INF-253 Lenguajes de Programación
Tarea 4: Scheme

Profesor: Esteban Daines - Roberto Diaz
Ayudante Cátedras: Sebastian Godinez
Ayudante Tareas: Gabriel Valenzuela - Monserrat Figueroa

12 de Noviembre de 2018

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 para resolverlo.
- Para implementar las funciones utilice DrRacket, el cual está instalado en los computadores del LDS. El lenguaje Scheme es multiplataforma, por lo que lo pueden trabajar en su ordenador, recordando realizar las pruebas finales en el LDS.
    - http://racket-lang.org/download

3. Problemas

1. Búsqueda de raíces por bisección
    - **Sinopsis:** `((biseccion funcion) intervalo iter)`
    - **Característica Funcional:** Funciones lambda.

- **Descripción:**Un método numérico para buscar raíces de una función de una variable, es a través del método de bisección. Este método basado en el teorema de valor medio determina la raíz de una función dado un intervalo de búsqueda y reduciéndolo a la mitad en cada iteración. En caso que el intervalo sea invalido la función debe retornar `null`.
- **Ejemplos:**

```
>(biseccion ((lambda(x)(+ (* x x) (* 2 x) -3))) '(0 2)
10)
1
```

2. Números Triangulares

- **Sinopsis:** `(triangular_cola numero) (triangular_simple numero)`
- **Característica Funcional:** recursión simple vs de cola.
- **Descripción:** Un número triangular es aquel que puede componer a través de puntos la forma de un triangulo equilátero, comenzando desde el 1. Las funciones deben ser capaces de reconocer si un número es triangular o no, utilizando recursión de cola y recursión simple.
- **Ejemplos:**

```
>(triangular_cola 10)
#t
```

3. Suma de nodos de un árbol

- **Sinopsis:** `(suma_arbol arbol numero)`
- **Característica Funcional:** Búsqueda en árbol binario.
- **Descripción:** Un árbol binario puede ser representado por una lista mediante:

  `(valor_nodo arbol_izquierdo arbol_derecho)`

  Por lo tanto, una hoja sería un nodo con dos hijos nulos:

  `(valor_nodo () ())`

  La función debe buscar un número en un árbol binario y obtener la suma de los nodos recorridos, incluyendo la raíz y el número buscado. En caso que no exista el número en el árbol la función debe retornar `null`.
- **Ejemplos:**

```
>(suma_arbol '(8 (3 (1 () ()) (6 (4 () ()))) (10 () ()))
1)
12
```

4. Replicación de elementos de una lista

- **Sinopsis:** `(replicar lista lista_replicacion)`
- **Característica Funcional:** Recorrido y creación de listas

- **Descripción:** La función recibirá dos listas, la primera tendrá los elementos a ser replicados y la segunda del mismo tamaño contendrá las veces que tiene que ser replicado cada elemento respectivamente. La función debe retornar una lista con los elementos repicados en el orden correspondiente.
- **Ejemplos:**
```
>(replicar '(a b c) '(1 2 3))
'(a b b c c c)
```

5. Serie de potencia para coseno

- **Synopsis:** `(iter_cos valor iter) (rec_cos valor iter)`
- **Característica Funcional:** Recursión vs Iteración.
- **Descripción:** Utilizando el teorema de Maclaurin se puede tener una aproximación de coseno a través de la siguiente serie de potencias

$$\cos(x) = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n}}{(2n)!}$$

  Se debe realizar una función que calcule la aproximación del coseno de un valor dado, utilizando el número de expansión señalado. Se debe implementar cada función utilizando recursión e iteraciones respectivamente.
- **Ejemplos:**
```
>(rec_cos (/ pi 2) 3)
0.01996895999999998
```

6. Cruzamiento

- **Sinopsis:** `(cruzar lista1 lista2 n)`
- **Característica Funcional:** Listas simples.
- **Descripción:** La función debe realizar el cruce entre dos listas del mismo tamaño, es decir, intercambiar el primer trozo de tamaño n de cada lista generando dos listas nuevas.
- **Ejemplos:**
```
>(cruzar '( a b a a b c) '(a c c b c a) 3)
'((a c c a b c) (a b a b c a))
```

7. Operaciones entre binarios

- **Sinopsis:** `((operar func) lista1 lista2)`
- **Característica Funcional:** Expresiones lambda.
- **Descripción:** La función, recibe una función y dos listas binarias del mismo largo. Se debe aplicar la función `func` como operación entre las listas, las cuales solo estarán compuestas de 0s y 1s, y retornar el resultado como una lista binaria. La función `func` recibe 2 parámetros, cada uno correspondiente a un 0 o 1 y retorna 0 o 1.

- **Ejemplos:**
```
>((operar xor) (list 0 1 1 1) (list 0 0 1 0))
(0 1 0 1)
```

8. Componentes de un grafo

   - **Synopsis:** `(componentes grafo)`
   - **Característica Funcional:** Recorrido de grafos.
   - **Descripción:** Un vértice de un grafo se puede expresar como: `(vertice '(vecinos adyacentes))` y un grafo es una lista de vértices. La función debe retornar el número de componentes de un grafo.
   - **Ejemplos:**
```
>(componentes '((1 (2 3)) (2 (1 4)) (3 (1)) (4 (2)) (5
(6)) (6 (5)) (7 ()))
3
```

9. Generar permutaciones

   - **Sinopsis:** `(permutaciones lista)`
   - **Característica Funcional:** Listas simples.
   - **Descripción:** La función debe retornar una lista con todas las permutaciones posibles de la lista pasada como parámetro (incluyéndola). El orden del resultado puede variar dependiendo de la implementación.
   - **Ejemplos:**
```
>(permutaciones '(1 2 3))
'((1 2 3) (1 3 2) (2 1 3) (2 3 1) (3 1 2) (3 2 1))
```

10. 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
```

4. Sobre Entrega

- Cada función debe llevar una descripción según lo establecido por el siguiente ejemplo:

```
#|
Funcion: Suma_Enteros
Descripcion: suma dos enteros positivos
Parametros:
n1 entero
n2 entero
Retorno: resultado de la operacin aritmtica de la suma entero.
|#
```

- Se debe trabajar en grupos de dos personas, no es necesario que sean del mismo curso.
- La entrega debe realizarse en tarball (tar.gz) y debe llevar el nombre: Tarea4LP_RolIntegrante-1_RolIntegrante-2.
- El archivo README.txt debe contener nombre y rol de los integrantes del grupo e instrucciones para la utilización de su programa
- La entrega será mediante moodle y el plazo máximo de entrega es hasta el Miércoles 28 de Noviembre a las 23:55.
- Por cada día o fracción de atraso se descontarán **50 puntos**. Si la fracción es menor a 20 minutos, el descuento será de sólo **5 puntos**.
- Las copias serán evaluadas con nota 0.

5. Evaluación

Puntajes

- Problemas 8 y 9: 14 puntos c/u
- Resto de los problemas: 9 puntos c/u

Descuento

- Comentario faltante (5 Puntos c/u)
- No respetar reglas de entrega (40 puntos c/u)