Funciones Recursivas

¿Qué es la recursividad?

- *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.*

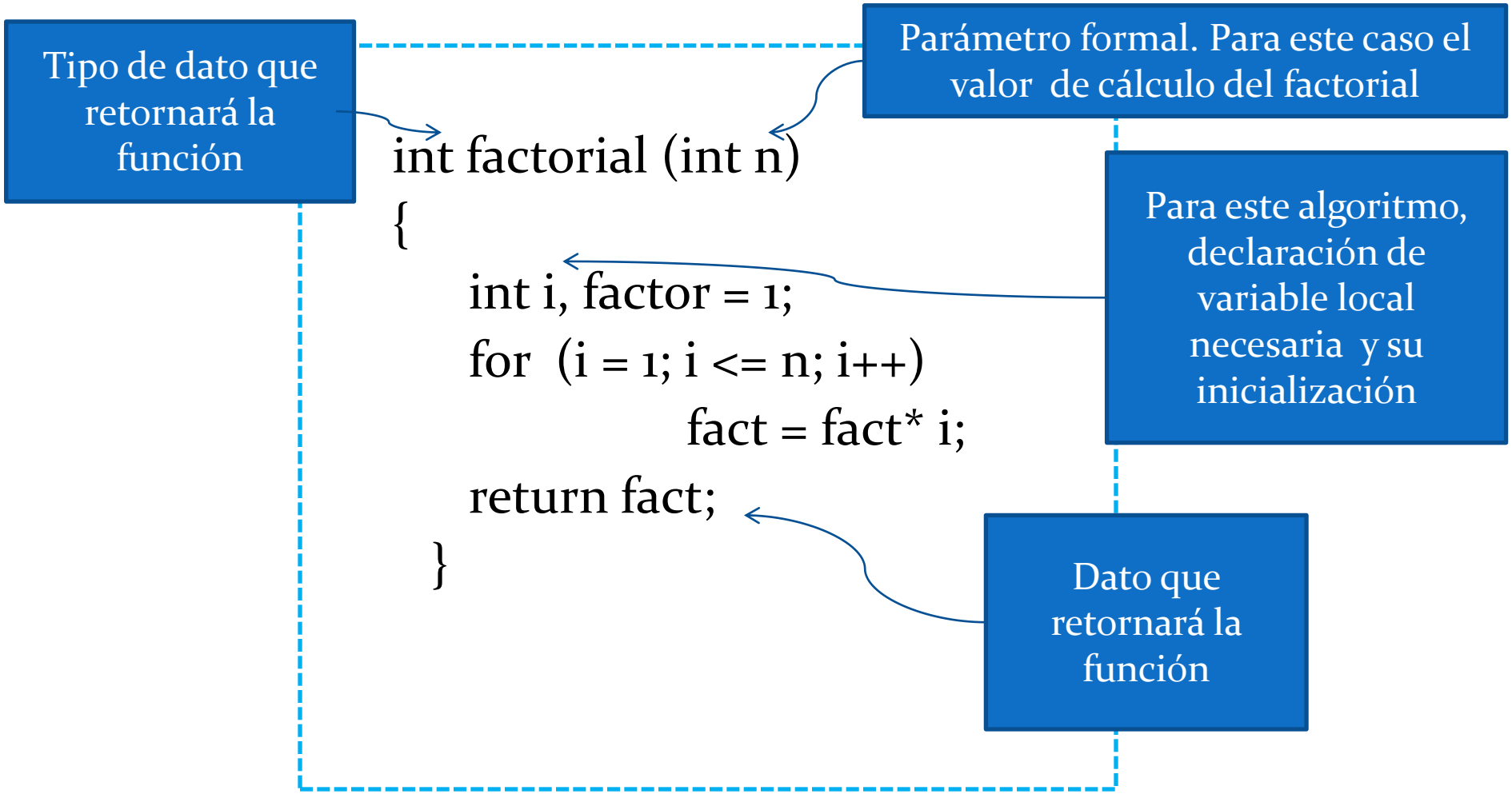
- *Se puede usar en toda situación en la cual* ***la solución pueda ser expresada como una secuencia de movimientos, pasos o transformaciones gobernadas por un conjunto de reglas no ambiguas.***

Las funciones recursivas se componen de:	
Caso base:	**Caso recursivo:**
Una solución simple para un caso particular (puede haber más de un caso base).	*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:*
▪ La secuenciación, iteración condicional y selección son estructuras válidas de control que pueden ser consideradas como enunciados. ▪ Las estructuras de control que se pueden formar combinando de manera válida, la secuenciación, iteración condicional y selección también son válidos.	▪ La función se llama a sí misma. ▪ El problema se resuelve, resolviendo el mismo problema pero de tamaño menor. ▪ La manera en la cual el tamaño del problema disminuye asegura que el caso base eventualmente se alcanzará.

Ejemplo: Analicemos el cálculo del factorial de un número…

- 0! = 1
- 1! = 1
- 2! = 2 → 2! = 2 * 1!
- 3! = 6 → 3! = 3 * 2!
- 4! = 24 → 4! = 4 * 3!
- 5! = 120 → 5! = 5 * 4!

Ejemplo: Desarrollo de la función factorial de manera iterativa:



Analizando la secuencia de factoriales:

- El factorial de 0 = 0! → 1
- El factorial de 1 = 1! → 1 * 0! = 1
- El factorial de 2 = 2! → 2 * 1! = 2
- El factorial de 3 = 3! → 3 * 2! = 6
- El factorial de 4 = 4! → 4 * 3! = 24
- El factorial de 5 = 5! → 5 * 4! = 120

Concluimos en que:

$$n! = n * (n - 1)!$$

Ejemplo: Desarrollo de la función factorial en forma recursiva

```
int factorial (int n) {

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

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.***

Llamada recursiva

Representando el factorial de 3…

Espacio de main()
factorial(3)

Traza de algoritmos recursivos:

Se representan en cascada cada una de las llamadas al módulo recursivo, y los valores que devuelven.

Espacio de main()
factorial(3)

Espacio de factorial(3)

```
if(3==0)
        return 1;
else
        return factorial(3-1)*3;
```

Espacio de main()
factorial(3)

Espacio de factorial(3)

```
if(3==0)
        return 1;
else
        return factorial(3-1)*3;
```

Espacio de factorial(2)

