgrande(caballo, perro).**  
**masgrande(perro, raton).**  
**masgrande(raton, hormiga).**

**muchomasgrande(A, B) :- masgrande(A, B).** ← 1ra Regla  
**muchomasgrande(A, B) :- masgrande(A, X), muchomasgrande(X, B).**  
← 2da Regla





# EJEMPLO

1era  
Comprobación

`muchomasgrande(elefante,caballo).`

**A** **B**  
`masgrande(elefante, caballo).`

`masgrande(caballo, perro).`

`masgrande(perro, raton).`

`masgrande(raton, hormiga).`

`muchomasgrande(A, B) :- masgrande(A, B).` ✓

`muchomasgrande(A, B) :- masgrande(A, X), muchomasgrande(X, B).`



UNIVERSIDAD  
**NACIONAL**  
DE COLOMBIA



# EJEMPLO

## 2da Comprobación

`muchomasgrande(elefante , perro).`

**A**  
`masgrande(elefante, caballo).`

**B**  
`masgrande(caballo, perro).`

`masgrande(perro, raton).`

`masgrande(raton,hormiga).`

`muchomasgrande(A, B) :- masgrande(A,B).`

`muchomasgrande(A, B) :- masgrande(A,X),muchomasgrande(X,B).`



UNIVERSIDAD  
**NACIONAL**  
DE COLOMBIA



# EJEMPLO

**Nota:** prolog determina que el único objeto que comparte la relación masgrande() con elefante, es caballo, por lo cual asigna a Caballo la variable X

2da  
Comprobación

muchomasgrande(elefante , perro).

A

X

**masgrande(elefante, caballo).**

X

B

**masgrande(caballo, perro).**

**masgrande(perro, raton).**

**masgrande(raton,hormiga).**

**muchomasgrande(A, B) :- masgrande(A,B).**

**muchomasgrande(A, B) :- masgrande(A,X),muchomasgrande(X,B).**





# EJEMPLO

El **factorial** de un número entero no negativo, denotado como  **$n!$** , es el producto de todos los enteros positivos menores o iguales a  $n$

```
def factorial(n):  
    if n == 0:  
        return 1  
    else:  
        return n * factorial(n-1)
```

**factorial(4)**

El **factorial** en PROLOG

```
factorial(0, 1) :- !.  
factorial(N, Resultado) :-  
    N > 0,  
    N1 is N - 1,  
    factorial(N1, Subresultado),  
    Resultado is N * Subresultado.
```



# 02 - Tail Recursion



UNIVERSIDAD  
**NACIONAL**  
DE COLOMBIA



# TAIL Recursion

Es un llamado recursivo que se hace al **final** de la funcion.  
**Siempre** es **mas eficiente** que la recursion normal.

A=[1,2,3,4,5,6,7,8]

```
sumVector(A,8)
[ 8 + SumVector(A,7) ]
[ 8 + [ 7 + SumVector(A,6) ] ]
[ 8 + [ 7 + [ 6 + SumVector(A,5) ] ] ]
[ 8 + [ 7 + [ 6 + [ 5 + SumVector(A,4) ] ] ] ]
[ 8 + [ 7 + [ 6 + [ 5 + [ 4 + SumVector(A,3) ] ] ] ] ]
[ 8 + [ 7 + [ 6 + [ 5 + [ 4 + [ 3 + SumVector(A,2) ] ] ] ] ] ]
[ 8 + [ 7 + [ 6 + [ 5 + [ 4 + [ 3 + [ 2 + SumVector(A,1) ] ] ] ] ] ] ]
[ 8 + [ 7 + [ 6 + [ 5 + [ 4 + [ 3 + [ 2 + [ 1 ] ] ] ] ] ] ] ] ]
[ 8 + [ 7 + [ 6 + [ 5 + [ 4 + [ 3 + [ 3 ] ] ] ] ] ] ] ] ]
[ 8 + [ 7 + [ 6 + [ 5 + [ 4 + [ 6 ] ] ] ] ] ] ] ] ]
[ 8 + [ 7 + [ 6 + [ 5 + [ 10 ] ] ] ] ] ] ] ] ]
[ 8 + [ 7 + [ 6 + [ 15 ] ] ] ] ] ] ] ] ]
[ 8 + [ 7 + [ 21 ] ] ] ] ] ] ] ] ]
[ 8 + [ 28 ] ] ] ] ] ] ] ] ]
36
```

