Recursividad

Curso de Estructuras de Datos y Algoritmos I

Prof. Luis E. Garreta U.

lgarreta@uao.edu.co

Universidad Autonoma de Occidente – Cali
Depto. Operaciones y Sistemas
Facultad de Ingeniería

27 de febrero de 2018

Funciones Recursivas

- La recursividad constituye una de las herramientas más potentes en programación.
- Es un concepto matemático muy conocido.
- Ejemplo de una definición función matemática recursiva:

$$n! = \begin{cases} 1 & \text{si } n = 0 \\ n \cdot (n-1)! & \text{si } n > 0 \end{cases}$$

Una función que se llama a sí misma se denomina recursiva

Cuando usar la Recursión

Podemos usar recursividad si:

1. La solución de un problema está expresada en función de si misma, aunque de menor tamaño y
2. Conocemos la solución no-recursiva para un determinado caso.

Ventajas y Desventajas Recursión

- **Ventajas**: No es necesario definir la secuencia de pasos exacta para resolver el problema.
- **Desventajas**: Podría ser menos eficiente.

Ejemplos de Funciones Recursivas

- Factorial de un número **n**:

$$n! = \begin{cases} 1 & \text{si } n = 0 \\ n \cdot (n-1)! & \text{si } n > 0 \end{cases}$$

- Potencia de un número **x** elevado a la **n**:

$$x^n = \begin{cases} 1 & \text{si } n = 0 \\ x \cdot x^{n-1} & \text{si } n > 0 \end{cases}$$

- Productos de dos números **a** y **b**:

$$producto(a, b) = \begin{cases} 0 & \text{si } b = 0 \\ a + producto(a, b-1) & \text{si } b > 0 \end{cases}$$

Elementos de una Función Recursiva

$$n! \begin{cases} 0 & \text{si } n = 0 \quad \Rightarrow \text{ CASO BASE} \\ n * (n-1) & \text{si } n > 0 \quad \Rightarrow \text{ LLAMADO RECURSIVO} \end{cases}$$

Ejemplo Calculo del Factorial

```
3! = 3 * 2!
          2! = 2 * 1!
                    1! = 1 * 0!
                                  0! = 1
                                  1
                    1! = 1 * 1
          2! = 2 * 1
3! = 3 * 2
```

```
3! = 3 * 2
   = 6
```

Diseño de Funciones Recursivas

- Primer paso será la identificación de un algoritmo recursivo
  - Descomponer el problema de forma que su solución quede definida en función de ella misma pero para un tamaño menor
  - y exista un caso simple (trivial) que resuelva el problema.
- Tendremos que diseñar:
  - casos base,
  - casos generales
  - y la solución en términos de ellos.

Caso Base

Para que una definición recursiva esté completamente identificada es necesario tener un **caso base**

$$n! \begin{cases} 0 & \text{si } n = 0 \quad \Rightarrow \text{ CASO BASE} \\ n * (n-1) & \text{si } n > 0 \quad \Rightarrow \end{cases}$$

- Generalmente es es caso trivial del problema.
- Se resuelve con un segmento de código sin recursividad:
  - Este no se calcula utilizando casos anteriores, y
- Siempre debe existir al menos un caso base
- Debería ser simple y eficiente.
- La división del problema **debe** converger a ese caso base.

Casos generales

Para que una definición recursiva avance en la solución del problema es necesario un **caso general o paso recursivo**

$$n! \begin{cases} 0 & \text{si } n = 0 \quad \Rightarrow \\ n * (n-1) & \text{si } n > 0 \quad \Rightarrow \text{ CASO GENERAL} \end{cases}$$

La solución se expresa de forma recursiva como la unión de:

- La solución de uno o más subproblemas:
  - De igual naturaleza pero menor tamaño
- Un conjunto de pasos adicionales

Los casos generales siempre deben avanzar hacia un caso base. Es decir, la llamada recursiva se hace a un subproblema más pequeño y, en última instancia, los casos generales alcanzarán un caso base.

