

# Recursividad

Las funciones recursivas son aquellas que se resuelven combinando resultados de invocaciones a sí misma con subproblemas más pequeños.

*Ejemplo: El factorial tiene dos definiciones. Para  $n \in \mathbb{N}$*

Iterativa:  $n! = \prod_{i=1}^n i$

Recursiva:  $n! = \begin{cases} 1 & \text{Si } n = 0 \\ n * (n - 1)! & \text{Si } n > 0 \end{cases}$

*Caso trivial* → Si  $n = 0$

*Caso NO trivial* → Si  $n > 0$

Iterativa:  $n! = \prod_{i=1}^n i$

```
1  #include<iostream>
2
3▼ int factorial(int n) {
4    int i, resultado;
5    resultado = 1;
6    for (i = n; i > 1; i--)
7        resultado *= i;
8    return(resultado);
9▲ }
10
11▼ int main(void) {
12    int resultado;
13    resultado = factorial(3);
14    std::cout << resultado << std::endl;
15▲ }
```

Recursiva:  $n! = \begin{cases} 1 & \text{Si } n = 0 \\ n * (n - 1)! & \text{Si } n > 0 \end{cases}$

```
1  #include<iostream>
2
3▼ int factorial(int n) {
4    if (n == 0) return(1);
5    else return(n * factorial(n-1));
6▲ }
7
8▼ int main(void) {
9    int resultado;
10    resultado = factorial(3);
11    std::cout << resultado << std::endl;
12▲ }
```

En C, C++ o en los lenguajes de alto nivel  
La recursividad funciona muy bien y ayuda a  
Resolver problemas complejos.

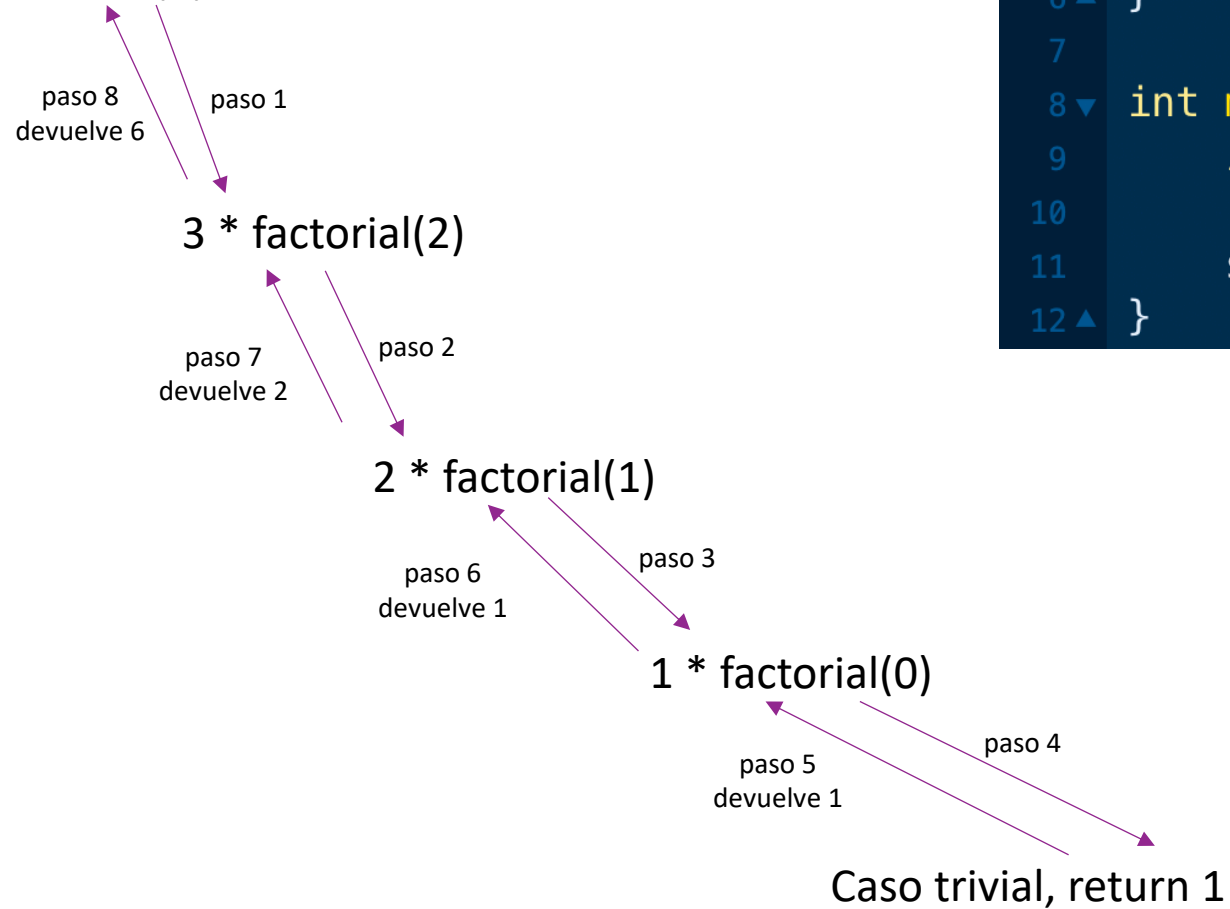
Iterativa:  $n! = \prod_{i=1}^n i$

```
1  #include<iostream>
2
3  int factorial(int n) {
4      int i, resultado;
5      resultado = 1;
6      for (i = n; i > 1; i--)
7          resultado *= i;
8      return(resultado);
9  }
10
11 int main(void) {
12     int resultado;
13     resultado = factorial(3);
14     std::cout << resultado << std::endl;
15 }
```

```
1      .text
2      # Funcion factorial
3      # Input: $a0 numero a calcular factorial
4      # Output: $v0 = Factorial de la entrada. ($a0)!
5      factorial:
6          li $v0,1 # en $v0 se devuelve el resultado
7          # $t1 iterador
8          move $t1,$a0
9          loop:    blt $t1,2,finloop
10                 mul $v0,$v0,$t1
11                 addi $t1,-1
12                 j loop
13      finloop:
14          jr $ra # return
15
16      main:
17          li $a0,3
18          jal factorial
19          # el valor se devuelve en $v0
20          # print
21          move $a0,$v0
22          li $v0,1
23          syscall
24
25          li $v0,10
26          syscall
27
```

