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

Caso trivial

Caso NO trivial

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

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

```cpp
#include<iostream>

int factorial(int n) {
    int i, resultado;
    resultado = 1;
    for (i = n; i > 1; i--)
        resultado *= i;
    return(resultado);
}

int main(void) {
    int resultado;
    resultado = factorial(3);
    std::cout << resultado << std::endl;
}
```

```cpp
#include<iostream>

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

int main(void) {
    int resultado;
    resultado = factorial(3);
    std::cout << resultado << std::endl;
}
```

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

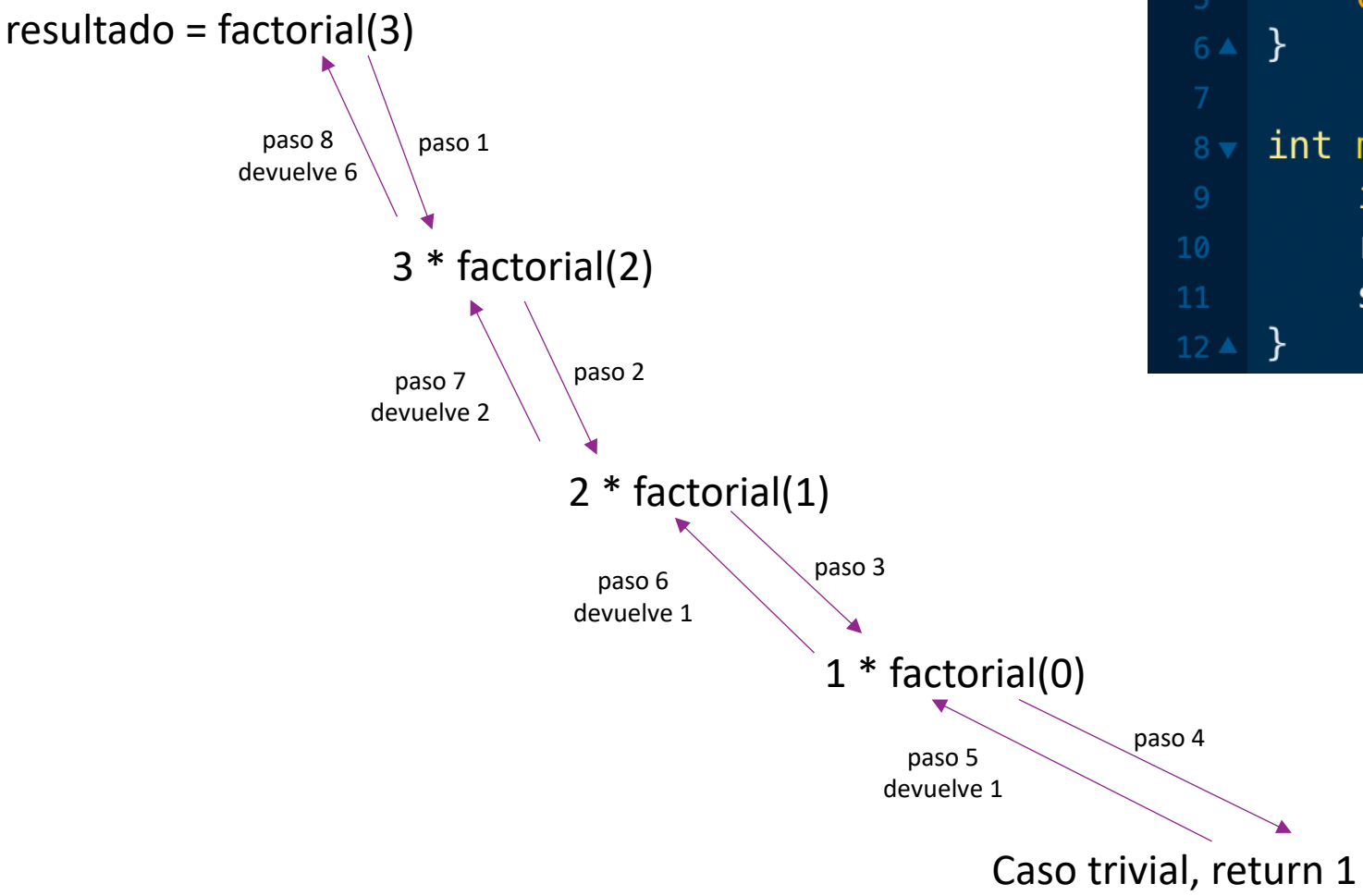
Iterativa: $n! = \prod_{i=1}^{n} i$

```cpp
#include<iostream>

int factorial(int n) {
    int i, resultado;
    resultado = 1;
    for (i = n; i > 1; i--)
        resultado *= i;
    return(resultado);
}

int main(void) {
    int resultado;
    resultado = factorial(3);
    std::cout << resultado << std::endl;
}
```

```asm
    .text
# Funcion factorial
# Input: $a0 numero a calcular factorial
# Output: $v0 = Factorial de la entrada. ($a0)!
factorial:
    li $v0,1 # en $v0 se devuelve el resultado
    # $t1 iterador
    move $t1,$a0
    loop:   blt $t1,2,finloop
            mul $v0,$v0,$t1
            addi $t1,-1
            j loop
    finloop:
        jr $ra # return

main:
    li $a0,3
    jal factorial
    # el valor se devuelve en $v0
    # print
    move $a0,$v0
    li $v0,1
    syscall

    li $v0,10
    syscall
```

