Recursividad Funciones Recursivas



¿Qué es la recursividad?

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

Las funciones recursivas se componen de:

Caso base:

Una solución simple para un caso particular (puede haber más de un caso base).

- 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.

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:

- 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...

$$\Box$$
 $o! = 1$

$$\square$$
 1! = 1

$$\square$$
 2! = 2

$$3! = 6$$

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

Ejemplo: Desarrollo de la función factorial de manera iterativa:

Tipo de dato que retornará la función Parámetro formal. Para este caso el valor de cálculo del factorial

int factorial (int n)

Para este algoritmo, declaración de variable local necesaria y su inicialización

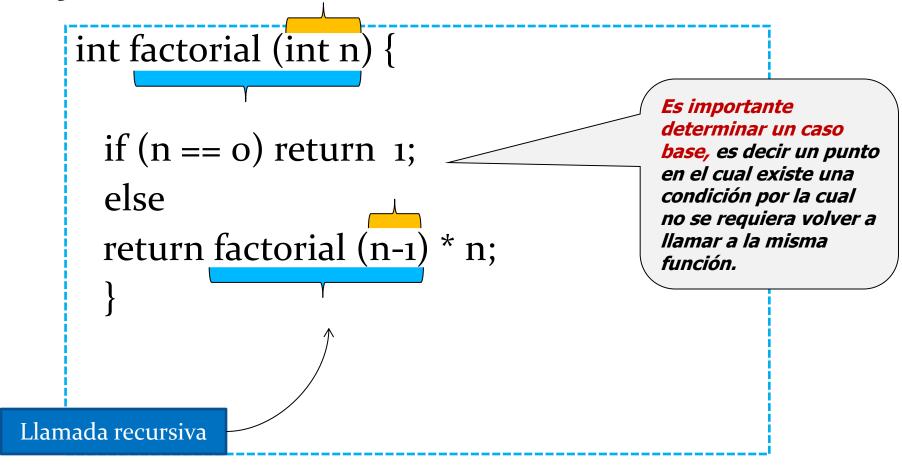
Dato que retornará la función

Analizando la secuencia de factoriales:

- El factorial de $o = o! \rightarrow 1$
- El factorial de $1 = 1! \rightarrow 1 * 0! = 1$
- El factorial de $2 = 2! \rightarrow 2 * 1! = 2$
- El factorial de $3 = 3! \rightarrow 3 * 2! = 6$
- El factorial de $4 = 4! \rightarrow 4 * 3! = 24$
- El factorial de $5 = 5! \rightarrow 5 * 4! = 120$ Concluimos en que:

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

Ejemplo: Desarrollo de la función factorial en forma 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;
                                                       Espacio de factorial(o)
              else
                                                       if(o==o)
                   return factorial(1-1)*1;
                                                             return 1;
                                                       else
                                                             return factorial(o-1)*o;
```

```
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;
                                                       Espacio de factorial(o)
              else
                                                       if(o==o)
                   return factorial(1-1)*1;
                                                             return 1;
                                                       else
                                                             return factorial(o-1)*o;
```

```
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;
                                                       Espacio de factorial(o)
              else
                                                       if(o==o)
                   return factorial(1-1)*1;
                                                             return 1;
                                                       else
                                                             return factorial(o-1)*o;
```

```
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;
                                                       Espacio de factorial(o)
              else
                                                       if(o==o)
                   return factorial(1-1)*1;
                                                             return 1;
                                                       else
                                                             return factorial(o-1)*o;
```