Ejemplo Implementación Factorial

$$n! \begin{cases} 0 & \text{si } n = 0 \quad \Rightarrow \text{ CASO BASE} \\ n * (n-1) & \text{si } n > 0 \quad \Rightarrow \text{ CASO GENERAL} \end{cases}$$

```
// Version corta
int factorial (int n) {
if (n==0)  //Caso base
  return 1;
else  //Caso general
  return n*factorial(n
      -1);
}
```

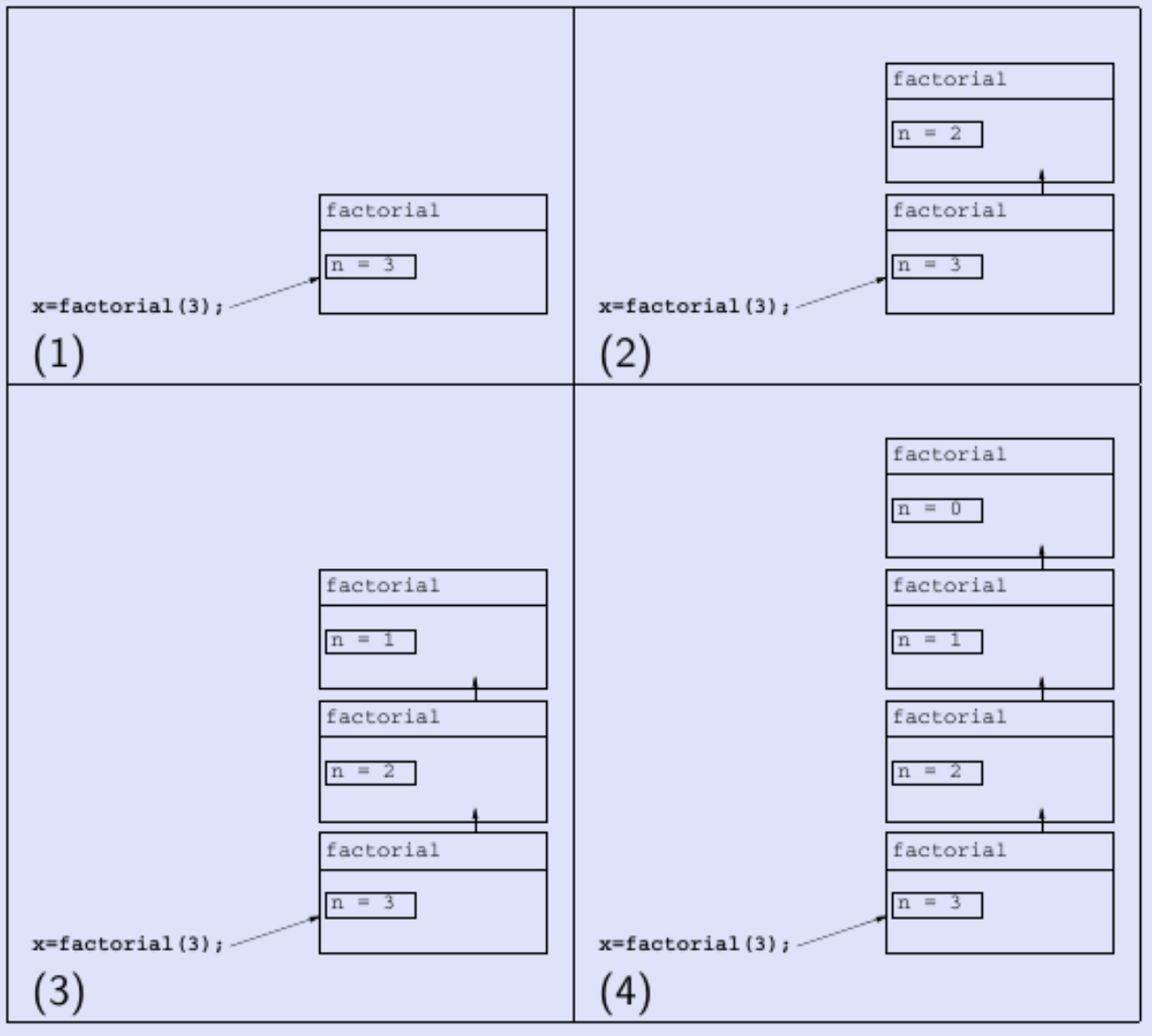
```
// Version larga
int factorial (int n) {
  int resultado;
  if (n==0) //Caso base
    resultado = 1;
  else      //Caso general
    resultado = n*factorial
        (n-1);

  return resultado;
}
```