B=[1,2,3,4,5,6,7,8]

```
sumVector(B,8,0)
[ sumVector(B,7,0+8) ]
[ sumVector(B,6,8+7) ]
[ sumVector(B,5,15+6) ]
[ sumVector(B,4,21+5) ]
[ sumVector(B,3,26+4) ]
[ sumVector(B,2,30+3) ]
[ sumVector(B,1,33+2) ]
[ sumVector(B,0,35+1) ]
36
```





# Ejemplo

**% Definición del predicado factorial/2**

**factorial(N, Resultado) :-**

**factorial\_aux(N, 1, Resultado).**

**% Caso base: factorial de 0 es 1**

**factorial\_aux(0, Acumulador, Acumulador).**

**% Caso recursivo**

**factorial\_aux(N, Acumulador, Resultado) :-**

**N > 0,**

**NuevoAcumulador is N \* Acumulador,**

**NuevoN is N - 1,**

**factorial\_aux(NuevoN, NuevoAcumulador,  
Resultado).**

**factorial(5, Resultado).**

**Definición del predicado factorial/2:**

El predicado factorial/2 toma dos argumentos: N (el número del cual queremos calcular el factorial) y Resultado (donde almacenaremos el resultado del cálculo del factorial). Este predicado simplemente llama a factorial\_aux/3 con N y un acumulador inicializado en 1.

**Caso base:**

El caso base se activa cuando N es igual a 0. En este caso, el resultado del factorial es 1, por lo que el predicado factorial\_aux/3 devuelve el valor del acumulador en el tercer argumento (Resultado).

**Caso recursivo:**

En el caso recursivo, cuando N es mayor que 0, se realiza el cálculo del factorial. Primero, se multiplica el valor actual de N por el acumulador actual. Luego, se decrementa N en 1 y se llama recursivamente a factorial\_aux/3 con el nuevo valor de N, el nuevo valor del acumulador y el mismo Resultado.

**Recursión en cola:**

La recursión se realiza en cola, lo que significa que la llamada recursiva es la última operación realizada dentro del cuerpo del predicado. Esto permite que el Prolog optimice la recursión, evitando un crecimiento excesivo de la pila de llamadas. En cada paso recursivo, se actualiza el valor del acumulador y se reduce N, asegurando que el cálculo del factorial se realice eficientemente.



UNIVERSIDAD  
**NACIONAL**  
DE COLOMBIA



# Recursión Normal vs Tail Recursion

Aspecto	Recursividad Normal	Recursión de Cola
Definición	Una función se llama a sí misma en cualquier punto.	La llamada recursiva es la última acción en la función.
Uso de la memoria	Cada llamada recursiva agrega una nueva pila de memoria.	No acumula pila de llamadas, utiliza memoria constante.
Eficiencia	Puede ser menos eficiente debido al uso de memoria adicional.	Más eficiente en términos de memoria y tiempo de ejecución.
Optimización automática	No se puede optimizar automáticamente en muchos lenguajes.	Algunos compiladores pueden optimizar automáticamente las llamadas de cola.





# 03 - Listas



UNIVERSIDAD  
**NACIONAL**  
DE COLOMBIA



# Manejo de listas

En prolog las listas se pueden ver como una serie de **cabezas** y **colas**.

Cada lista se puede dividir conceptualmente en dos partes: la "cabeza" (head) y la "cola" (tail).

**La cabeza es el primer elemento de la lista, y la cola es una lista de los elementos restantes.**


Normalmente se manejan con **recursividad**.



