LECCIÓN 2: Funciones recursivas

María de la Paz Guerrero Lebrero

Curso 2014 / 2015

Grado en Matemáticas

maria.guerrero@uca.es

UCA
Universidad de Cádiz

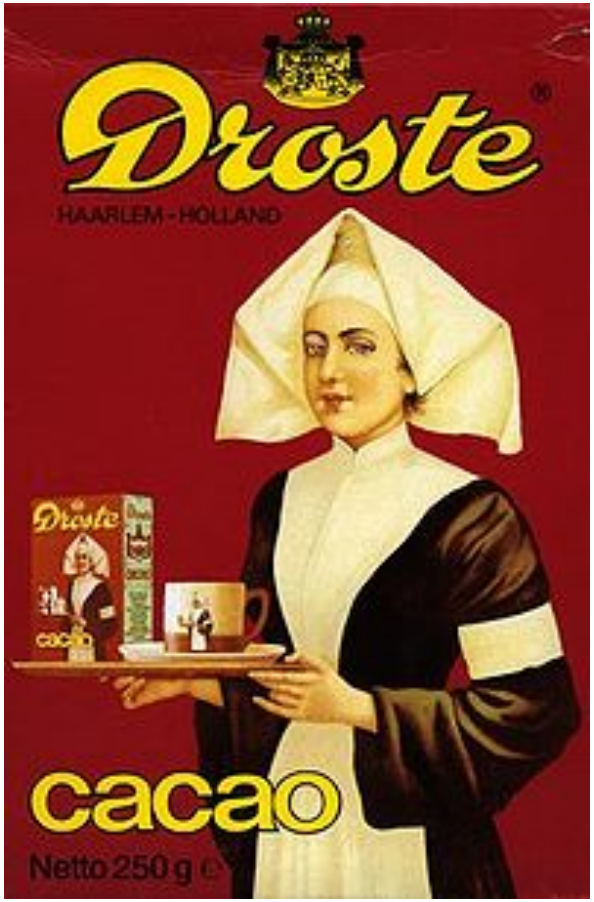
Índice

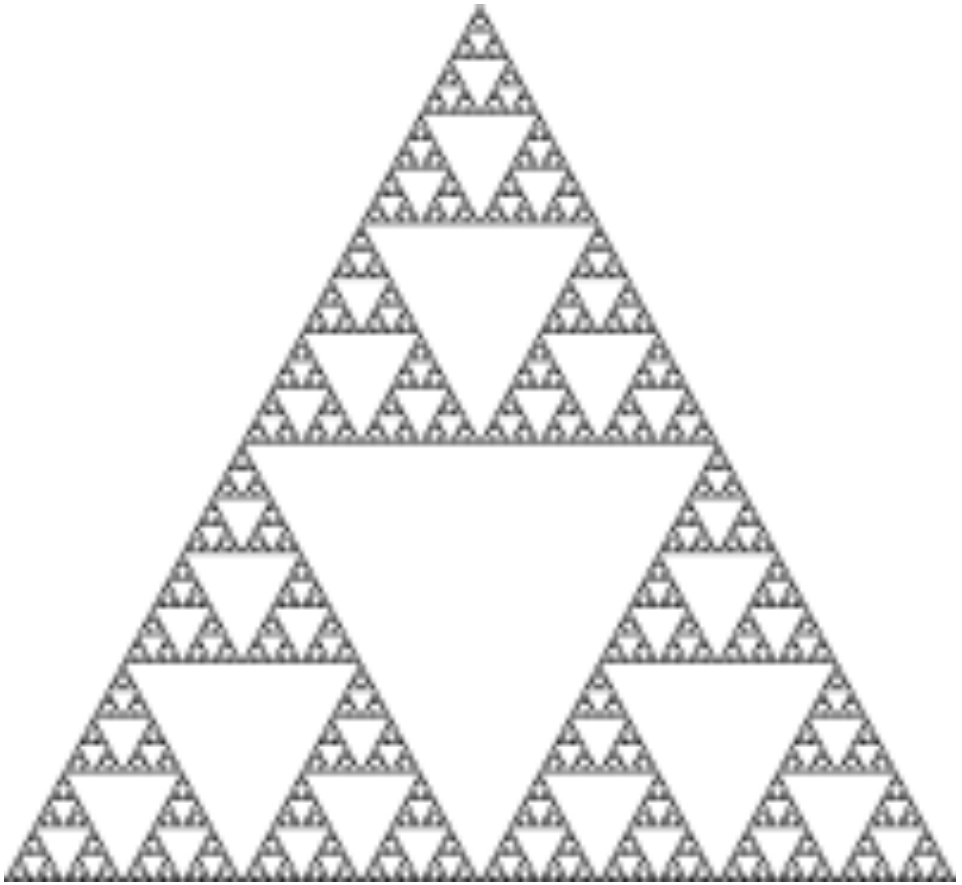
- Introducción
- ¿Qué es la recursividad?
- ¿Cómo funciona la recursividad?
- Propiedades de las funciones recursivas
- ¿Recursión o iteración?
- Tipos de funciones recursivas

