



DEPARTAMENTO DE INFORMÁTICA
UNIVERSIDAD CARLOS III DE MADRID

Ingeniería en Informática

Paradigmas de Programación

Marzo 2008

Hoja de Ejercicios 3: Problemas de orden superior 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 de orden superior.
- Describir las soluciones a los ejercicios de la manera más formal posible

1. Definir una función `swap` que, aplicada a una función cuyo argumento es un par de elementos, intercambie el orden de sus componentes: $\text{Swap}(f(x, y)) = f(y, x)$

```
> (defun funcion (a b) (/ a b))  
FUNCION  
> (funcall (swap #'funcion) 3 7)  
7/3
```

```
; Swap  
; (elemento * elemento -> elemento) -> (elemento * elemento -> elemento)  
(defun swap (f)  
  #'(lambda(x y) (funcall f y x)))
```

2. Escribir la función `calcula-cuadrados` que recibe una lista de números y devuelve otra lista con el cuadrado de cada elemento

```
> (calcula-cuadrados '(1 2 3 4))  
> (1 4 9 16)
```

```
(defun calcula-cuadrados (lista)  
  (mapcar #'(lambda(x) (* x x)) lista))
```

3. Escribir la función `resta-listas` que devuelve una lista con los elementos de la primera lista que no aparecen en la segunda

```
> (resta-listas '(1 2 3 4) '(2 3))
> (1 4)
```

```
; Sin el uso de funciones de orden superior puede hacerse de la
; siguiente manera
(defun resta-lista (l1 l2)
  (cond ((null l1) nil)
        ((numberp l1) (unless (member l1 l2) (list l1)))
        ((cdr l1) (append (resta-lista (car l1) l2) (resta-lista (cdr l1) l2)))
        (t (resta-lista (car l1) l2))))

; Sin embargo, con funciones de orden superior es, simplemente
(defun resta-lista2 (l1 l2)
  (remove-if #'(lambda(x) (member x l2)) l1))
```

4. Escribir la función `get-profundidad` que obtiene el número máximo de listas anidadas que aparecen en una lista

```
> (get-profundidad '((1 (2)) ((5 7))) 4))
> 3
```

```
(defun get-profundidad (lista)
  (apply #'max (mapcar #'(lambda (x) (extrae-profundidad x 0)) lista)))

(defun extrae-profundidad (lista nivel)
  (if (not (listp lista))
      nivel
      (apply #'max
              (mapcar #'(lambda (x) (extrae-profundidad x (+ 1 nivel))) lista))))
```

5. 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:

- Programar una función llamada `intercambio-directo-sup` (L p) que ordene los elementos de una lista L de acuerdo con el predicado de comparación p . ¿Cómo debe invocarse `intercambio-directo-sup` para que ordene números de menor a mayor? ¿y de mayor a menor?
- Programar una expresión que emplee `intercambio-directo-sup` y una función lambda en lugar del predicado p para que el resultado ponga primero los números pares de la lista L (en cualquier orden) y luego los números impares (en cualquier orden).

```

(defun intercambio-directo-sup-aux (L p)
  (cond ((null L)
        L)
        ((= (length L) 1)
         L)
        (t
         (append (if (funcall p
                              (car L)
                              (car (intercambio-directo-sup-aux (cdr L) p)))
                    (list (car L)
                          (car (intercambio-directo-sup-aux (cdr L) p)))
                    (list (car (intercambio-directo-sup-aux (cdr L) p))
                          (car L)))
                  (cdr (intercambio-directo-sup-aux (cdr L) p))))))

(defun intercambio-directo-sup (L p)
  (cond ((null L)
        L)
        ((= (length L) 1)
         L)
        (t
         (cons (car (intercambio-directo-sup-aux L p))
               (intercambio-directo-sup (cdr (intercambio-directo-sup-aux L p)) p))))))

```

; Estas funciones pueden invocarse como:

```

(intercambio-directo-sup L #'<)
(intercambio-directo-sup L #'>)

(intercambio-directo-sup L #'(lambda (x y) (evenp x)))

```

6. Definir una función de orden superior `maneja-lista-sup` (`L f`) que recibe en `L` una lista de números enteros positivos y en `f` una función:

- a) Si la lista no tiene elementos, `maneja-lista-sup` (`L f`) devuelve -1
- b) Si la lista tiene un único elemento, `maneja-lista-sup` (`L f`) devuelve el único elemento de la lista
- c) Si la lista tiene más elementos aplica la función `f` recursivamente al primer elemento de la lista y el resto de ella

¿De qué maneras debe invocarse la función `maneja-lista-sup` para que calcule el máximo, mínimo, suma y producto de los elementos de la lista `L`?

```

(defun maneja-lista-sup (L f)
  (cond ((null L)
        -1)
        ((= (length L) 1)
         (car L))
        (t
         (funcall f
                  (car L)
                  (maneja-lista-sup (cdr L) f))))))

(defun max-lista-sup (L)
  (maneja-lista-sup L #'max))

```

```
(defun min-lista-sup (L)
  (maneja-lista-sup L #'min))
```

```
(defun suma-lista-sup (L)
  (maneja-lista-sup L #'+))
```

```
(defun prod-lista-sup (L)
  (maneja-lista-sup L #'*))
```