UNIVERSIDAD  
**NACIONAL**  
DE COLOMBIA



# EJEMPLO



SWISH

File Edit Examples Help

Program Program +

```
1 mostrarLista([]).  
2  
3 mostrarLista([Cabeza|Resto]) :-  
4 write(Cabeza) , nl , mostrarLista(Resto).  
5
```

**mostrarLista([]):- !.**

**mostrarLista([Cabeza|Resto]) :-**

**write(Cabeza),nl,mostrarLista(  
Resto).**

**mostrarLista([2,4,6,8,20]).**



UNIVERSIDAD  
**NACIONAL**  
DE COLOMBIA



# EJEMPLO

## 1. **mostrarLista([]):- !.**

- Esta regla define el caso base. Cuando la lista está vacía ([]), el predicado **mostrarLista/1** se satisface y termina. El corte (!) evita que Prolog busque más soluciones después de haber encontrado esta.

## 2. **mostrarLista([Cabeza | Resto]) :-**

**write(Cabeza),nl,mostrarLista(Resto):.**

- Esta regla se aplica cuando la lista tiene al menos un elemento. Se divide la lista en dos partes: la cabeza (**Cabeza**), que es el primer elemento de la lista, y el resto de la lista (**Resto**), que son los elementos restantes después de la cabeza.
- **write(Cabeza)** imprime el primer elemento de la lista.
- **nl** agrega un salto de línea después de imprimir el primer elemento, para que cada elemento se imprima en una nueva línea.
- **mostrarLista(Resto)** es una llamada recursiva que muestra el resto de la lista. Esto significa que se aplica la misma regla (**mostrarLista/1**) al resto de la lista.
- La recursión continúa hasta que se llega al caso base, donde la lista está vacía, momento en el que la ejecución termina.

**mostrarLista([]):- !.**

**mostrarLista([Cabeza | Resto]) :-**

**write(Cabeza),nl,mostrarLista(Resto).**

**mostrarLista([2,4,6,8,20]).**



UNIVERSIDAD  
**NACIONAL**  
DE COLOMBIA



# EJEMPLO

## 1. Caso Base (`contar_elementos([], 0)`):

Cuando la lista está vacía (`[]`), significa que no hay más elementos que contar, por lo que el contador es simplemente 0. Este es el caso base de la recursión.

## 2. Caso Recursivo (`contar_elementos([_ | Resto], Contador)`):

En este caso, tenemos una lista que tiene al menos un elemento, representado por `[Cabeza | Resto]`. Aquí, **Cabeza** es el primer elemento de la lista, y **Resto** es el resto de la lista sin el primer elemento. Usamos el guion bajo (`_`) para indicar que no nos importa el valor de **Cabeza**, solo queremos seguir contando los elementos restantes.

## 3. Llamada Recursiva (`contar_elementos(Resto, ContadorResto)`):

Después de tomar el primer elemento, llamamos recursivamente a **contar\_elementos** con el resto de la lista, es decir, **Resto**. Esto nos permite reducir el tamaño del problema a uno más pequeño, es decir, contar los elementos del resto de la lista. El contador resultante de esta llamada recursiva se almacena en **ContadorResto**.

## 4. Cálculo del Contador (`Contador is ContadorResto + 1`):

Una vez que hemos contado los elementos en el **Resto** de la lista, incrementamos el contador en 1 para tener en cuenta el primer elemento (**Cabeza**) que hemos contado en este paso. Esto se hace sumando 1 al contador del **Resto** y almacenándolo en **Contador**.

**`contar_elementos([], 0).`**

**`contar_elementos([_ | Resto], Contador) :-`**

**`contar_elementos(Resto, ContadorResto), Contador is ContadorResto + 1.`**

`contar_elementos([a, b, c], Contador).`



UNIVERSIDAD  
**NACIONAL**  
DE COLOMBIA

# TAREA

(recordar hacer la entrega del archivo  
apellido\_nombre\_grupoX.pl  
antes del domingo 17 a las 12 de la noche)

<https://forms.gle/fLNKD28ooHMnfPSCA>



UNIVERSIDAD  
**NACIONAL**  
DE COLOMBIA