```
if(2==0)
        return 1;
else
        return factorial(2-1)*2;
```

Espacio de main()
factorial(3)

Espacio de factorial(3)

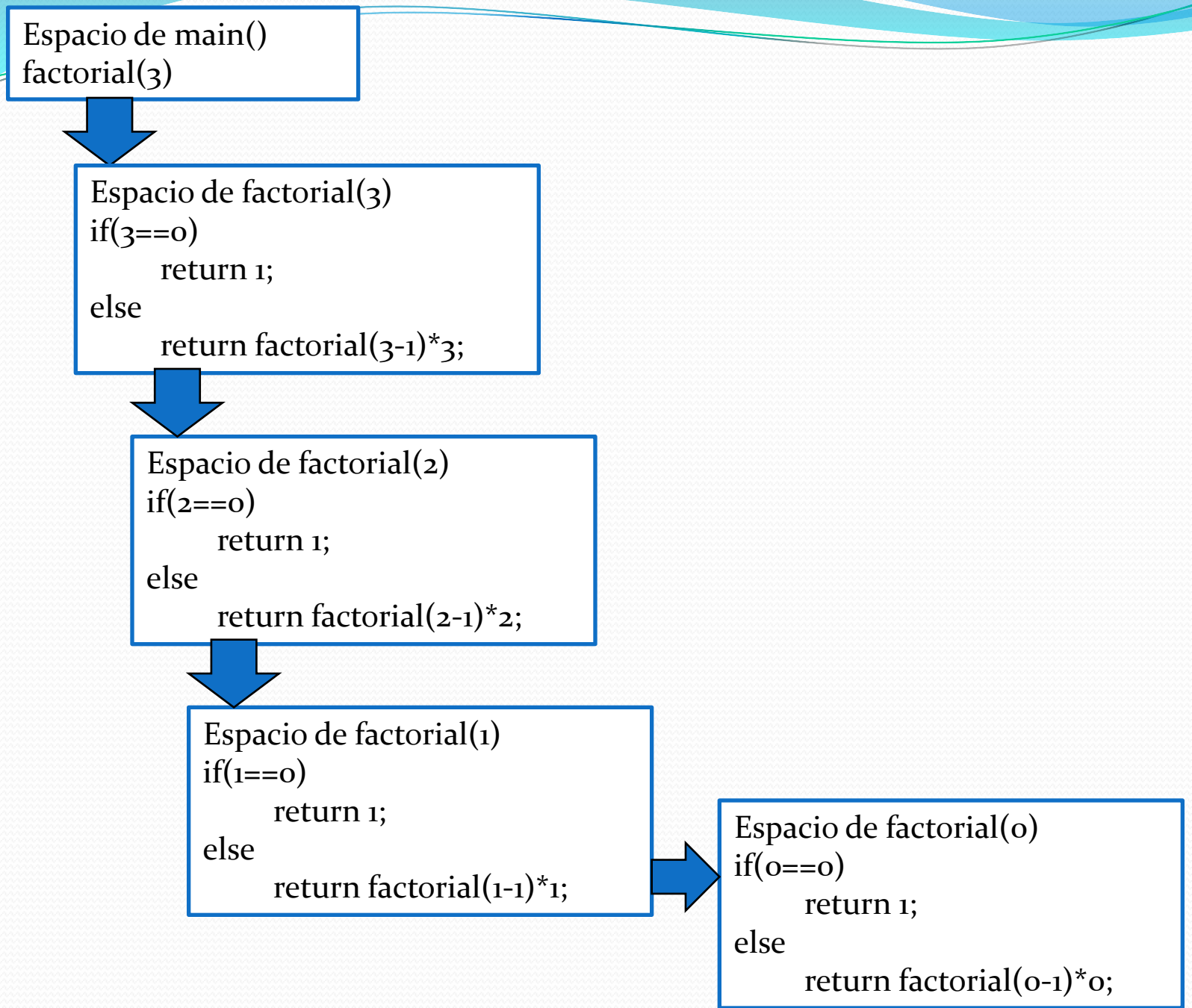
```
if(3==0)
        return 1;
else
        return factorial(3-1)*3;
```

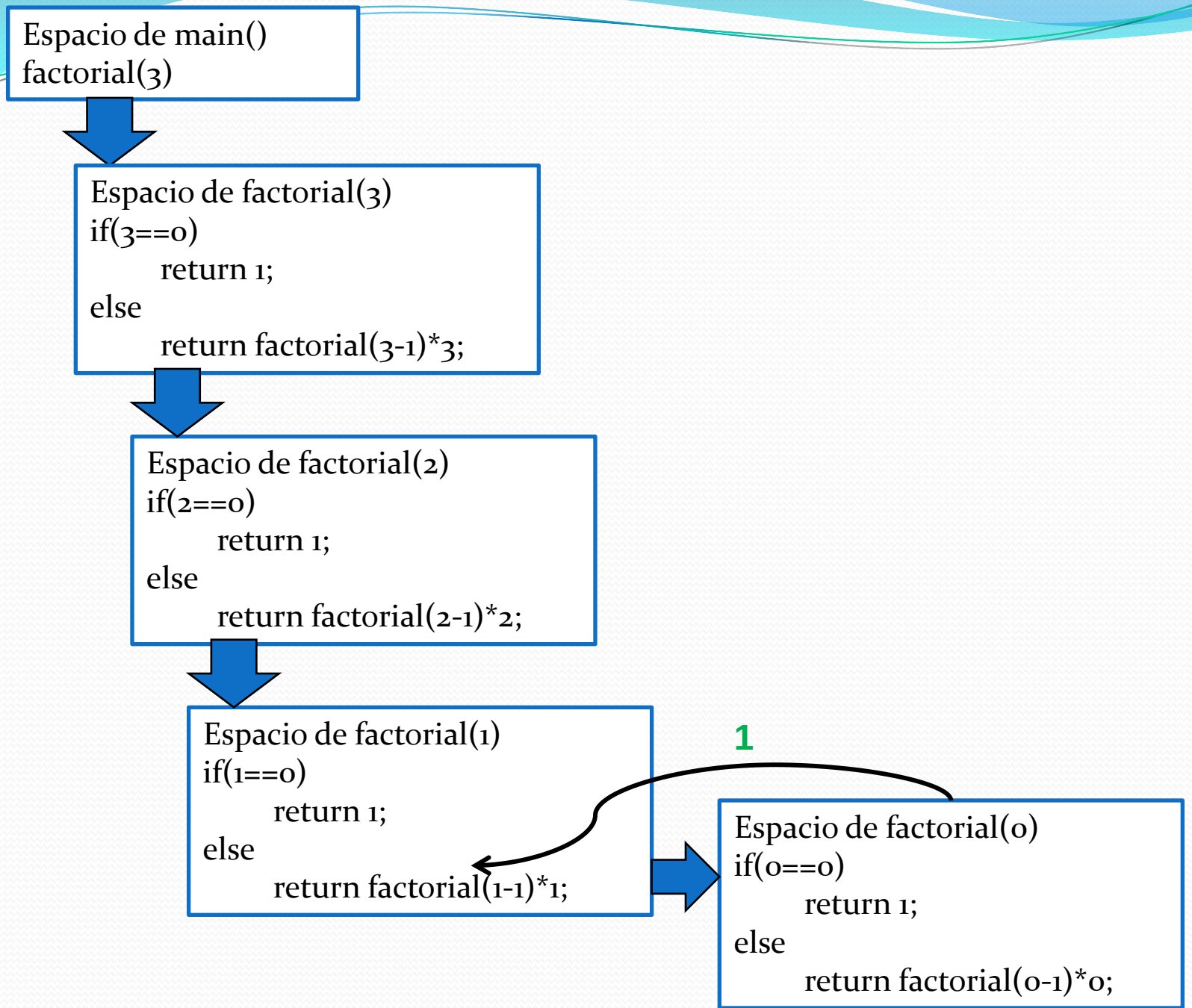
Espacio de factorial(2)

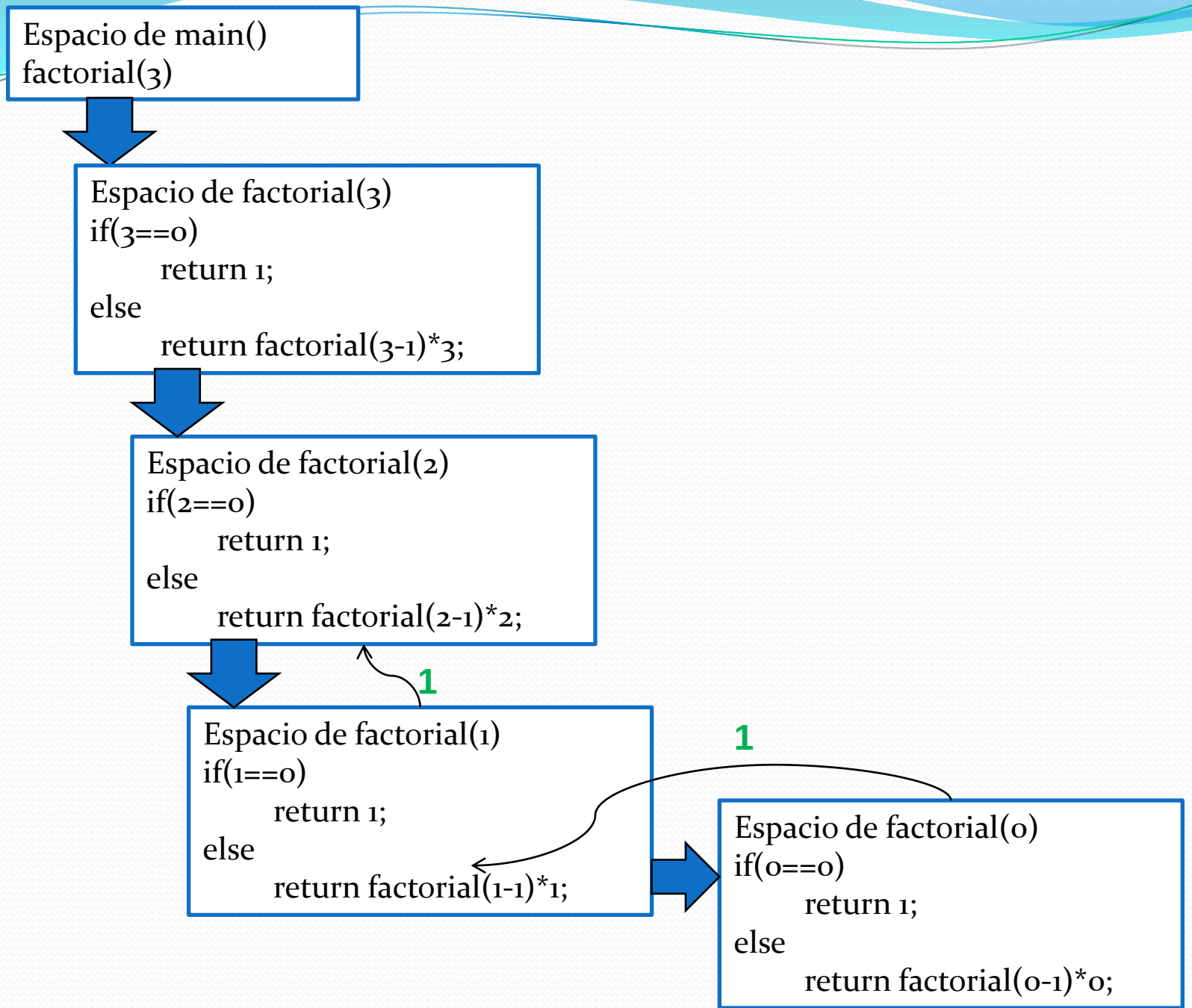
```
if(2==0)
        return 1;
else
        return factorial(2-1)*2;
```

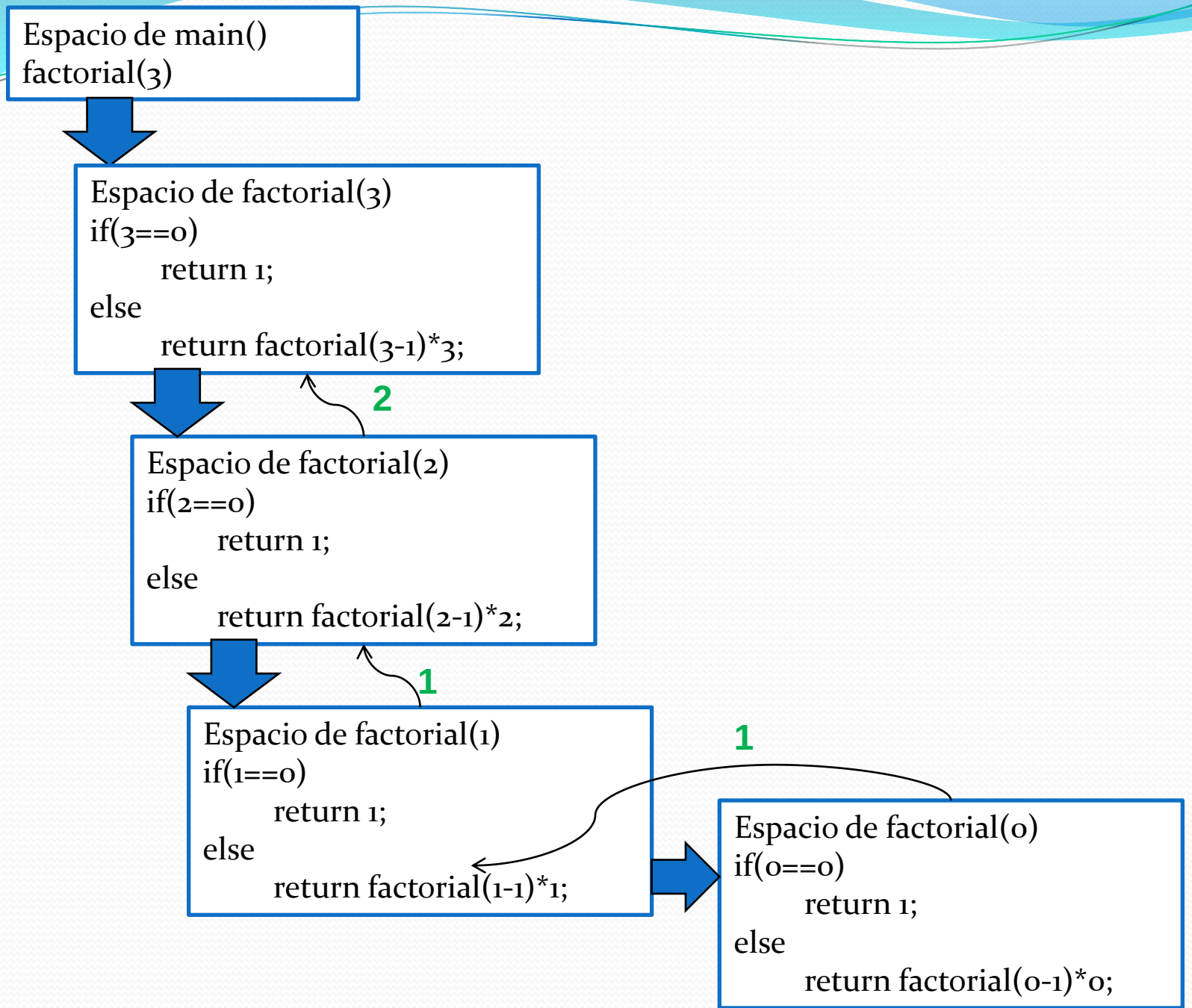
Espacio de factorial(1)

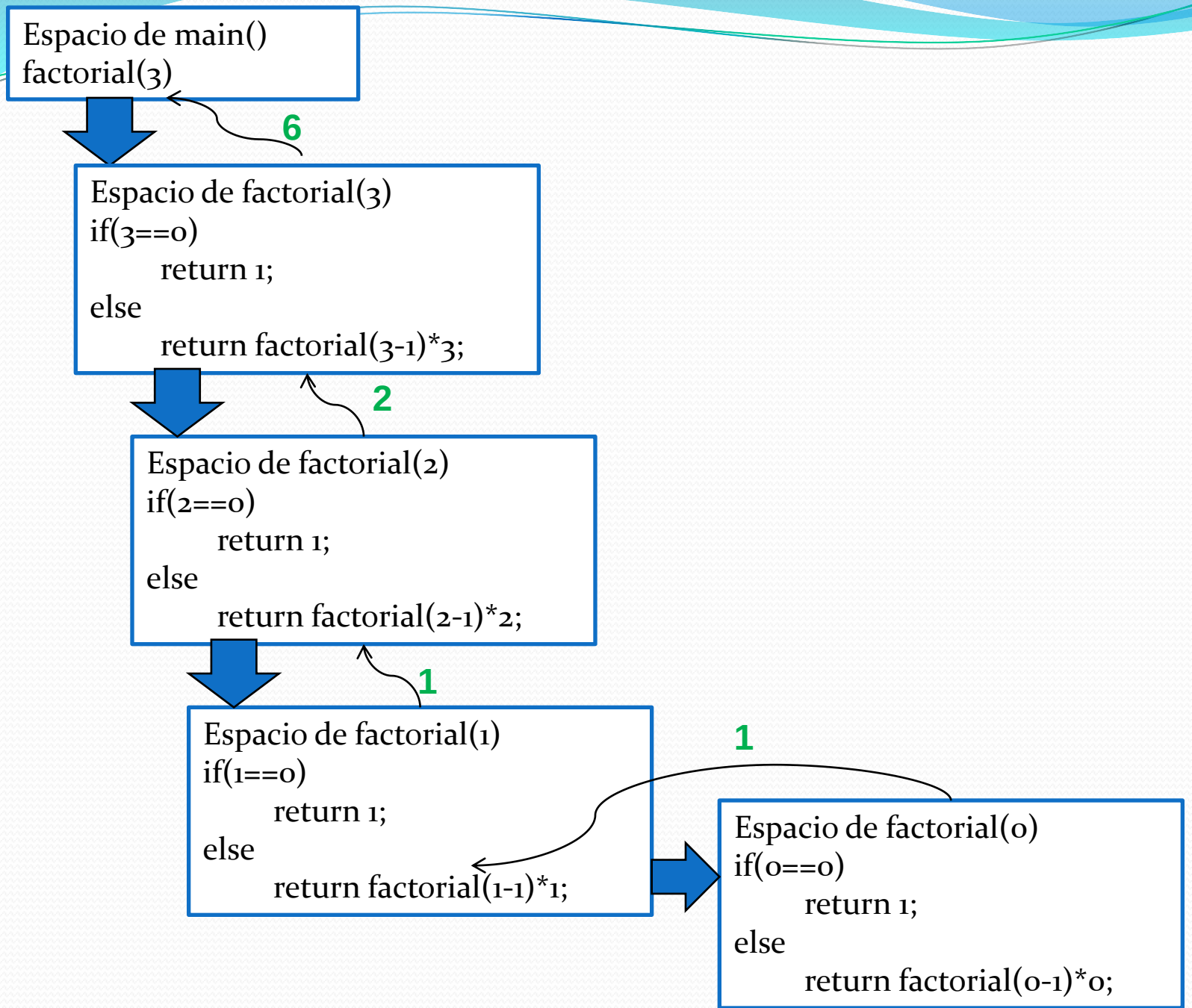
```
if(1==0)
        return 1;
else
        return factorial(1-1)*1;
```


Espacio de main()
factorial(3)
Espacio de factorial(3)
if(3==0)
return 1;
else
return factorial(3-1)*3;
Espacio de factorial(2)
if(2==0)
return 1;
else
return factorial(2-1)*2;
Espacio de factorial(1)
if(1==0)
return 1;
else
return factorial(1-1)*1;
Espacio de factorial(0)
if(0==0)
return 1;
else
return factorial(0-1)*0;


Espacio de main()
factorial(3)
Espacio de factorial(3)
if(3==0)
return 1;
