



DEPARTAMENTO DE INFORMÁTICA  
UNIVERSIDAD CARLOS III DE MADRID

# Ingeniería en Informática

## Paradigmas de Programación

Marzo 2008

### Hoja de Ejercicios 2: Problemas de recursividad en Lisp

#### Comentarios generales sobre los ejercicios

- Es una buena idea pensar y razonar las soluciones antes de intentar programarlas
- Para la resolución de los problemas, sólo pueden emplearse las funciones de LISP vistas en teoría
- **Todos** los ejercicios de esta hoja deben resolverse usando funciones recursivas
- Describir las soluciones a los ejercicios de la manera más formal posible

1. Escribir una función que calcule el factorial de un número natural
2. Definir la función de `fibonacci` recordando que:
  - `fibonacci(0) = fibonacci(1) = 1`
  - `fibonacci(n) = fibonacci(n - 1) + fibonacci(n - 2)`Tener en cuenta su eficiencia computacional a la hora de escribirla.
3. Escribir una función que calcule el valor de la función  $a^b$  recursivamente (esto es, no se permite usar la función `expt`), siendo  $a$  y  $b$  números enteros **cualesquiera**
4. Escribir funciones que calculen el valor de los operadores booleanos `and`, `or`, `xor` (binario) y `not` (unario), para un número cualquiera de operandos
5. Escribir una función recursiva que compruebe si un número es primo
6. Escribir la función `cuenta-pares` que recibe una lista y devuelve cuantos números pares tiene

```
> (cuenta-pares '(1 2 3 4))
> 2
```
7. Definir una función que calcule el número de términos que siguen a un átomo dado dentro de una lista
8. Escribir la función `get-posicion` que determina la posición de la primera aparición de un elemento en una lista

```
> (get-posicion 'c '(a b c d))
> 3
```
9. Escribir la función `invertir` que invierte el contenido de una lista

```
> (invertir '(1 2 3 4))
> (4 3 2 1)
```

10. Escribir la función `mi-butlast` que tome una lista y un número natural `n` y retorne la lista original sin los últimos `n` elementos. **No está permitido usar la función `reverse` de Lisp**

11. Definir la función `palíndromo` que crea el palíndromo de una lista dada como argumento. Un palíndromo es una estructura capicúa que, por lo tanto, se lee igual de izquierda a derecha que de derecha a izquierda. **No está permitido usar la función `reverse` de Lisp**

12. Escribir la función `get-numeros` que extrae todos los números que aparecen en una lista

```
> (get-numeros '((1 (2)) a (((5 c 7))) 4))
> (1 2 5 7 4)
```

13. Definir de forma **recursiva** la función `primero-y-ultimo` que toma como argumento una lista y que devuelva otra lista con el primer y último elemento de la lista

14. Definir de forma **recursiva** la función `mi-reverse` que toma como entrada una lista y devuelve la misma lista en orden inverso

```
> (mi-reverse '(es una lista))
> (LISTA UNA ES)
```

15. Definir de forma **recursiva** la función `mi-reverse-arbol` que toma como entrada una lista y devuelve la misma lista en orden inverso en cada subnivel de la lista

```
> (mi-reverse-arbol '(es una lista))
> (LISTA UNA ES)
> (mi-reverse-arbol '(es una (esto es una sublista)))
> ((SUBLISTA UNA ES ESTO) UNA ES)
```

16. Definir de forma **recursiva** la función `mi-member-p` que toma como entrada una expresión LISP y una lista y devuelve T si la expresión pertenece a la lista del primer nivel y NIL en otro caso

```
> (mi-member-p 'es '(es una lista))
> T
> (mi-member-p '(es sublista) '(es una (es sublista)))
> T
> (mi-member-p 'es '((es) una lista))
> NIL
```

17. Definir de forma **recursiva** la función `mi-member-arbol-p` que toma como entrada una expresión LISP y una lista y devuelve T si la expresión pertenece a la lista en cualquiera de sus niveles y NIL en otro caso

```
> (mi-member-arbol-p 'es '(es una lista))
> T
> (mi-member-arbol-p '(es sublista) '(es una (es sublista)))
> T
> (mi-member-arbol-p 'es '((es) una lista))
> T
```

