INF-253 Lenguajes de Programación
Tarea 4: Scheme

Profesor: Roberto Nicolas Diaz Urra
Ayudante Cátedras: Sebastián Godínez
Ayudante Tareas: Monserrat Figueroa - Sebastián Campos

1. Historia

.)).)) ..)).. .) / . ...))))).)) / .) ... ..) ...) .)).. ... ..)))))))..)) / . ...))))).)) / .)) .). .) .)). .))..))) / .). / .).. .) /)). .)) .) /).. . / .)..))) ... / .)). .) .). .).) . ... /).)) /).))) / .)). ..) .)..))) / ... .) .).. .. .). .).).) /).))) /).). .). . .)). / ... ..) ... /)) .).) .. .). .) ...))..)) / .). . .)). ..))))))..)) /).))) /).). .). . .)). / ... ..) ... /)) .).) .. .). .) ...))..)) / . .).. / .)). .) ...) . .).. / . ... / ..)). .) /)) .).) .. .). .) / .). .)).).). .))). ..) .) /).))) / .).) .. ...) . /).)) / ..). . ... /)).)))).. /) .. .). . / ..)). / .)). .). . ..). .. .)))))) / .). / ... ..) / .) .)). . .).. .).. ..)..))) .).).

2. Objetivos

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

3. Reglas

- Se presentarán 10 problemas en Scheme. Cada uno posee una función a implementar, con su nombre y parámetros respectivos. Esto no restringe que se puedan crear funciones auxiliares. Cada problema debe ser resuelto por separado, en **archivos distintos**.
- Pueden crear funciones que no estén especificadas para utilizar en los problemas planteados, pero solo se revisará que la función pedida funcione correctamente y el problema este resuelto con la característica funcional planteada en el enunciado.
- Para implementar las funciones utilice DrRacket: `http://racket-lang.org/download/`

4. Problemas

1. **Mira mamá, nacieron gemelos**

   - **Sinopsis**: `(gemelos arbol1 arbol2)`
   - **Característica Funcional**: Árboles.
   - **Descripción**:A partir de dos árboles binarios, determinar si son isomorfos. 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 () ())`
   - **Ejemplos**:

     ```
     >(gemelos '(2 (12 () () ) (1 () (8 () () ) ) ) '(3 (2 () () ) (18 () (9 () () ) ) ) )
     #t
     >(gemelos '() '(4 (7 () (3 () (1 (4 () () ) () ) ) ) () ) )
     #f
     ```

2. **Hermanito mayor y menor**

   - **Sinopsis**: `(maymecola listadelistas)(maymecolasimple listadelistas)`
   - **Característica Funcional**: Recursividad de cola.
   - **Descripción**: Desarrollar la función *mayme*, la cual recibe una lista de listas. En cada lista, su cabeza corresponde al 'nombre' de la lista y la cola son números enteros. La función debe retornar el 'nombre' de las dos listas que la suma de los elementos de su cola sea el máximo y el mínimo. (No existirán dos listas que sumen lo mismo, ni listas sin nombre, ni listas con nombres iguales).

   - **Ejemplos**:

     ```
     >(maymecola '((1 2 3 4) (432 4 3 2 3) (4 0 1 4) (-1 -1 -1 -1 -2 -3)))
     (432 -1)
     ```

3. **1 vs 1 skrub**

   - **Sinopsis**: `(vs lista)`
   - **Característica Funcional**: Listas Simples y Operadores.
   - **Descripción**: Se le entregará una lista con tres elementos:
     - Una letra A, O o X que representa los operadores lógicos AND, OR y XOR respectivamente.
     - Dos listas del mismo largo llenas de 0's y 1's.

     Las listas entrarán en combate, y el ganador será quien tenga más puntaje al final de la batalla. La batalla se realizará de la siguiente forma:

     - Cada elemento de la lista competirá con el elemento que esté en la misma posición que él en la lista contaria.
     - La batalla entre elementos se realizará utilizando el operador ubicado en el primer elemento de la lista más grande (O, A o X).
     - Cada elemento se comparará con el enemigo utilizando el operador que corresponda.

- Si el resultado de la comparación es `#t` se suma 1 punto a la lista que corresponda, si es `#f` suma 0 puntos a la lista que corresponda.
- El puntaje de las batallas se asignan de la siguiente forma: Las batallas en posiciones pares se le asignan a la lista 1, mientras que el puntaje de las batallas en las posiciones impares se le asignan a la lista 2.

Finalmente, el programa debe retornar la lista ganadora utilizando el número que la identifica (1 para la lista de la izquierda, 2 para la lista de la derecha), en caso de empate retornar `#f`.

Nota: La listas tendrán el mismo largo. El largo es par. Posiciones parten de 1.

- **Ejemplos**:

```
>(vs '(A (0 1 1 0 1 1) (1 0 1 1 0 1)))
#f
```

Nota: Los puntajes de esta batalla son: Izquierda:1 Derecha:1

```
>(vs '(D (1 0 0 0 0 0 1 0) (1 0 0 0 0 1 1 0)))
2
```

Nota: Los puntajes de esta batalla son: Izquierda:1 Derecha:2

4. **Me gusta el orden**

- **Sinopsis**: `(orden n matriz)`
- **Característica Funcional**: Matrices.
- **Descripción**: Se le entregará una matriz de $n \times n$ elementos. Se debe determinar si la matriz tiene una fila, una columna o la diagonal ordenada de menor a mayor. Debe retornar `#t` o `#f`.
  Nota: Menor a mayor se considera de arriba a abajo y de izquierda a derecha.
- **Ejemplos**:

```
>(orden 3 '( (1 4 3) (2 2 12) (1 5 15) ) )
#t
>(orden 3 '( (3 9 8) (6 5 4) (2 1 7) ) )
#t
```

5. **Al infinito y más allá**

- **Sinopsis**: `(voy nodo grafo)`
- **Característica Funcional**: Grafos.
- **Descripción**: Dado un nodo, señalar el camino más largo en cantidad de nodos a cualquier otro nodo. El camino más largo debe tener los nodos que lo componen, si hay dos caminos del mismo largo devolver cualquiera. El grafo es una lista de listas, cada sub-lista tiene en su primera posición el nodo que representa, y la segunda posición tiene la lista de adyacencia del nodo.
  Nota: Los nodos están ordenados, comienzan en el 1 y son consecutivos.