else
return factorial(3-1)*3;
Espacio de factorial(2)
if(2==0)
return 1;
else
return factorial(2-1)*2;
Espacio de factorial(1)
if(1==0)
return 1;
else
return factorial(1-1)*1;
1
Espacio de factorial(0)
if(0==0)
return 1;
else
return factorial(0-1)*0;


Espacio de main()
factorial(3)
Espacio de factorial(3)
if(3==0)
return 1;
else
return factorial(3-1)*3;
Espacio de factorial(2)
if(2==0)
return 1;
else
return factorial(2-1)*2;
1
Espacio de factorial(1)
if(1==0)
return 1;
else
return factorial(1-1)*1;
1
Espacio de factorial(0)
if(0==0)
return 1;
else
return factorial(0-1)*0;


Espacio de main()
factorial(3)
Espacio de factorial(3)
if(3==0)
return 1;
else
return factorial(3-1)*3;
2
Espacio de factorial(2)
if(2==0)
return 1;
else
return factorial(2-1)*2;
1
Espacio de factorial(1)
if(1==0)
return 1;
else
return factorial(1-1)*1;
1
Espacio de factorial(0)
if(0==0)
return 1;
else
return factorial(0-1)*0;


Espacio de main()
factorial(3)
6
Espacio de factorial(3)
if(3==0)
return 1;
else
