Ejercicios de Scheme (I)
Ingeniería en Desarrollo de Contenidos Digitales
Prácticas

Álvaro Castro-Castilla

Febrero 2012

1 Algoritmos con listas

1. Selecciona un elemento con un índice, contando desde la derecha
   (define (list-ref-right l k) ...)

2. ¿Cómo seleccionamos un elemento con un índice contando desde la derecha empleando una función definida en R5RS?

3. Imagina que una lista es circular, es decir, que una vez que llegamos al último elemento, vuelve a comenzar. ¿Cómo haríamos que un algoritmo como el anterior funcionase con una de estas listas? Dicho de otro modo, si k (el índice) es mayor que la longitud de la lista, tendría que comenzar de nuevo desde el principio.

   Este tipo de comportamiento puede ser útil al trabajar con imágenes. Intenta pensar por qué.

   (define (list-ref-circular l k) ...)

4. Busca, si no lo recuerdas, qué es un *palíndrome* y aplica el concepto a una lista de elementos. Crea una función que te diga si una lista es un palíndrome o no.

   (define (palindrome? l) ...)

5. Rota una lista hacia la izquierda *k veces*, cogiendo la cabeza y poniéndola en la cola.

```
(rotate-right k lis)
(rotate-left '(a b c d e) 1) -> '(b c d e a)
```

6. Rota una lista hacia la derecha *k veces*, cogiendo el último elemento y poniéndolo en la cabeza.

```
(rotate-right k lis)
(rotate-right '(a b c d e) 1) -> '(e a b c d)
```

7. Inserta un elemento en una posición n determinada

```
(insert-at new k lis)
(insert-at 'N 2 '(a b c d e f)) -> '(a b N c d e f)
```

8. Inserta un elemento a la izquierda de todos los elementos que coincidan con el argumento *e*.

```
(insert-left new e lis)
(insert-left 'N c '(a b c d e c f)) -> '(a b N c d e N c f)
```

9. Haz el algoritmo equivalente recursivo, para que trabaje en listas de distinta profundidad.

```
(insert-left* new e lis)
(insert-left* 'N c '(a (b c) d e ((c f)))) -> '(a (b N c) d e ((N c f)))
```

10. Inserta un elemento a la derecha de todos los elementos que coincidan con el argumento *e*.

```
(insert-right new e lis)
(insert-right 'N c '(a b c d e c f)) -> '(a b c N d e c N f)
```

11. Haz el algoritmo equivalente recursivo, para que trabaje en listas de distinta profundidad.

```
(insert-right* new e lis)
(insert-right* 'N c '(a (b c) d e ((c f)))) -> '(a (b c N) d e ((c N f)))
```

12. Elimina el elemento que esté situado en la posición k de la lista.

```
(remove-at k lis)
(remove-at 3 '(a b c d e f)) -> '(a b c e f)
```

13. Construye una nueva lista que repita cada elemento un número determinado de veces

```
(replicate n l)
(replicate '(a b c) 2) -> '(a a a b b b c c c)
```

14. Agrupa todos los elementos iguales y consecutivos de una lista en una sublista. La nueva lista contendrá el conunto de nuevas sublistas.

```
(pack l)
(pack '(a a a a b b c c c)) -> '((a a a a) (b b) (c c c))
```

15. Haz n sublistas de una lista, con igual número de elementos. Si no es divisible, el último grupo tendrá menos elementos.

```
(n-groups n l)
(n-groups 2 '(a b c d e f g)) -> '((a b c d) (e f g))
```

16. Haz grupos con n elementos cada uno, partiendo de una lista. Si no se pueden poner n elementos en todas las sublistas, la última tendrá menos.

```
(group n l)
(group 2 '(a b c d e f g)) -> '((a b) (c d) (e f) (g))
```

17. Al igual que en el anterior ejercicio, construye un algoritmo que agrupo en sublistas, pero dando una lista con un serie de número que indican la cantidad de elementos que debe haber en cada sublista (en el mismo orden). Propón qué podría hacer el algoritmo en caso de que no coincida el número de elementos indicados por g con el número de elementos de l.

```
(group g l)
(group '(2 3 1 1) '(a b c d e f g)) -> '((a b) (c d e) (f) (g))
```

18. Ordena una lista de números de menor a mayor. No importa el método que emplees: halla tú un método aunque no sea muy eficiente.