Introducción

- El área de la programación es muy amplia y tiene muchos detalles.
- Es necesario poder resolver todos los problemas que se presenten aun cuando en el lenguaje que se utilice no haya una manera directa de resolver dichos problemas.
- En C, así como en otros lenguajes de programación, se puede aplicar una técnica llamada **recursividad,** cuyo nombre es debido a su funcionalidad.

Introducción





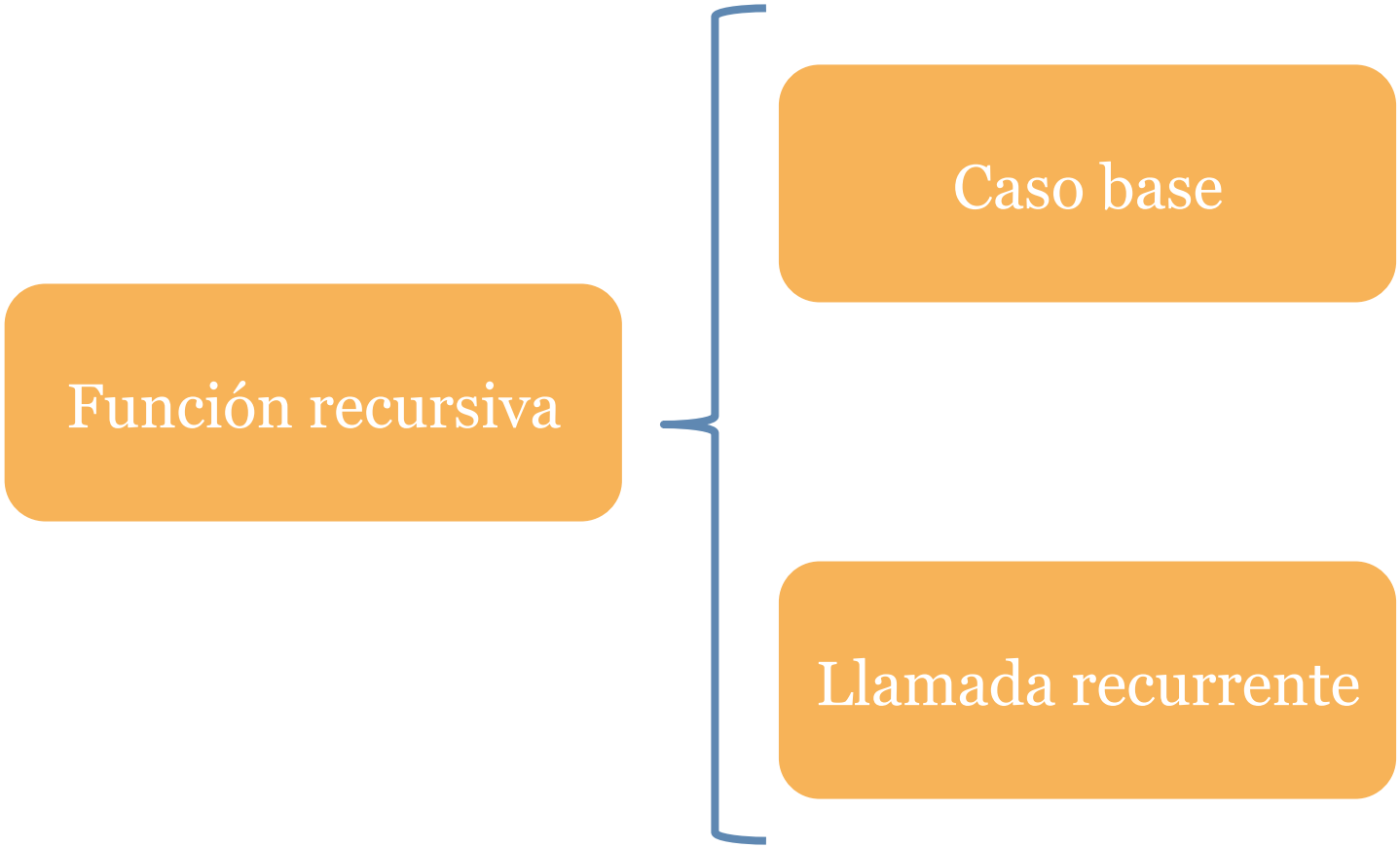
¿Qué es la recursividad?