return factorial(3-1)*3;
2
Espacio de factorial(2)
if(2==0)
return 1;
else
return factorial(2-1)*2;
1
Espacio de factorial(1)
if(1==0)
return 1;
else
return factorial(1-1)*1;
1
Espacio de factorial(0)
if(0==0)
return 1;
else
return factorial(0-1)*0;

Solución

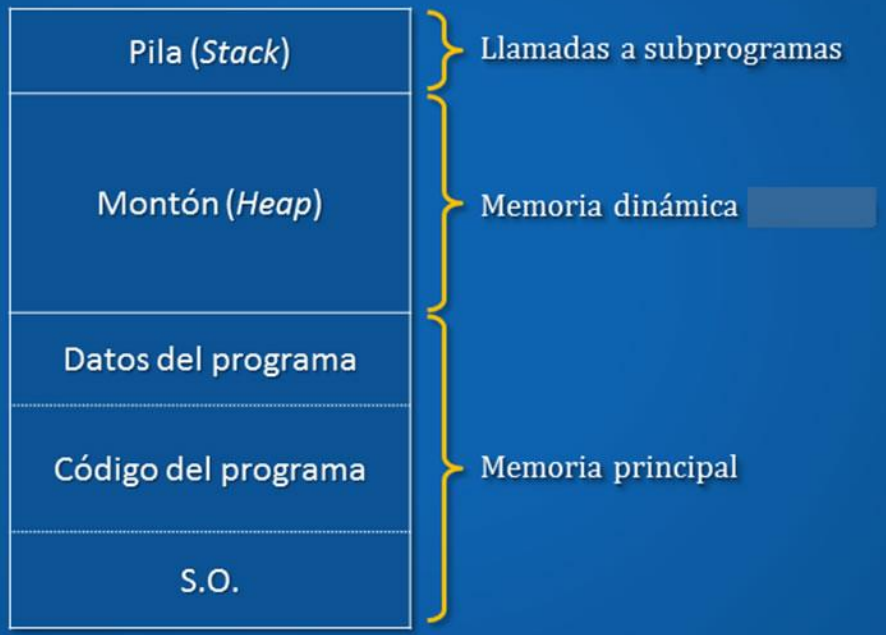
Aquí podemos ver la secuencia que toma el factorial:

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

*Un razonamiento recursivo tiene dos partes: **la base y la regla recursiva de construcción.** La base no es recursiva y es el punto tanto de partida como de terminación de la definición.*

La pila del sistema (*stack*)

Regiones de memoria que distingue el sistema operativo:

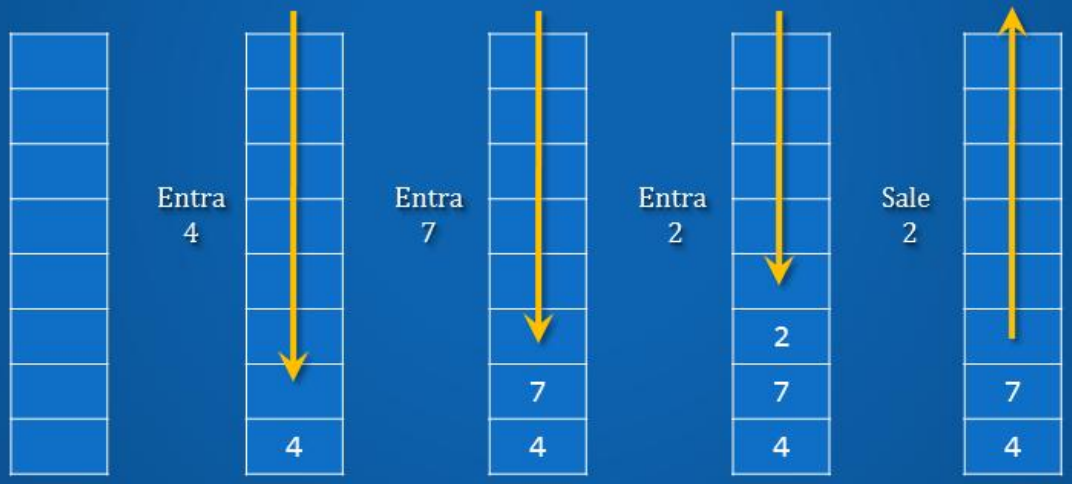


La pila del sistema (*stack*)

Mantiene los datos locales de la función y la dirección de vuelta

Estructura de tipo *pila*: lista LIFO (*last-in first-out*)

El último que entra es el primero que sale:



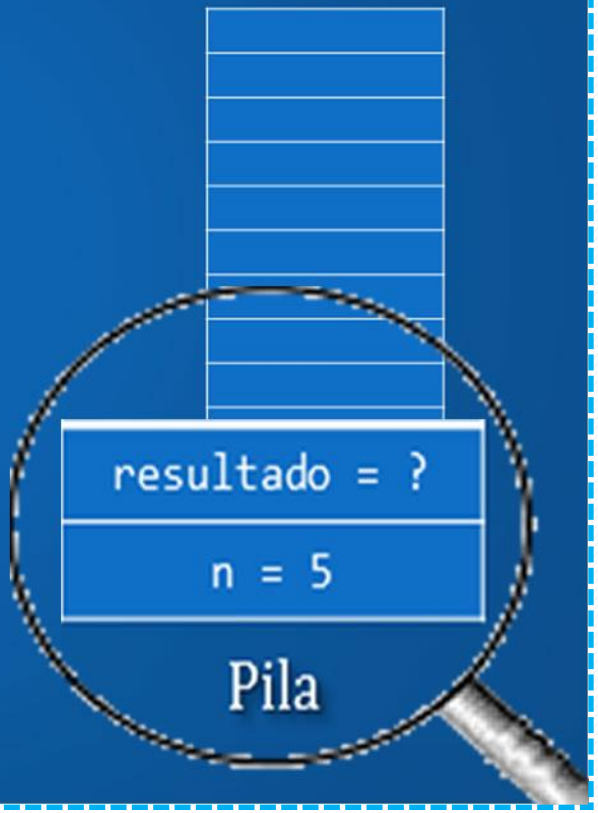
Ejecución de la función factorial()