18. Definir de forma **recursiva** la función `mi-member` que toma como entrada una expresión LISP y una lista y devuelve la lista que comienza con la primera ocurrencia de la expresión especificada y NIL en otro caso

```
> (mi-member 'es '(es una lista))
> (ES UNA LISTA)
> (mi-member '(es sublista) '(es una (es sublista)))
> ((ES SUBLISTA))
> (mi-member 'es '((es) una lista))
> NIL
```

19. Escribir una función denominada **evalúa-expresión** que tome una expresión aritmética en notación infija con los operadores +, -, \* y / en una lista de entrada, y retorne el resultado de su evaluación. La prioridad de los operadores será la típica, esto es, multiplicativos sobre aditivos
20. Elegir un modelo factible para la representación de árboles binario y escribir sendas funciones para recorrer este árbol en in-orden, pre-orden y post-orden. El resultado del recorrido debe devolverse en una lista. En cada nodo del árbol se tendrá un elemento (que puede ser un símbolo, número, lista, etc.) así como los hijos del nodo
21. Dada una lista de números enteros positivos, escribir algoritmos recursivos que computen:
  - a) El máximo de un elemento de la lista, o -1 si la lista está vacía
  - b) El mínimo elemento de la lista, o -1 si la lista está vacía
  - c) La suma de los elementos de la lista, o -1 si la lista está vacía
  - d) El producto de los elementos de la lista, o -1 si la lista está vacía
  - e) La media de los elementos de la lista, o -1 si la lista está vacía

22. Implementar una función recursiva que calcule la longitud de la mayor secuencia de 1s en una lista que contiene únicamente 0s y 1s

```
> (max-ones-fwd '(1 1 0 0 1 0 0 0 1 1 1 1 0 0 1 1 1))
4
```

23. Programar la función (**sublist L inicio fin**) de modo recursivo que devuelva la sublista de L con los elementos en el intervalo [inicio,fin) donde la primera posición tiene el índice 0

```
> (sublist '(1 2 3 4 5) 2 4)
(3 4)
```

24. El método de ordenación de intercambio directo se basa en la comparación sucesiva e intercambio de elementos adyacentes hasta que, por fin, esté ordenada la lista. En cada paso se van comparando, empezando por el final, cada elemento con el anterior y, en caso de estar desordenados, se intercambian. De esta forma, al final del primer paso quedará el menor elemento en la posición más a la izquierda, en el segundo el menor del resto, quedará situado en la segunda posición, etc. Al final, la lista quedará ordenada en (n-1) pasos donde n es el número de elementos de la lista L.

Se pide implementar una función recursiva que ordene una lista de numeros L por el método de intercambio directo de menor a mayor

25. Programar una función de ordenación de una lista por la técnica de Divide y Vencerás con Merge Sort:  
Merge Sort:

- a) Sea  $k$  el elemento medio del vector
- b) Ordenar los elementos  $A_1, A_k$
- c) Ordenar los elementos  $A_{k+1}, A_n$
- d) Mezclar ambos

26. Una forma de calcular el máximo común divisor (mcd) de dos números es la siguiente:

$$\begin{aligned}
 MCD(x, y) &= y && \text{si } y \leq x, x \bmod y = 0 \\
 MCD(x, y) &= MCD(y, x) && \text{si } x < y \\
 MCD(x, y) &= MCD(y, x \bmod y) && \text{en otro caso}
 \end{aligned}$$

Escribir una función para calcularlo

27. Escribir una función `prof-elementos` que indique cuál es la profundidad de cada elemento de una lista (es decir, el nivel de la lista anidada en la que aparece). Para ello devolverá una lista en la que cada elemento es una lista de exactamente dos elementos: el elemento en cuestión y su profundidad. Se considera que los elementos del primer nivel tienen profundidad 1

```
> (prof-elementos '( 3 (1 (2)) ((( 5 7))) 4))  
> ((3 1) (1 2) (2 3) (5 4) (7 4) (4 1))
```