INF-253 Lenguajes de Programación
Tarea 4: (((((Scheme)))))

Profesor: José Luis Martí – Esteban Daines
Ayudante Cátedras: Juan Pablo Escalona
Ayudantes Tareas: Javier Echiburú – Cristian Vallejos

4 de noviembre de 2016

1. Objetivos

Conocer y aplicar correctamente los conceptos y técnicas del paradigma de programación funcional, utilizando el lenguaje funcional **Scheme**.

2. Reglas

- Se presentarán 10 problemas para resolver en Scheme, cada uno con la función a implementar, su nombre y parámetros respectivos. Esto no restringe que se puedan crear funciones auxiliares. Cada problema debe ser resuelto por separado, en archivos distintos.
- Las soluciones implementadas deben hacer uso de la característica funcional planteada en el enunciado del problema.
- Pueden crear funciones que no estén especificadas para utilizar en los problemas planteados, pero solo se revisara que la función pedida funcione y el problema este resuelto con la característica funcional planteada en el enunciado.
- Para implementar las funciones utilice DrRacket.
    - Descargar DrRacket

3. Problemas

1. **Compresor de listas**
    - **Sinopsis**: `(compresor lista)`
    - **Característica Funcional**: Creación y Recorrido de Listas
    - **Descripción**: Una lista que contiene elementos continuos repetidos puede ser comprimida escribiendo la cantidad de veces que se repite el numero a la derecha de este. Escriba un algoritmo que permita llevar dicha idea a cabo retornando una lista de tuplas, las cuales serán la cantidad de veces que se repite un numero junto con el numero que se repite. En caso de que exista una lista dentro de la lista principal ingresada, se encerraran las tuplas de dicha lista dentro de una nueva lista.

- **Ejemplos**:

```
>(compresor '(4 4 4 3 3 3 3 1 5 5 5))
((3 4) (4 3) (1 1) (3 5))
>(compresor '(1 1 1 9 9 9 9 (9 9 5 5) 5 5 1 1))
((3 1) (4 9) ((2 9) (2 5)) (2 5) (2 1))
```

2. **Cálculo de costo de expresiones**

   - **Sinopsis**: (calculo costs expr)
   - **Característica Funcional**: Listas de Asociación
   - **Descripción**: El valorizador debe calcular el costo de una expresión expr utilizando un conjunto de costos costs. El conjunto de costos es una lista de asociación que contiene los costos de los símbolos de una expresión de la forma (símbolo, costo), si un símbolo no se encuentra en el conjunto, se asume su costo igual a 0.
   - **Ejemplos**:

```
>(calculo (list (cons 'a 20) (cons 'b 50) (cons 'n 10)) '(b a n a n a))
130
>(calculo (list (cons '* 60) (cons '3 30) (cons 'a 10)) '(2 3 b 3 a *))
130
```

3. **Medición de peso de móvil equilibrado**

   - **Sinopsis**: `(pesomovil árbol)`
   - **Característica Funcional**: Recorrido de Árbol
   - **Descripción**: Un árbol binario puede ser representado por una lista mediante:
     (valor_nodo árbol_izquierdo árbol_derecho)
     Por lo tanto, una hoja sería un nodo con dos hijos nulos:
     (valor_nodo () ())
     Un móvil puede ser representado como árbol tomando al valor de nodo como el peso de la barra que sostiene los dos submóviles. La función pesomovil debe revisar si el móvil se encuentra equilibrado en peso (los submóviles de cada nodo deben tener el mismo peso), de estarlo debe devolver el peso correspondiente a todo el móvil, en caso contrario retorna nulo.
   - **Ejemplos**:

```
>(pesomovil '(4 (1 (5 () ()) (1 (2 () ()) (2 () ()))) (3 (4 () ()) (4 () ()))))
26
>(pesomovil '(4 (1 (3 () ()) (1 (2 () ()) (2 () ()))) (3 (4 () ()) (2 () ()))))
()
```

4. **Suma de cubos**

   - **Sinopsis**: (sumar-cubos num-list)
   - **Característica Funcional**: Recursividad de Cola
   - **Descripción**: Dada una lista de números, se debe calcular la suma total de los cubos de cada uno utilizando recursión de cola. En caso que la lista se encuentre vacía, se debe devolver el valor 0.