```
factorial(5)
```

Pila

Ejecución de la función factorial()

```
factorial(5)
```



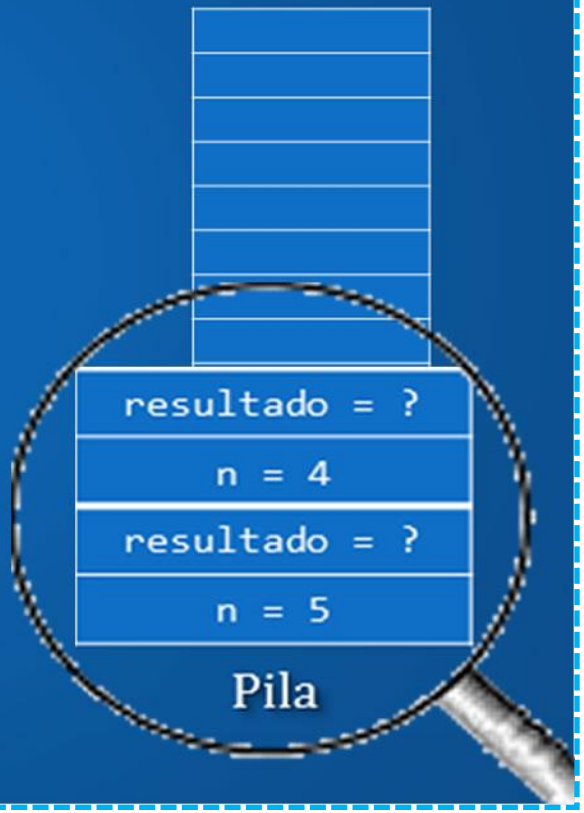
Ejecución de la función factorial()

