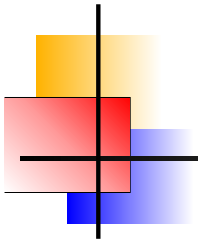


Algoritmos y Estructuras de Datos:

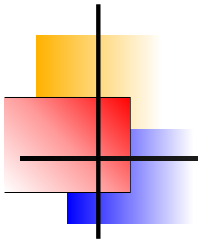
Introducción a la Recursión de Programas

Guillermo Román Díez

groman@fi.upm.es

Universidad Politécnica de Madrid

Curso 2015-2016



Métodos Recursivos

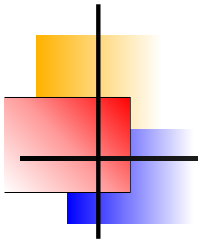
Pregunta

¿Qué es un método recursivo?

- Un método recursivo es un método que se invoca a sí mismo
- El ejemplo típico es el *factorial*(n)
  - *factorial*(n) se calcula $n * (n-1) * (n-2) * (n-3) * \ldots * 1$
  - Observad que la expresión $(n-1) * (n-2) * (n-3) \ldots * 1$ es el *factorial*$(n-1)$

```
n * (n-1) * (n-2) * ... * 1
    \_____________________/
          factorial(n-1)
\_________________________/
       factorial(n)

factorial(n) = n * factorial(n-1)
```



Recursión: Factorial

Ejercicio

Escribir una versión iterativa y otra recursiva de `factorial(n)`

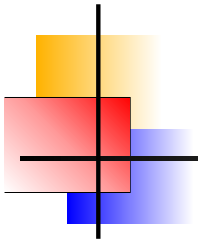
Versión Iterativa

```
int factorial(int n) {
  int r = 1;
  while (n > 1) {
    r = n * r;
    n--;
  }
  return r;
}
```

Versión Recursiva

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

- Cualquier programa iterativo tiene uno equivalente recursivo y viceversa
- Llamarse recursivamente es una forma de iterar



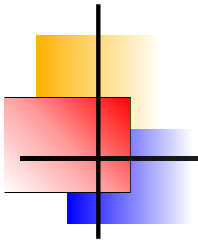
Ejemplo de Ejecución

Ejercicio

Dibujar la ejecución recursiva de `factorial(4)`

```
factorial(4)
4 *   factorial(3)
4 *   3 *   factorial(2)
4 *   3 *   2 *   factorial(1)
4 *   3 *   2 *   1
4 *   3 *   2
4 *   6
24
```

- Vemos que las llamadas recursivas se van **apilando**
- Hasta llegar al **caso base** que se van resolviendo hasta llegar a la primera llamada



Recursión de Cola

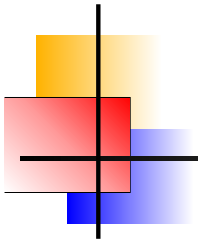
- La recursión se suele implementar utilizando la pila de la memoria
- Se puede implementar a estilo **paso de continuaciones** dejando **recursión de cola**
- Para esto pasamos un parámetro extra sobre el que vamos acumulando el resultado
- La llamada recursiva sería el último mandato del programa

Recursión "normal"

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

Recursión de Cola

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



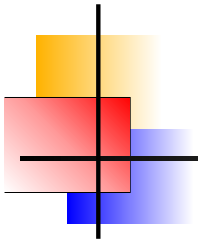
Errores Comunes

- La función debe tener uno o más casos base (no recursivos)

```
public int factorial(int n) {
   return n * factorial(n-1);    // no hay caso base
}
```

- Debe haber un orden bien fundado en los valores de los argumentos
  - Las llamadas recursivas se realizan sobre valores anteriores y eventualmente se llega a alguno de los casos base

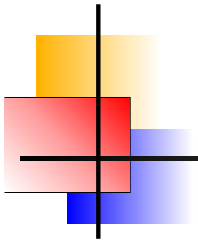
```
public int factorial(int n) {
  if (n == 1)
    return 1;
  else
   return n * factorial(n-2); // no llega al caso base
}
```