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

6. **Me desarmé**

- **Sinopsis**: `(armar k lista)`
- **Característica Funcional**: Listas simples.

- **Descripción**: Dado un número k, encontrar en la lista un elemento i y un elemento j tal que $i+j=k$. Devolver todos los elementos i y j que cumplan éstas características en una lista de listas. Si no existe retornar `#f`. Cada elemento solo puede aparecer una vez en el resultado
- **Ejemplos**:

```
>(armar 12 (1 2 3 8 23 8 14 4 9))
((8 4) (3 9))
```

7. **Iteración de punto fijo**

   - **Sinopsis**: `(fpi funcion umbral i)`
   - **Característica Funcional**: Funciones lambda.
   - **Descripción**: El punto fijo de una función f es la solución a la ecuación:

     $$f(x) = x$$

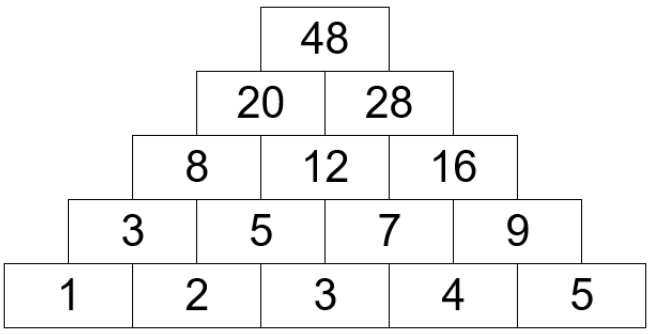
     Para encontrar la solución iterativamente, se comienza con un valor inicial x_0 y este es evaluado en la funcion f, se observa si se cumple la ecuación anterior, en caso que no se cumpla el valor de x_0 es remplazado por $f(x_0)$ y se vuelve a repetir el mismo proceso hasta que x_i y $f(x_i)$ sean iguales.

     La función `fpi` recibe una función que cumple con los criterios de convergencia, un valor inicial y un umbral que representa el valor absoluto de la diferencia admisible entre x_i y $f(x_i)$, es decir, $|f(x_i) - x_i|$. La función debe retornar las iteraciones que se tuvieron que realizar hasta alcanzar dicho umbral.
   - **Ejemplo**:

```
>(fpi (lambda (x) (cos x)) 0.02 1)
8
```

8. **Cima de la piramide**

   - **Sinopsis**: `(cima lista)`
   - **Característica Funcional**: Recorrido de listas.
   - **Descripción**: Teniendo una lista de números, estos se pueden sumar de a pares múltiples veces formando una piramide de la siguiente manera:



   La funcion cima, recibe una lista de números y retorna el número que se encuentra en la cima de la pirámide.

- **Ejemplo**:
  ```
  >(cima '(1 2 3 4 5))
  48
  ```

9. **Segmentación**
   - **Sinopsis**: `(segm funcion lista)`
   - **Característica Funcional**: Listas y operadores lógicos.
   - **Descripción**: La función recibe una función f y una lista. `segm` debe evaluar cada elemento de la lista con la función f. La lista debe quedar ordenada de forma tal que los elementos que la función f retorna verdadero preceden aquellos que retorna falso. No es necesario conservar el orden relativo.
   - **Ejemplos**: `>(segm (lambda (x) (< x 4)) (7 4 3 8 2))`
     `(3 2 7 4 8)`

10. **Soy un sumador!**
    - **Sinopsis**: `(serie funcion entero)`
    - **Característica Funcional**: Recursión y paso de funciones.
    - **Descripción**: Serie recibe una función f y un entero n. Retorna la sumatoria de la aplicación de la función de $i = 1$ hasta n, es decir, $\sum_{i=1}^{n} f(i)$.
    - **Ejemplos**:
      ```
      >(serie (lambda (x) (* x x)) 3)
      14
      ```

5. 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 bien explicada.
  ;;Retorno de la función
  ```

- Se debe trabajar en grupos de dos personas, cualquier cambio de grupo respecto a tareas anteriores debe ser avisado con brevedad.
- Cuidado con el orden y la identación de su tarea, puede llevar descuentos.
- Cada problema debe resolverse en archivos separados, el nombre de cada archivo debe ser 'PX' con X el número del problema resuelto.
- **La entrega debe realizarse en 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 en caso de ser necesarias.**
- **El no cumplir con las reglas de entrega conllevará un descuento máximo de 30 puntos en su tarea.**

- La entrega será vía moodle y el plazo máximo de entrega es hasta el día **10 de enero de 2020 a las 23:55 hrs.**.
- **Serán -10 puntos por cada hora de atraso.**
- Las copias serán evaluadas con nota 0 y se informarán a las respectivas autoridades.

6. Calificación

- Código no ordenado (-20 puntos)
- Código no comentado (-5 puntos)
- Problema resuelto (10 puntos cada uno)
- Reglas de entrega (-30 puntos)