Recursiva:  $n! = \begin{cases} 1 & \text{Si } n = 0 \\ n * (n - 1)! & \text{Si } n > 0 \end{cases}$

resultado = factorial(3)



```
1  #include<iostream>
2
3  ▼ int factorial(int n) {
4      if (n == 0) return(1);
5      else return(n * factorial(n-1));
6  ▲ }
7
8  ▼ int main(void) {
9      int resultado;
10     resultado = factorial(3);
11     std::cout << resultado << std::endl;
12  ▲ }
```

Recursiva: 
$$n! = \begin{cases} 1 & \text{Si } n = 0 \\ n * (n - 1)! & \text{Si } n > 0 \end{cases}$$

```

1  #include<iostream>
2
3  int factorial(int n) {
4      if (n == 0) return(1);
5      else return(n * factorial(n-1));
6  }
7
8  int main(void) {
9      int resultado;
10     resultado = factorial(3);
11     std::cout << resultado << std::endl;
12 }

```

```

1  .text
2  # Funcion factorial
3  # Input: $a0 numero a calcular factorial
4  # Output: $v0 = Factorial de la entrada. ($a0)!
5  factorial:
6      bne $a0,0,notrivial
7          li $v0,1 # en $v0 se devuelve el resultado
8          jr $ra #return
9  notrivial:
10     move $t1,$a0
11     addi $a0,-1
12     jal factorial
13     # en $v0 esta el resultado de esta ultima llamada
14     mul $v0,$t1,$v0
15     jr $ra # return
16
17  main:
18     li $a0,3
19     jal factorial
20     # el valor se devuelve en $v0
21     # print
22     move $a0,$v0
23     li $v0,1
24     syscall
25
26     li $v0,10
27     syscall

```

Recursiva: 
$$n! = \begin{cases} 1 & \text{Si } n = 0 \\ n * (n - 1)! & \text{Si } n > 0 \end{cases}$$

```

1  #include<iostream>
2
3  int factorial(int n) {
4      if (n == 0) return(1);
5      else return(n * factorial(n-1));
6  }
7
8  int main(void) {
9      int resultado;
10     resultado = factorial(3);
11     std::cout << resultado << std::endl;
12 }

```

Realiza la traza paso a paso en QtSpim y comprobarás que esto no funciona. Fíjate de forma especial en la evolución del registro \$t1 y \$ra

```

1  .text
2  # Funcion factorial
3  # Input: $a0 numero a calcular factorial
4  # Output: $v0 = Factorial de la entrada. ($a0)!
5  factorial:
6      bne $a0,0,notrivial
7          li $v0,1 # en $v0 se devuelve el resultado
8          jr $ra #return
9  notrivial:
10     move $t1,$a0
11     addi $a0,-1
12     jal factorial
13     # en $v0 esta el resultado de esta ultima llamada
14     mul $v0,$t1,$v0
15     jr $ra # return
16
17  main:
18     li $a0,3
19     jal factorial
20     # el valor se devuelve en $v0
21     # print
22     move $a0,$v0
23     li $v0,1
24     syscall
25
26     li $v0,10
27     syscall

```



Recursiva:  $n! = \begin{cases} 1 & \text{Si } n = 0 \\ n * (n - 1)! & \text{Si } n > 0 \end{cases}$

```

1  #include<iostream>
2
3  int factorial(int n) {
4      if (n == 0) return(1);
5      else return(n * factorial(n-1));
6  }
7
8  int main(void) {
9      int resultado;
10     resultado = factorial(3);
11     std::cout << resultado << std::endl;
12 }
```

```

1      .text
2      # Funcion factorial
3      # Input: $a0 numero a calcular factorial
4      # Output: $v0 = Factorial de la entrada. ($a0)!
5      factorial:  bgt $a0,$zero,notrivial
6                  li  $v0,1
7                  jr  $ra
8      notrivial:
9
10     push
11
12     addi  $sp, $sp, -8    # reservo 8 bytes en la pila
13     sw  $ra, 4($sp)    # guardo $ra
14     sw  $a0, 0($sp)    # guardo $a0
15
16     addi $a0,$a0,-1    # fact(n) = n * fact(n-1)
17     jal factorial      # llamo a fact(n-1) devuelve en $v0
18
19     pop
20
21     lw  $a0,0($sp)      # recupero $a0 en el mismo orden
22     lw  $ra,4($sp)      # recupero $ra en el mismo orden
23     addi $sp, $sp, 8    # dejo el puntero de pila como estaba
24
25     mul  $v0,$a0,$v0    # multiplico n*fact(n-1) que es $a0*$v0
26     jr  $ra
27
28     # FIN FUNCION factorial
29
30     main:
31     li  $a0,3
32     jal factorial
33     # el valor se devuelve en $v0
34     # print
35     move $a0,$v0
36     li  $v0,1
37     syscall
38
39     li  $v0,10
40     syscall
```