- **Ejemplos**:

```
>(sumar-cubos '(5 6 4 10 3))
1432
>(sumar-cubos '(9 3 4 2 1))
829
```

5. **La sucesión de Goferk**

   - **Sinopsis**: `(goferkcola n) y (goferkiter n)`
   - **Característica Funcional**: Recursividad de Cola vs Iteración
   - **Descripción**: Se tiene la siguiente sucesión:

     $A_0 = 0$
     $A_1 = 1$
     $A_2 = 3$
     ...
     $A_n = A_{n-1} - A_{n-2} + n$

     Se pide implementar la sucesión anteriormente descrita en recursividad de cola y en iteración.
   - **Ejemplos**:

```
>(goferkcola 7)
7
>(goferkiter 16)
18
```

6. **Teorema del Binomio**

   - **Sinopsis**: (binomcola x y n) y (binomsimple x y n)
   - **Característica Funcional**: Recursividad de cola vs Recursividad simple
   - **Descripción**: Debe retonar el resultado de elevar la suma entre x e y a la potencia n. El coeficiente binomial se calcula como:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

     Luego, para el cálculo del binomio elevado a n se utiliza:

$$(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} x^{n-k} y^k$$

     Se debe presentar 2 funciones, una escrita con recursividad simple y una con recursividad de cola.
   - **Ejemplos**:

```
>(binomcola 3 7 5)
100000
>(binomsimple 2 1 4)
81
```

7. **Función de evaluación impar**

   - **Sinopsis**: `((impar func) valor)`
   - **Característica Funcional**: Expresiones Lambda
   - **Descripción**: Se debe implementar una función que retorne una función que, cada vez que se utilice, sólo funcionará si se le ha llamado un numero impar de veces. Para otro caso, retorna nulo o el numero ingresado.
   - **Ejemplos**:

```
>(define p (impar (lambda (x) (* x x))))
>(p 3)
9
>(p 5)
5
>(p 6)
36
```

8. **Mapeo de función sobre una lista simple**

   - **Sinopsis**: ((mapear func) lista)
   - **Característica Funcional**: Expresiones Lambda
   - **Descripción**: Se debe implementar una función que retorne una función que mapea a la original sobre una lista argumento, obviamente, sin utilizar map.
   - **Ejemplos**:

```
>((mymap not) '(#t #f #t #t))
(#f #t #f #f)
>((mymap (lambda (x) (* x x))) '(1 2 3 4))
(1 4 9 16)
```

9. **Merge de listas**

   - **Sinopsis**: `(merge lista1 lista2)`
   - **Característica Funcional**: Listas Simples
   - **Descripción**: Teniendo dos listas ordenadas ascendentemente (pueden tener distinto tamaño) se pide que se unan en una sola lista, manteniendo el orden ascendente de estas.
   - **Ejemplos**:

```
>(merge '(1 3 5 7 9) '(1 2 6))
(1 1 2 3 5 6 7 9)
>(merge '(11 23 61 72 94) '(4 15 88 89 92 99))
(4 11 15 23 61 72 88 89 92 94 99)
```

10. **Evaluación Lógica**

    - **Sinopsis**: (logiceval expr)
    - **Característica Funcional**: Listas Simples y Operadores
    - **Descripción**: Se debe evaluar una expresión lógica que contiene los átomos #t y #f. La expresión, además, utiliza los operadores Y, O, OEX y NO en lugar de and, or, xor y not.

- **Ejemplos**:

```
>(logiceval '(Y #t (O #t #f))
#t
>(logiceval '(O #f (Y (OEX #t #t) (NO #f))))
#t
```

4. Sobre Entrega

- Cada función que no este definida en el enunciado del problema debe llevar una descripción según lo establecido por el siguiente ejemplo:

```
;;(Nombre_función parámetros)
;;Breve descripción
```

- Se debe trabajar en grupos de dos personas, cualquier cambio de grupo debe ser avisado a mas tardar el día Viernes 28 de Octubre.
- **La entrega debe realizarse en tarball (tar.gz) y debe llevar el nombre: Tarea4LP_RolIntegrante-1_RolIntegrante-2.tar.gz**
- **El archivo README.txt debe contener nombre y rol de los integrantes del grupo e instrucciones para la utilización de su programa.**
- **El no cumplir con las reglas de entrega conllevará un máximo de -30 puntos en su tarea.**
- La entrega será via moodle y el plazo máximo de entrega es hasta el **Miércoles 9 de Noviembre**.
- Las primeras 3 horas de atraso se descontaran 10 puntos por hora, luego por día de atraso se descontaran 30 puntos.
- Las copias serán evaluadas con nota 0 y se informarán a las respectivas autoridades.

5. Calificación

Todos comienzan con nota máxima y se les irá descontando por cada punto omitido (el puntaje que aparece al lado es el máximo de puntos que se les descontara en caso de incumplimiento):

- Código no ordenado (-10 puntos)
- Código no comentado (-5 puntos)
- Problema no resuelto (-10 puntos cada uno)