```
factorial(5)
  └→ factorial(4)
```

Pila

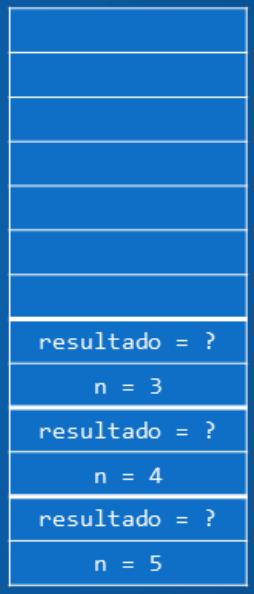
Ejecución de la función factorial()

```
factorial(5)
  └→ factorial(4)
```



Ejecución de la función factorial()

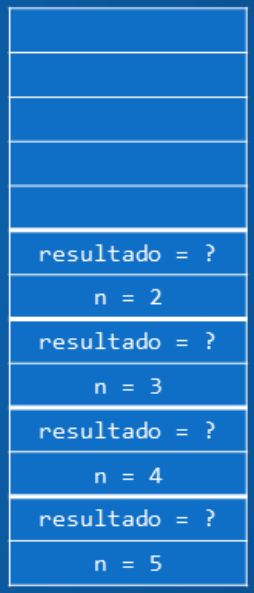
```
factorial(5)
  └→ factorial(4)
       └→ factorial(3)
```



Pila

Ejecución de la función factorial()

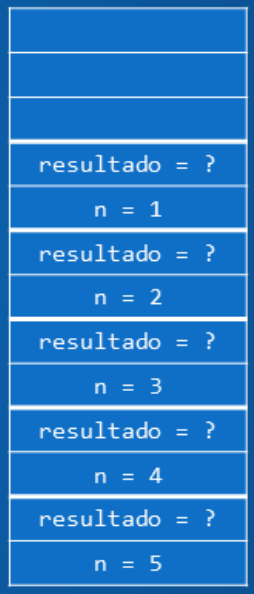
```
factorial(5)
  └→ factorial(4)
       └→ factorial(3)
            └→ factorial(2)
```



Pila

Ejecución de la función factorial()

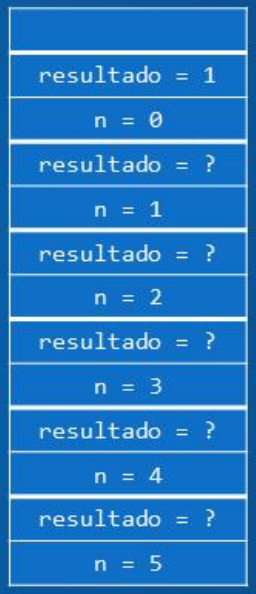
```
factorial(5)
  └→ factorial(4)
       └→ factorial(3)
            └→ factorial(2)
                 └→ factorial(1)
```



Pila

Ejecución de la función factorial()

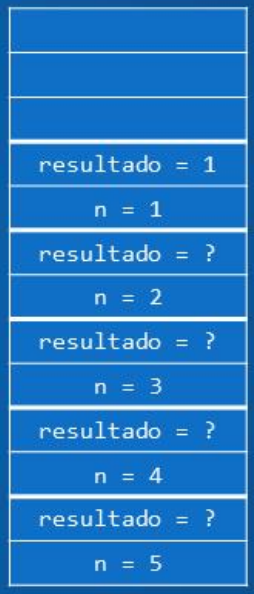
```
factorial(5)
  └→ factorial(4)
       └→ factorial(3)
            └→ factorial(2)
                 └→ factorial(1)
                      └→ factorial(0)
```



Pila

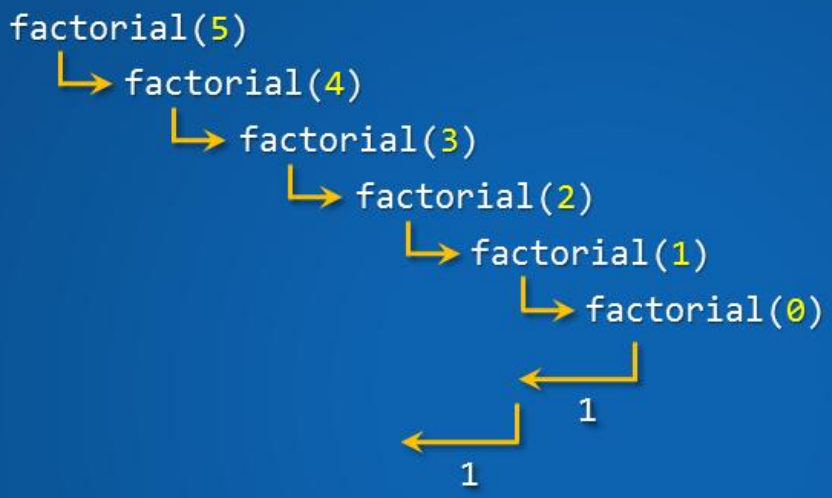
Ejecución de la función factorial()