    (define (my-sort l) ...)

    Cuando lo hayas realizado, observa que hay un punto de la función en el que estás empleando la función $<$ o $>$. Esto es inevitable, porque si no, no podrías saber que número es mayor de cada dos que compares. Por eso, a esto se le llama comparador.

    Extrae el comparador como un elemento independiente, es decir, como un argumento de la función, y úsala con este nuevo argumento que se introduce desde fuera. A ese comparador lo llamaremos *less?*

    (define (my-sort l less?) ...)

    Ahora investiga un algoritmo de ordenación (el que quieras) e impleméntalo tú mismo. Algunos ejemplos:

    (define (quicksort l less?) ...)
    (define (mergesort l less?) ...)
    (define (introsort l less?) ...)

2 Matemáticas

1. Calcula la media aritmética de una lista de números.

   (define (mean l) ...)

2. De una lista de números extraer una lista que contenga solo el elemento que más veces se repite. Extraemos una lista por si hay varios elementos que se repiten igual número de veces (en cuyo caso esa lista de salida contendrá todos esos elementos "empatados" a máximas repeticiones). Intenta imaginar un caso en el que usarías esto para un problema real y descríbelo.

   Esto se llama "moda", y si el resultado te da dos números, el conjunto de números inicial es bimodal; si te da 3, trimodal... todos ellos son "multimodales".

   (define (mode l) ...)

3. Realizar una función que reciba una lista de números y calcule el número que, una vez ordenados, se encuentra a medio camino entre el

menor y el mayor.

A esto se le denomina "mediana", y es el valor medio de un conjunto de valores ordenados. Intenta buscar un caso práctico de la vida real en el que hayas empleado este cálculo de manera consciente o no.

Primero emplea una lista de números que ya esté ordenada. Luego preocúpate de ordenarla con el algoritmo que has diseñado anteriormente *(sort)*.

(define (median l) ...)

4. Implementa la media geométrica:

   $G(x_1, \ldots, x_n) = \sqrt[n]{x_1 \cdot \ldots \cdot x_n}$

   Investiga para qué se emplea esta media.

   (define (geometric-mean l) ...)

5. Implementa la media harmónica:

   $H(x_1, \ldots, x_n) = \frac{n}{\frac{1}{x_1}+\ldots+\frac{1}{x_n}}$

   Investiga para qué se emplea esta media.

   (define (harmonic-mean l) ...)

6. A las tres medias: aritmética, geométrica y harmónica se las denomina *medias pitagóricas*.

   Cambia el nombre de la media a *arithmetic-mean* para que no haya confusión con las otras.

   (define (arithmetic-mean l) ...)

7. Crea una función que calcule la media ponderada, recibiendo una lista de parejas. Cada pareja contiene dos elementos: un *car* con el valor y un *cdr* con un peso.

   $W(x_1, \ldots, x_n) = \frac{x_1 w_1 + \ldots + x_n w_n}{w_1 + \ldots + w_n}$

   (define (weighted-mean l-pairs) ...)

   con l-pairs = *'((1 . 0.5) (4 . 0.8) (3 . 0.4))*

Describe un caso sencillo en el que esto se emplee en la vida real

8. La media cuadrática se describe como $Q(x_1, \ldots, x_n) = \sqrt{\frac{x_1^2+\ldots+x_n^2}{n}}$ impleméntala usando una de las funciones anteriores

   (define (quadratic-mean l) ...)

9. **Extra:** crea una versión de todas las medias para que vayan acumulando su valor y que cada vez que se llama tenga en cuenta los valores anteriores. Esta versión debe incorporar una forma de que se pueda reiniciar, es decir, borrar todos los valores anteriores.

10. Calcula el rango medio de un conjunto de números.

    $M(x_1, \ldots, x_n) = \frac{min+max}{2}$

    (define (mid-range l-pairs) ...)