Ejecución de un Función Recursiva

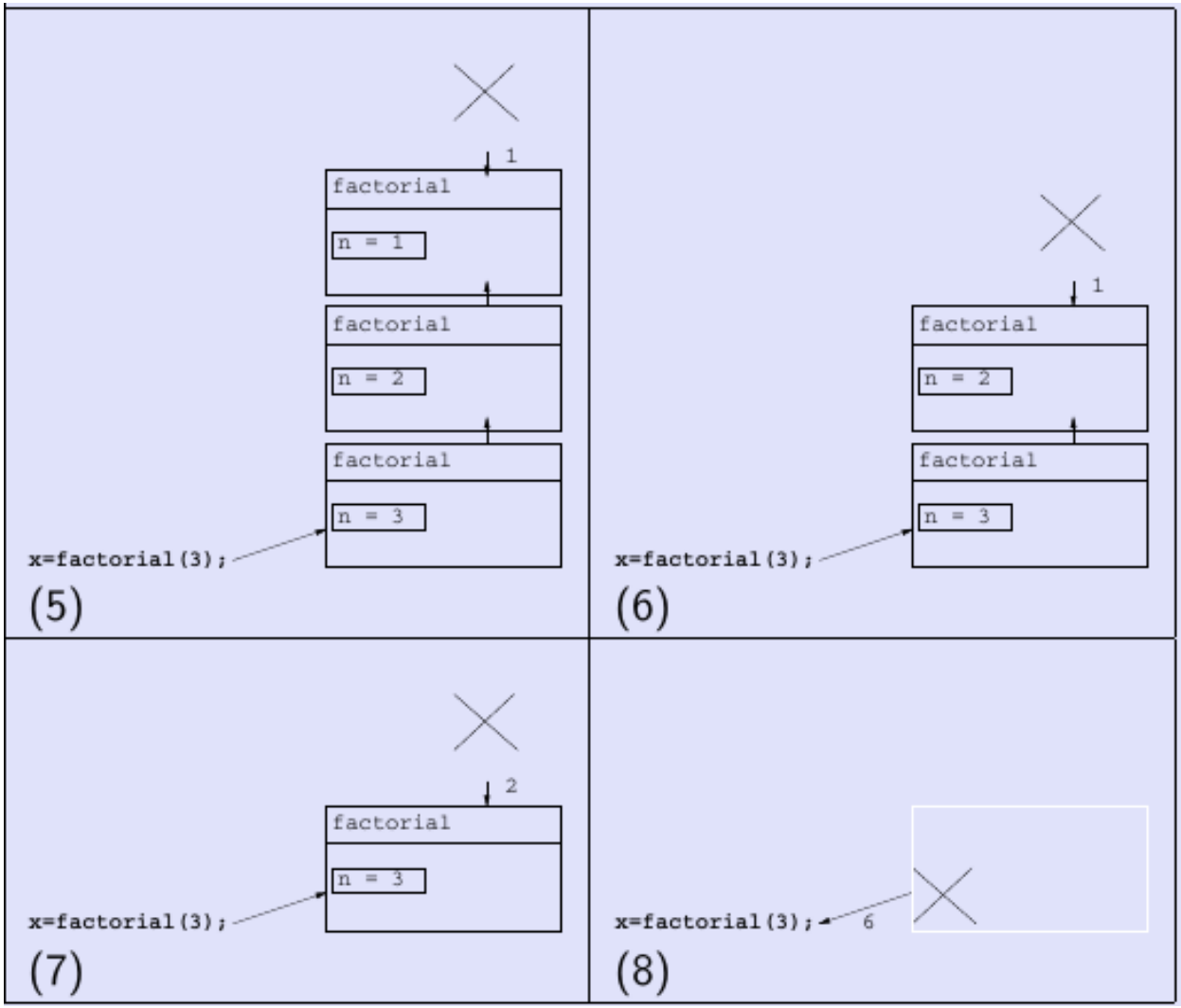
Qúe está pasando en la máquina?

- En general, en la pila se almacena el entorno asociado a las distintas funciones que se van activando.
- Cada llamada recursiva genera una nueva zona de memoria en la pila independiente del resto de llamadas.

Empilamiento de los llamados Recursivos



Desempilamiento de los llamados recursivos



Traza o Seguimiento de una Función Recursiva

factorial(3)

```
factorial(3) = 3*factorial (2)
                    factorial(2) = 2*factorial (1)
                                        factorial(1) = 1*factorial (0)
                                                            factorial (0) = 1
                                                            returna 1
                                        factorial(1) = 1 * 1
                                        returna 1
                    factorial(2) = 2 * 1
                    returna 2
factorial(3) = 3 * 2
returna 6
```