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

```cpp
#include<iostream>

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

int main(void) {
    int resultado;
    resultado = factorial(3);
    std::cout << resultado << std::endl;
}
```



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

```cpp
#include<iostream>

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

int main(void) {
    int resultado;
    resultado = factorial(3);
    std::cout << resultado << std::endl;
}
```

```
    .text
# Funcion factorial
# Input: $a0 numero a calcular factorial
# Output: $v0 = Factorial de la entrada. ($a0)!
factorial:
    bne $a0,0,notrivial
        li $v0,1 # en $v0 se devuelve el resultado
        jr $ra #return
    notrivial:
        move $t1,$a0
        addi $a0,-1
        jal factorial
        # en $v0 esta el resultado de esta ultima llamada
        mul $v0,$t1,$v0
        jr $ra # return

main:
    li $a0,3
    jal factorial
    # el valor se devuelve en $v0
    # print
    move $a0,$v0
    li $v0,1
    syscall

    li $v0,10
    syscall
```

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

```cpp
#include<iostream>

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

int main(void) {
    int resultado;
    resultado = factorial(3);
    std::cout << resultado << std::endl;
}
```

```
    .text
# Funcion factorial
# Input: $a0 numero a calcular factorial
# Output: $v0 = Factorial de la entrada. ($a0)!
factorial:
    bne $a0,0,notrivial
        li $v0,1 # en $v0 se devuelve el resultado
        jr $ra #return
    notrivial:
        move $t1,$a0
        addi $a0,-1
        jal factorial
        # en $v0 esta el resultado de esta ultima llamada
        mul $v0,$t1,$v0
        jr $ra # return

main:
    li $a0,3
    jal factorial
    # el valor se devuelve en $v0
    # print
    move $a0,$v0
    li $v0,1
    syscall

    li $v0,10
    syscall
```

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

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

```cpp
#include<iostream>

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

int main(void) {
    int resultado;
    resultado = factorial(3);
    std::cout << resultado << std::endl;
}
```

```
    .text
# Funcion factorial
# Input: $a0 numero a calcular factorial
# Output: $v0 = Factorial de la entrada. ($a0)!
factorial:  bgt $a0,$zero,notrivial
            li  $v0,1
            jr  $ra
    notrivial:
        push    addi    $sp, $sp, -8    # reservo 8 bytes en la pila
                sw $ra, 4($sp)  # guardo $ra
                sw $a0, 0($sp)  # guardo $a0

                addi $a0,$a0,-1 # fact(n) = n * fact(n-1)
                jal factorial       # llamo a fact(n-1) devuelve en $v0

        pop     lw $a0,0($sp)       # recupero $a0 en el mismo orden
                lw $ra,4($sp)       # recupero $ra en el mismo orden
                addi $sp, $sp, 8    # dejo el puntero de pila como estaba

                mul $v0,$a0,$v0     # multiplico n*fact(n-1) que es $a0*$v0
                jr  $ra
         # FIN FUNCION factorial
main:
    li $a0,3
    jal factorial
    # el valor se devuelve en $v0
    # print
    move $a0,$v0
    li $v0,1
    syscall

    li $v0,10
    syscall
```