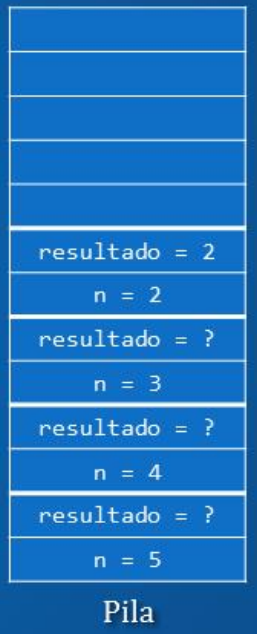
```
factorial(5)
  └→ factorial(4)
       └→ factorial(3)
            └→ factorial(2)
                 └→ factorial(1)
                      └→ factorial(0)
                      ←┘
                      1
```



Pila

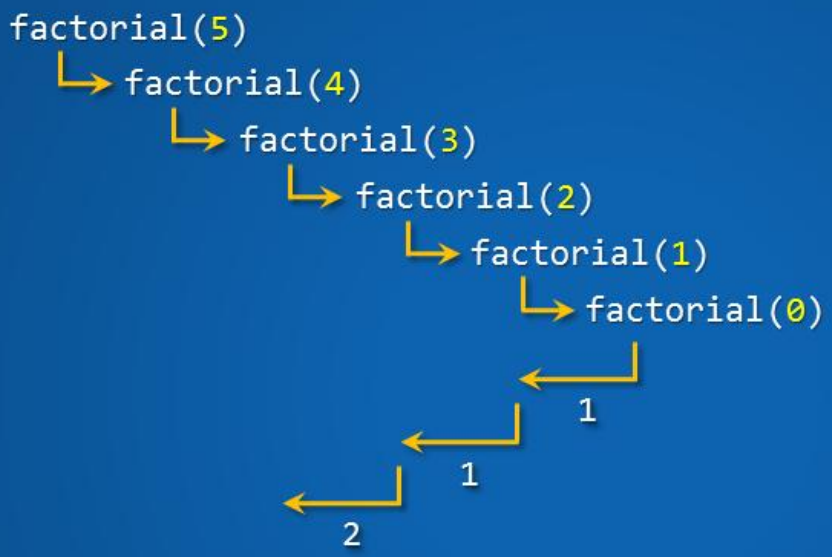
Ejecución de la función factorial()

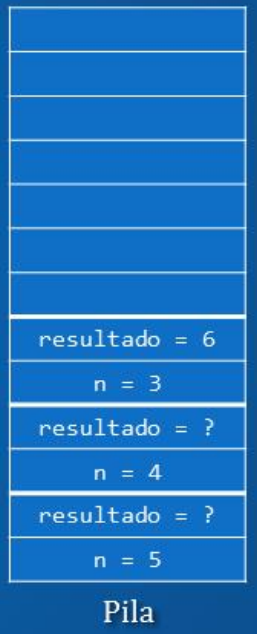




Pila

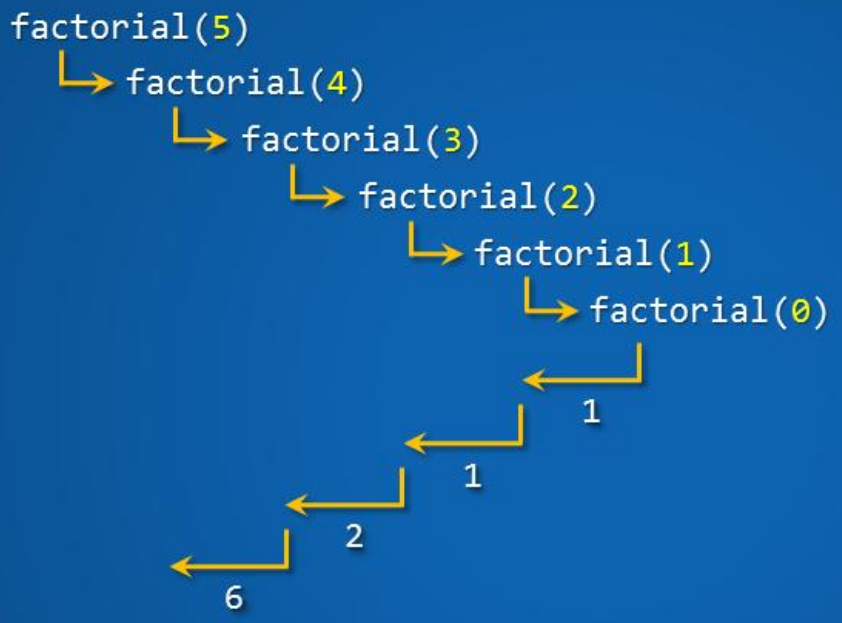
Ejecución de la función factorial()





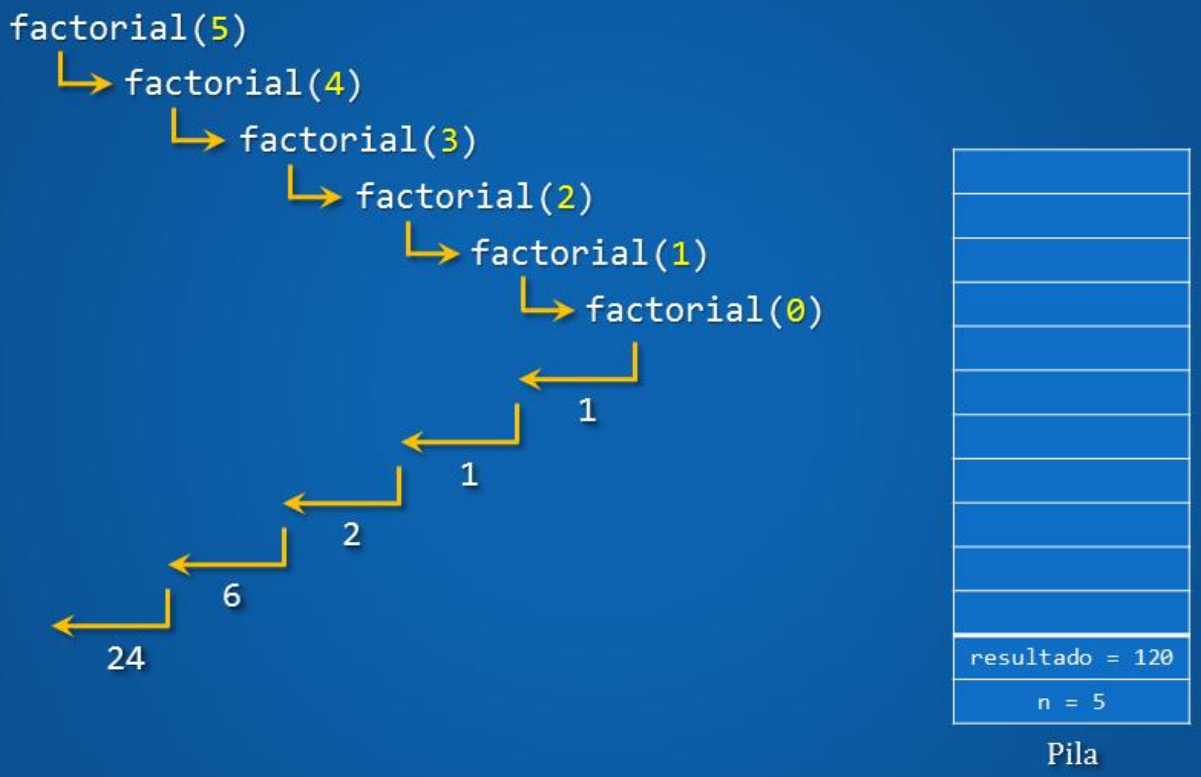
Pila

Ejecución de la función factorial()

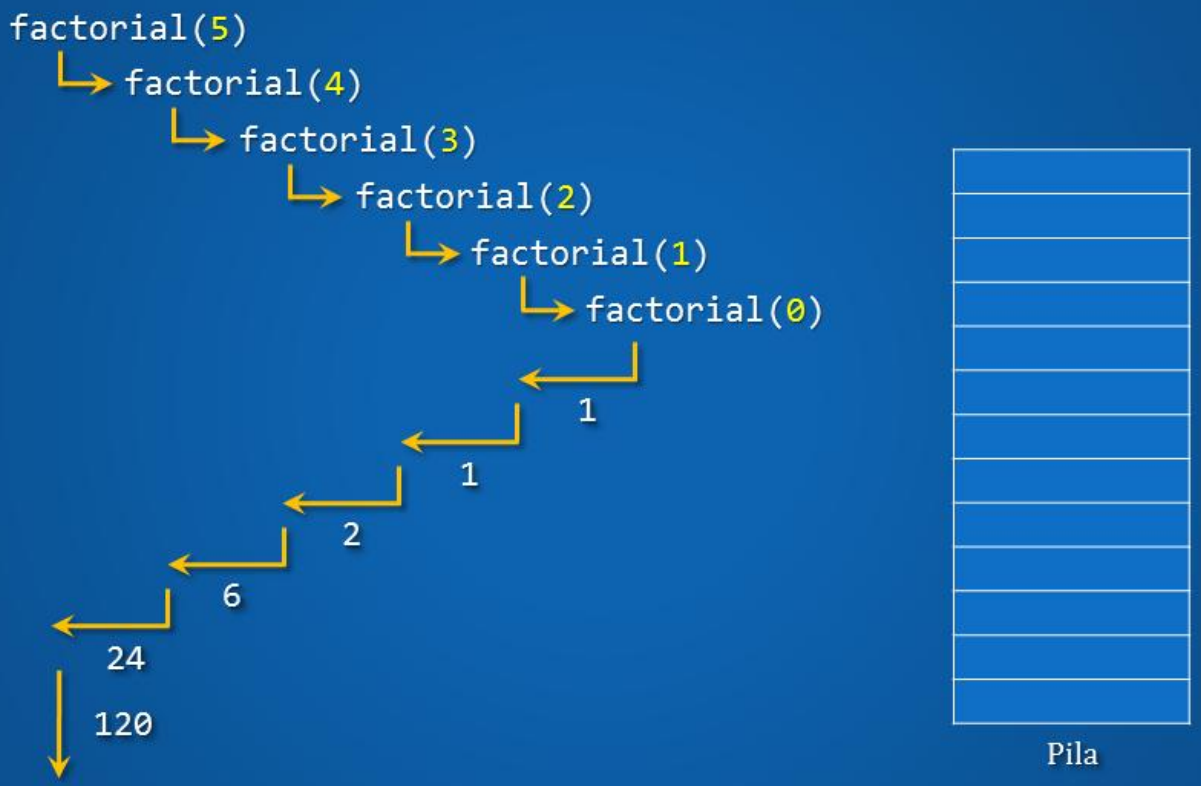


Pila

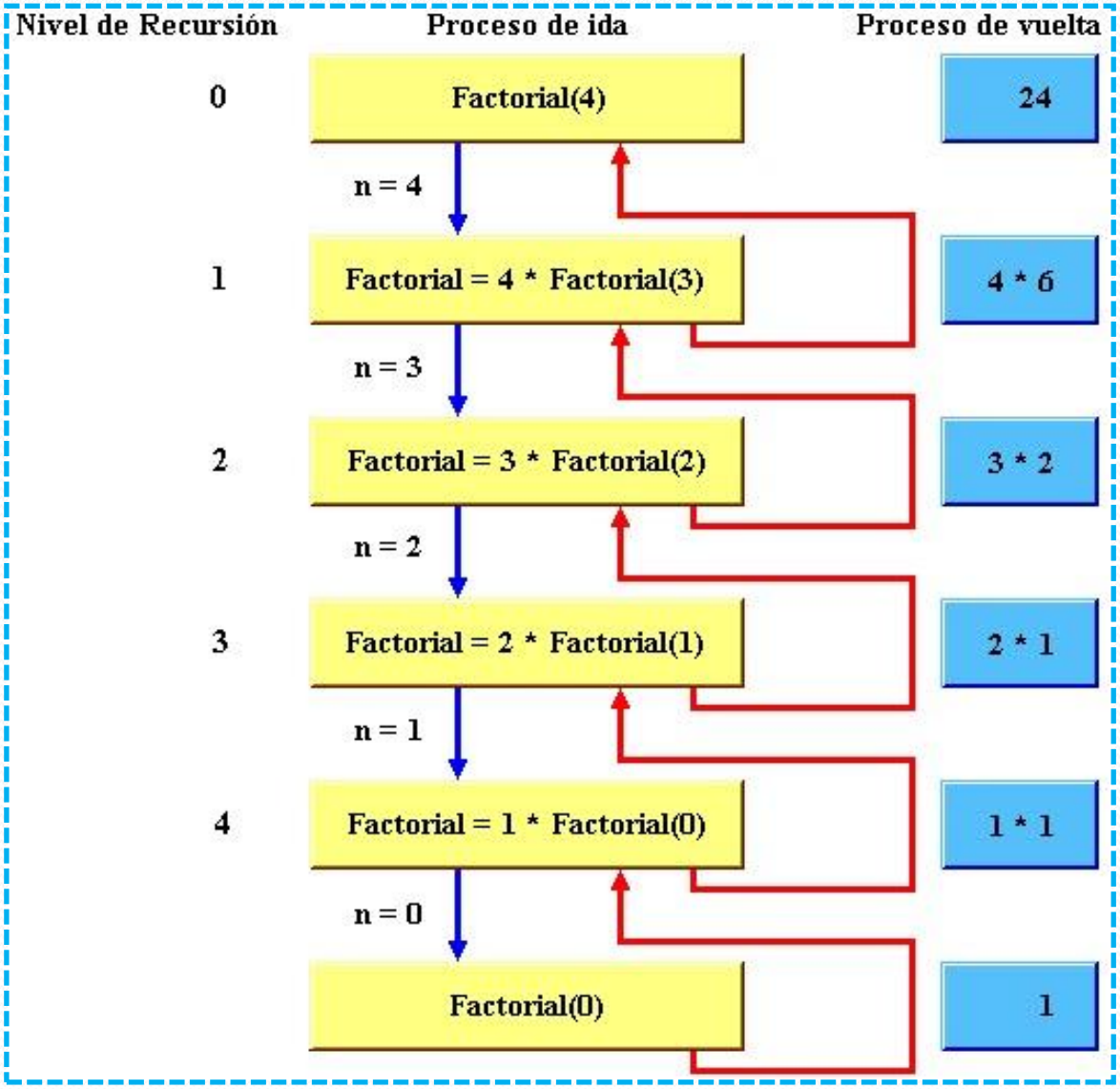
Ejecución de la función factorial()



Ejecución de la función factorial()



Resumiendo...


Nivel de Recursión
Proceso de ida
Proceso de vuelta
0
Factorial(4)
24
n = 4
1
Factorial = 4 * Factorial(3)
4 * 6
n = 3
2
Factorial = 3 * Factorial(2)
3 * 2
n = 2
3
Factorial = 2 * Factorial(1)
2 * 1
n = 1
4
Factorial = 1 * Factorial(0)
1 * 1
n = 0
Factorial(0)
1

¿Cómo escribir una función en forma recursiva?
Sintaxis:

```
<tipo_de_dato _de_retorno> <nom_fnc> (<param>) {

    [declaración de variables]
    [condición de salida] (caso base)
      [instrucciones]
    [llamada a <nom_fnc> (<param>)] (caso recursivo)
      return <resultado>
}
```

*Cuando una función recursiva se llama recursivamente a sí misma, para cada llamada se crean **copias independientes** de las variables declaradas en el procedimiento.*

¿Por qué escribir programas recursivos?...

- *Son más cercanos a la descripción matemática.*
- *Generalmente más fáciles de analizar.*
- *Se adaptan mejor a las estructuras de datos recursivas.*
- *Los algoritmos recursivos ofrecen soluciones estructuradas, modulares y elegantemente simples.*

¿Cuándo usar recursividad?	¿Cuándo NO usar recursividad?
Para simplificar el código.	*Cuando los métodos usen arrays largos.*
Cuando la estructura de datos es recursiva ejemplo: listas, árboles.	*Cuando el método cambia de manera impredecible de campos.*
	Cuando las iteraciones sean la mejor opción.

Clasificaciones	
recursión directa	***recursión indirecta***
Cuando una función incluye una llamada a sí misma se conoce como recursión directa. (ejemplo del factorial)	*Cuando una función llama a otra función y esta causa que la función original sea invocada, se conoce como recursión indirecta.*

Código de ejemplo de recursión indirecta

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

int par(int n);
int impar(int n);

int main() {
   int x;
   printf( "Introduzca un entero:\n " );
   scanf( "%d", &x );
   if (par(x)==1) printf( "\n %d Es par\n", x);
   else printf( "\n %d Es impar\n", x);

system("Pause");
return 0;
}
```

```
int par(int n) {
  if (n == 0) return 1;
  return impar(n-1);
}

int impar(int n) {
  if (n == 0) return 0;
  return par(n-1);
}
```

Recursión vs. Iteración	Iteración:	Recursión:
Repetición:	el ciclo es explícito	hay repetidas invocaciones a la función
Terminación:	cuando la condición del ciclo es falsa	llega al caso base

Fin