```
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;
                                                       Espacio de factorial(o)
              else
                                                       if(o==o)
                   return factorial(1-1)*1;
                                                             return 1;
                                                       else
                                                             return factorial(o-1)*o;
```

Solución

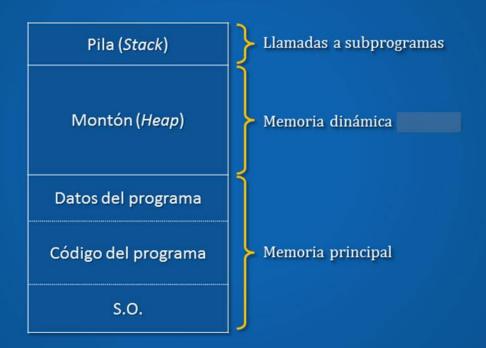
Aquí podemos ver la secuencia que toma el factorial:

$$N ! = \begin{cases} 1 & \text{si N = o (caso base)} \\ N ! & \text{si N > o (recursion)} \end{cases}$$

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

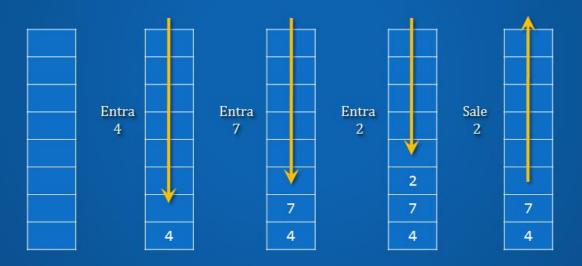
La pila del sistema (stack)

Regiones de memoria que distingue el sistema operativo:

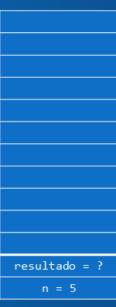


La pila del sistema (stack)

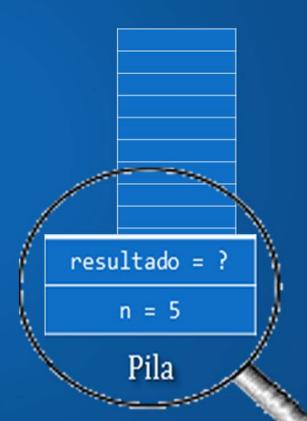
Mantiene los datos locales de la función y la dirección de vuelta Estructura de tipo *pila*: lista LIFO (*last-in first-out*) El último que entra es el primero que sale:



factorial(5)



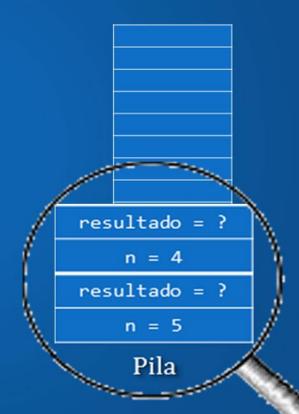
factorial(5)



resultado = ?

n = 4

resultado = ?



resultado = ?

resultado = ?

n = 4

resultado = ?

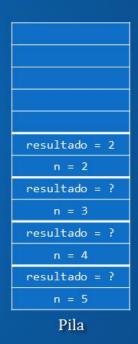
resultado = ?
 n = 2
 resultado = ?
 n = 3
 resultado = ?
 n = 4
 resultado = ?
 n = 5

resultado = ?
 n = 1
resultado = ?
 n = 2
resultado = ?
 n = 3
resultado = ?
 n = 4
resultado = ?
 n = 5

resultado = 1		
n = 0		
resultado = ?		
n = 1		
resultado = ?		
n = 2		
resultado = ?		
n = 3		
resultado = ?		
n = 4		
resultado = ?		
n = 5		
Dila		

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

resultado = 1		
n = 1		
resultado = ?		
n = 2		
resultado = ?		
n = 3		
resultado = ?		
n = 4		
resultado = ?		
n = 5		
Pila		



```
factorial(5)

factorial(4)

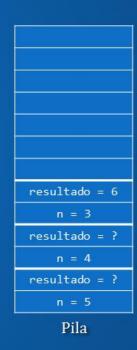
factorial(3)

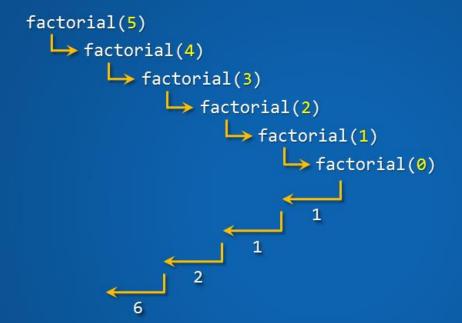
factorial(2)

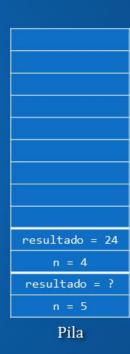
factorial(1)

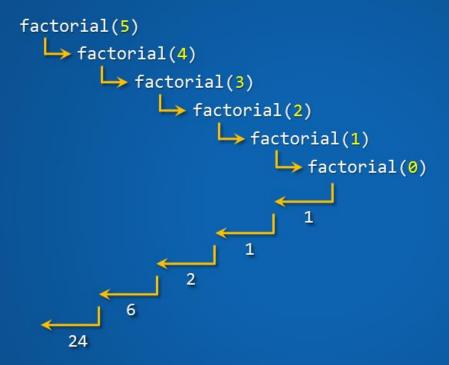
factorial(0)