- Definición
- Uso
- Ejemplos

Definición

- La recursividad o recurrencia es la forma en la cual se especifica un proceso basado en su propia definición.
- Un problema que pueda ser definido en función de su tamaño, sea este N, pueda ser dividido en instancias más pequeñas (< N) del mismo problema y se conozca la solución explícita a las instancias más simples, lo que se conoce como casos base, se puede aplicar inducción sobre las llamadas más pequeñas y suponer que estas quedan resueltas.

Definición



Uso

- La recursividad es una técnica de programación importante.
- Se utiliza para realizar una llamada a una función desde la misma función.
- La recursividad y la iteración (ejecución en bucle) están muy relacionadas, cualquier acción que pueda realizarse con la recursividad puede realizarse con iteración y viceversa.

Uso

- Normalmente, un cálculo determinado se prestará a una técnica u otra.
- Dependiendo del caso y de la habilidad de programador, la recursión puede ser más clara en términos de codificación.
- **CUIDADO!** Es fácil crear una función recursiva que no llegue a devolver nunca un resultado definitivo y no pueda llegar a un punto de finalización. Este tipo de recursividad hace que el sistema ejecute lo que se conoce como recursión "infinita".

Ejemplos

- **Factorial de un número:**

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

- Para calcular el factorial de cualquier número mayor que cero hay que calcular como mínimo el factorial de otro número. Para ello, se debe llamar de nuevo a la función para el número menor inmediato, de esta forma, se obtiene el factorial del número actual.

Ejemplos

- Código en C

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

Ejemplos

- **Función de Fibonacci:**

$$fib(n) = \begin{cases} \text{si } n = 0,\ n = 1 \rightarrow n \\ \text{si } n >= 2 \rightarrow fib(n-2) + fib(n-1) \end{cases}$$

- Obsérvese que la definición recursiva de los números de Fibonacci difiere de las definiciones recursivas de la función factorial. La definición recursiva de *fib* se refiere dos veces a sí misma .