Recursión Vs. Inducción

- La recursión es el reverso de la inducción
- Podemos demostrar que $factorial(n) = n * (n-1) * \ldots * 1$ para todo $n > 0$.
  - Caso base: factorial$(1) = 1$
  - Hipótesis de inducción: $factorial(n) = n * (n-1) * \ldots * 1$
    - Demostramos que se cumple $factorial(n+1) = (n+1) * n * (n-1) * \ldots * 1$

```
  factorial(n+1)
= { por la definicion de factorial con n > 0}
  (n+1) * factorial((n+1)-1)
= { aritmetica }
  (n+1) * factorial(n)
= { por la hipotesis de induccion
      factorial(n) = n * (n-1) * ... * 1 }
  (n+1) * n * (n-1) * ... * 1
```



Ejemplo: Máximo Común Divisor

Ejercicio

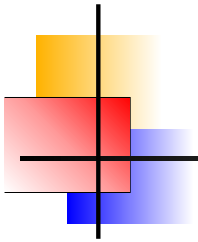
Función que calcule el Máximo Común Divisor

Versión Iterativa

```
int mcd(int n, int m) {
  while (m != 0) {
    int tmp = m;
    m       = n % m;
    n       = tmp;
  }
  return n;
}
```

Versión Recursiva

```
int mcd(int n, int m) {
  if (m == 0)
    return n;
  else
    return mcd(m, n % m);
}
```

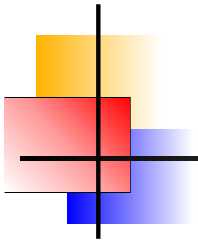


Ejemplo: Fibonacci

Ejercicio

Función que calcule el n-ésimo elemento de la sucesión de Fibonacci

```
public static int fib(int n) {
  if (n < 0)  return 0;
  if (n <= 1) return n;    /* dos casos base */
  return fib(n-1) + fib(n-2);
}
```

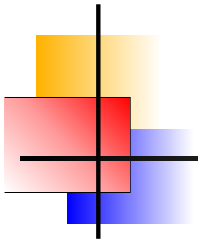


Búsqueda Binaria

Ejercicio

Implementar de forma iterativa el algoritmo de búsqueda binaria sobre un vector ordenado

```
public boolean member(int elem, int [] arr) {
  if (arr == null    || arr.length == 0 ||
      elem < arr[0] || elem > arr[arr.length - 1])
    return false;
  int start = 0;
  int end   = arr.length - 1;
  int m;
  while (start <= end && arr[(m = (start + end) / 2)] !=
      elem) {
    if (elem < arr[m])
      end   = m-1;
    else
      start = m+1;
  }
  // start > end || start <= end && arr[m] == elem
  return start <= end;
}
```

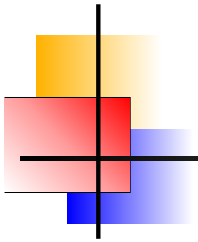


Búsqueda Binaria: Recursiva

- Es necesario un método auxiliar que lleve la cuenta del rango en el que estamos buscando [start–end]

```
public static boolean member(int elem, int [] arr) {
  if (elem < arr[0] || elem > arr[arr.length - 1])
    return false;
  else
    return memberRec(elem, arr, 0, arr.length-1);
}

private static boolean memberRec(int elem, int [] arr, int
    start, int end) {
  if (start > end)     return false;    /* caso base */
  int m = (start + end) / 2;
  if (elem == arr[m]) return true;     /* caso base */
  if (elem < arr[m]) return memberRec(elem,arr,start,m-1);
                     return memberRec(elem,arr,m+1,end);
}
```



Recursión sobre Listas de Posiciones

- Las listas de posiciones se pueden definir de forma recursiva
  - Como una secuencia de nodos que, o bien es vacía, o está formada por un nodo seguido por una secuencia de nodos
- Podemos definir métodos recursivos para explotar esta definición

```
public static <E> void show(PositionList<E> list) {
  if (list != null) showRec(list, list.first());
}

private static <E> void showRec(PositionList<E> list,
    Position<E> cursor) {
  if (cursor != null) {
    System.out.println(cursor.element());
    showRec(list, list.next(cursor));
  }
}
```