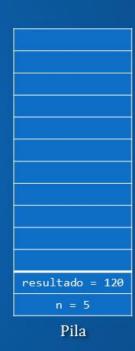
1
```

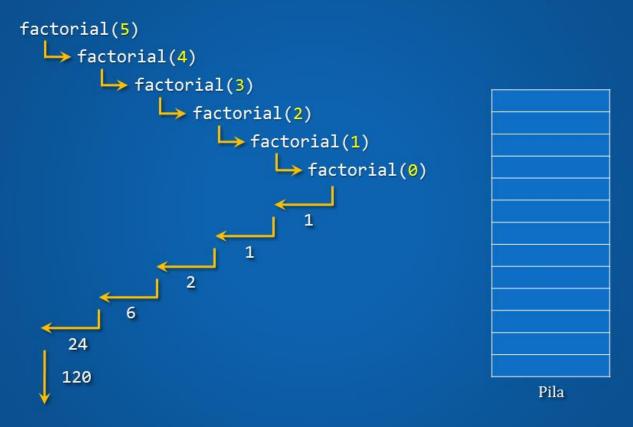




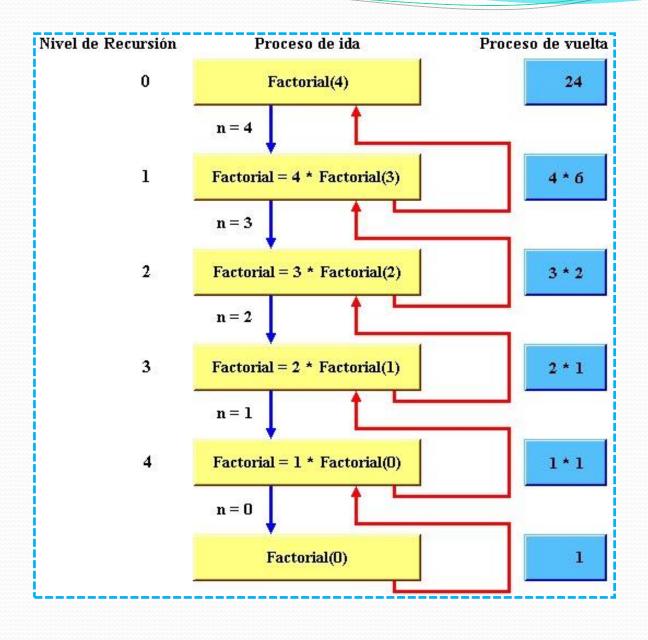








Resumiendo...



¿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 <u>copias independientes</u> de las variables declaradas en el procedimiento.

Construye un programa que tenga un menú de opciones que resuelvan los siguientes ejercicios...

Ejercicio I: escribe una función recursiva que calcule la potencia de un número.

```
Cálculo de la potencia
            x^n = \begin{cases} 1 & \text{si } n = 0 \\ x \cdot x^{n-1} & \text{si } n > 0 \end{cases}
int potencia(int base, int expo){
   if (expo==0)
      return 1;
   else
      return base * potencia(base, expo-1);
```

Ejercicio 2: escribe una función recursiva que sume dos valores.

```
La suma de forma recursiva
suma(a,b) = \begin{cases} a & \text{si } b = 0\\ 1 + suma(a,b-1) & \text{si } b > 0 \end{cases}
int suma(int a, int b){
   if (b==0)
      return a;
  else
     return 1+suma(a,b-1);
```

Ejercicio 3: escribe una función recursiva que calcule el producto de dos números.

3. El producto de forma recursiva $producto(a,b) = \left\{ \begin{array}{ll} 0 & \text{si } b=0 \\ a+producto(a,b-1) & \text{si } b>0 \end{array} \right.$ int producto(int a, int b) { if (b==0) return 0; else return a+producto(a,b-1);

Ejercicio 4: Construye un programa para resolver la serie de Fibonacci. Cada término de la serie suma los dos anteriores. Ejemplo: 0, 1, 1, 2, 3, 5, 8...

Fórmula	recursiva

$$fib(n) = fib(n - 1) + fib(n - 2)$$

Caso base:

Fib (0)=0; Fib (1)=1;

Caso recursivo:

$$Fib(i) = Fib(i-1) + Fib(i-2);$$

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

Clasificaciones

recursión directa

Cuando una función incluye una llamada a sí misma se conoce como <u>recursión directa.</u>
(ejemplo del factorial)

recursión indirecta

Cuando una función llama a otra función y esta causa que la función original sea invocada, se conoce como <u>recursión</u> 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:	Recursión:
<i>Iteración</i>		
Repetición:	el ciclo es explícito	hay repetidas invocaciones a la función
Terminación:	cuando la condición del ciclo es falsa	llega al caso base
En ambos casos podemos tener ciclos infinitos		