Ejemplos

- Código en C

```
int fibonacci(int n)
{
      if ( n == 0 || n == 1 )
         return n;
      else
         return (fibonacci( n - 1 ) + fibonacci( n - 2 ));
}
```

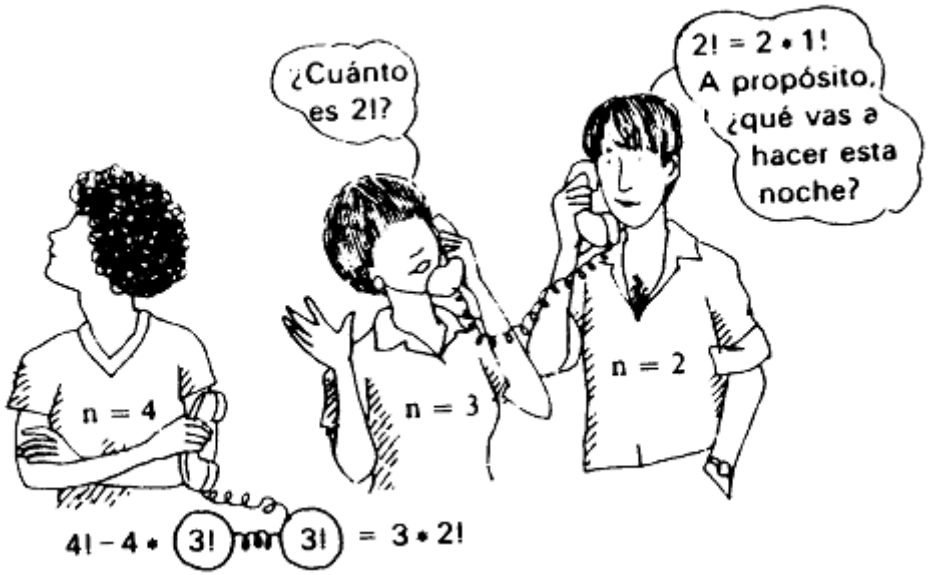
¿Cómo funciona la recursividad?

- Ejemplo de la función factorial

Factorial

- $4! = 4 * 3!$

- $3! = 3 * 2!$

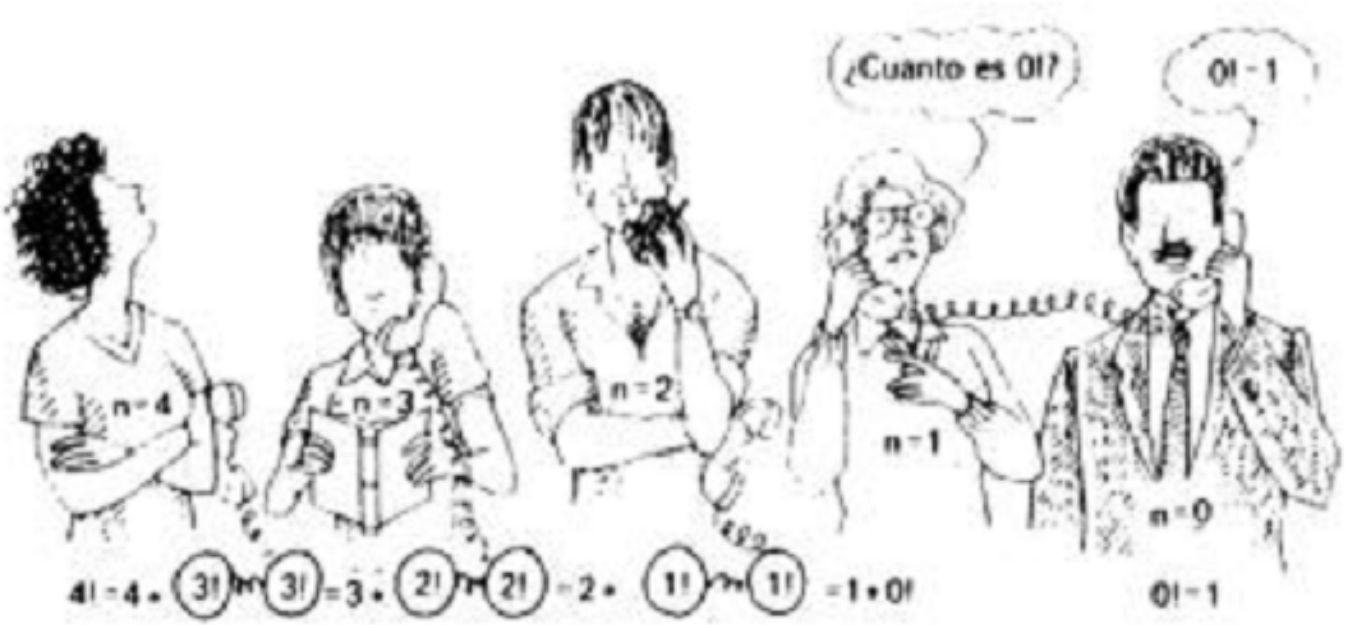


Factorial

- $2! = 2 * 1!$

Factorial

- $1! = 1 * 0! = 1 * 1$



Factorial

- Solución

Propiedades de las funciones recursivas

- Propiedad 1
- Propiedad 2

Propiedad 1

- Un algoritmo A que resuelve un problema P es recursivo si está basado *directa* o *indirectamente* en sí mismo.

Problema P con Datos I

↓↓

Resuelto en términos de …

↓↓

Problema P con Datos de I' perteneciente a I

Propiedad 2

- Un requisito importante para que sea correcto un algoritmo recursivo es que no genere una secuencia infinita de llamadas así mismo. Cualquier algoritmo que genere tal secuencia no termina nunca.
- Una función recursiva *f* debe definirse en términos que no impliquen a *f* al menos en un argumento o grupo de argumentos. Debe existir una "salida" de la secuencia de llamadas recursivas.

¿Recursión o iteración?

- Ventajas e inconvenientes
- Cuando usarlas

Ventajas e inconvenientes

- Ventajas de la recursividad:
  - Soluciones simples y claras
  - Soluciones elegantes
  - Soluciones a problemas complejos

Ventajas e inconvenientes

- Desventajas de la recursividad: **INEFICIENCIA**
  - Sobrecarga asociada a las llamadas recursivas:
    - Una simple llamada puede generar un gran numero de llamadas recursivas. (Fact(n) genera n llamadas recursivas)
    - ¿La claridad compensa la sobrecarga?
    - El valor de la recursividad reside en el hecho de que se puede usar para resolver problemas sin fácil solución iterativa.
  - La ineficiencia inherente de algunos algoritmos recursivos.

Cuando usarlas

La recursividad se debe usar cuando sea realmente necesaria, es decir, cuando no exista una solución iterativa simple

Tipos de funciones recursivas

- Recursividad lineal
- Recursividad múltiple

Recursividad lineal

- La recursión es lineal cuando cada llamada recursiva, genera, como mucho, otra llamada recursiva:
  - **Final:** si la llamada recursiva es la última operación que se efectúa, devolviéndose como resultado lo que se haya obtenido de la llamada recursiva sin modificación alguna.
  - **No final:** El resultado obtenido de la llamada recursiva se combina para dar lugar al resultado de la función que realiza la llamada.

Recursividad múltiple

- La recursión es múltiple cuando cada llamada recursiva, genera más de una llamada recursiva.

Ejercicios

1. Implementa una función recursiva que calcule el producto de dos números enteros.
2. Implementa una función recursiva que pase un número en base 10(decimal) a base 2 (binario).
3. Dados dos números a (número entero) y b (número natural mayor o igual que cero) determinar a^b.
4. Diseña un algoritmo recursivo que permita calcular la función de Ackermann de dos números enteros la cual se define:

$$\text{Ackermann}(n,m) = \begin{cases} \text{si } m = 0,\ n+1 \\ \text{si } n = 0,\ \text{Ackermann}(m-1,\ 1) \\ \text{Ackermann}(m-1,\ \text{Ackermann}(m,\ n-1)) \end{cases}$$