INF-253 Lenguajes de Programación
Tarea 4: Scheme

Profesor: José Luis Martí Lara, Jorge Ariel Díaz Matte,
Wladimir Elías Ormazabal Orellana.
Ayudante Cátedra: Norman Cortés Vega, Lucas Morrison Osorio,
Jhossep Martínez Velásquez.
Ayudante Coordinador: Álvaro Rojas Valenzuela.
Ayudante Tareas: Fernanda Araya Zárate, Bryan González Ramírez,
Andrés Jablonca Peña, Blas Olivares Gutiérrez, Nicolás Paz Tralma,
Bastián Salomon Ávalos, Cristian Tapia Llantén, Cristóbal Tirado Morales.

22 de Octubre de 2024

1. Los Recursive-Scheme

Después de terminar el juego de Game's Java Class, decides viajar a la ciudad de las funciones, ((((Land)))). Al llegar al bar **Los Tragos Lambdas**, un grupo de 4 miembros, conocido como los *Los Recursive-Scheme* te invitan a unirte a su mesa. Luego de unas bebidas, les cuentas las aventuras de las tres ciudades. Sorprendidos, te proponen 4 desafíos cada uno, uno más difícil que el anterior. Si logras superar cada uno, te invitarán algunos tragos extras. Para ello, tienes que crear funciones en el lenguaje Scheme para superar los distintos desafíos.

2. Funciones a Implementar

1. El Extraviado

   - **Sinopsis:** `(buscador lista elemento)`
   - **Característica Funcional:** Funciones simples, listas simples, recursión.
   - **Descripción:** El primer miembro le gusta ordenar sus pertenencias en listas. Se tiene una lista de objetos variados $(a_1, a_2, a_3, \dots)$, estos pueden ser, números, palabras, símbolos o listas. Estas listas terminan siendo tan grande que ya no logra encontrar lo que busca. Por lo que te pide que construyas una función `buscador` que al recibir una lista y un elemento a, encuentre la posición i de ese

elemento, o -1 si no está en la lista.
Nota: El primer miembro no te permite que uses funciones como `member, memw, memv, memq, findf` o `list-ref`, ya que necesita ver la implementación recursiva.
Nota: El elemento a buscar tienen que ser exactamente igual, por ejemplo, 2 y 2.0 son diferentes, como también `'ABC` y `"ABC"`, se asegura que todos los elementos de la lista son únicos.

- **Ejemplos:**

```
> (buscador '(1 2 3) 3)
3

> (buscador '(ABC "ABC" 3.0 1234) "ABC")
2

> (buscador '(ABC "ABC" 3.0 1234) 'ABC)
1

> (buscador '(389 (2 4 5.0) (40 here 2)) '(40 here 2))
3

> (buscador '() 'INF253)
-1
```

2. Serie de Taylor del Seno y Coseno

   - **Sinopsis:** `(taylorSenoSimple n x)` y `(taylorCosenoCola n x)`
   - **Característica Funcional:** Recursión simple, recursión cola.
   - **Descripción:** El segundo miembro le gustan mucho las matemáticas, y mucho más los métodos numéricos. Por lo que te presenta las Series de Taylor para el `Seno` y `Coseno`. Estas series son muy útiles para aproximar funciones en una serie de potencias, siendo más fáciles de trabajar computacionalmente. Para ello tienes que construir las funciones `taylorSenoSimple` que calcula la serie del seno con n términos, evaluado en x, **usando recursión simple** y `taylorCosenoCola` que calcula la serie del coseno con n términos, evaluado en x, **usando recursión de cola**. Utilice las siguientes ecuaciones:

$$\sin_n x = \sum_{i=0}^{n} \frac{(-1)^i}{(2i+1)!} x^{2i+1}$$

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

- **Ejemplos:** `> (taylorSenoSimple 300 3.14)`
  `0.0015926529164871975`

  `> (taylorCosenoCola 300 3.14)`
  `-0.9999987317275395`

  `> (taylorSenoSimple 1 2.14)`
  `0.5066093333333335`

  `> (taylorCosenoCola 1 2.14)`
  `-1.2898`

  `> (taylorSenoSimple 0 20.3)`
  `20.3`

  `> (taylorCosenoCola 0 20.3)`
  `1`

3. Composición Rotacional

   - **Sinopsis:** `(evaluador funciones numeros)`
   - **Característica Funcional:** Funciones lambda, recursión simple, recursión cola.
   - **Descripción:** Una operación de rotación a una lista $a_1, a_2, \ldots, a_{n-1}, a_n$ extrae el primer elemento y lo agrega al final de la lista, formando $a_2, \ldots, a_{n-1}, a_n, a_1$. El tercer miembro le gusta jugar con las composiciones de funciones, se tiene una lista de funciones lambda $f_1, f_2, \ldots, f_n$, y una lista de números $a_1, a_2, \ldots, a_n$. Para todos los números a_i, realizar $i-1$ rotaciones a la lista de funciones y evaluar $f_{i-1}(f_{i-2}(\ldots f_1(f_n(\ldots f_{i+1}(f_i(a_i))\ldots)$. Realizar las n evaluaciones es muy tedioso realizarlo a mano. Por lo que te pide que construyas las funciones `(evaluador funciones numeros)` usando alguna de las dos recursiones, que reciben una lista de funciones lambda y una lista de números, y que retorne una lista de todas las evaluaciones rotacionales de cada número. Es decir, retorne la lista:

$$
\begin{aligned}
&(f_n(f_{n-1}(\ldots f_2(f_1(a_1))\ldots),\\
&\ f_1(f_n(\ldots f_4(f_3(f_2(a_2))\ldots),\\
&\ f_2(f_1(f_n(\ldots f_4(f_3(a_3))\ldots),\\
&\ \ldots\\
&\ f_{n-1}(f_{n-2}(\ldots f_1(f_n(a_n))\ldots))
\end{aligned}
$$

Nota: No se permite el uso de `list-ref`. Se asegura que no se producirán indeterminaciones al momento de evaluar.

- **Ejemplos:**

```
> (evaluador (list (lambda (x) (+ x 1)) (lambda (x) (* x x)) (lambda
(x) (- x 2))) '(2 5 7))
(7 24 36)
```

Explicación: Se tiene las funciones: $f_1(x) = x + 1$, $f_2(x) = x \cdot x$, $f_3(x) = x - 2$. Las listas (f_1, f_2, f_3), (f_2, f_3, f_1) y (f_3, f_1, f_2) son el resultado de aplicar $0, 1$ y 2 rotaciones respectivamente. El resultado de evaluar en la lista $(2, 5, 7)$ es:

$$f_3(f_2(f_1(2))) = 7$$
$$f_1(f_3(f_2(5))) = 24$$
$$f_2(f_1(f_3(7))) = 36$$

```
> (evaluador (list (lambda (x) (/ x 3.2)) (lambda (x) (+ (* x 2) x))
(lambda (x) (- x (* 5.40 (* x x))))) '(5 2 -7))
(-113.96484375000001 -58.875 -254.625)

> (evaluador '() '())
()
```

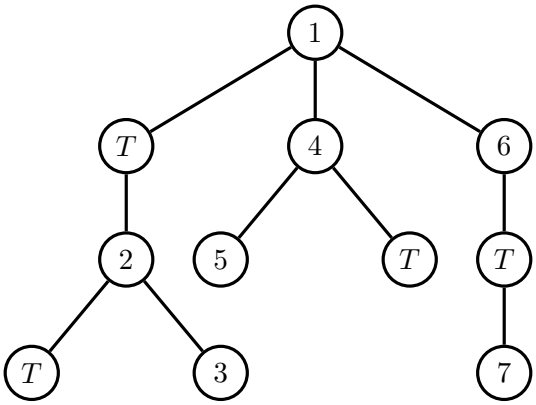
4. Las Profundidades de los Tesoros.

- **Sinopsis:** `(profundidades arbol)`
- **Característica Funcional:** Funciones simples, listas anidadas (estructura de árbol), recursión simple
- **Descripción:** Un árbol puede ser presentado como una lista de esta forma: `(valor_nodo sub_árbol_1 sub_árbol_2 ...)`. Por lo que una hoja será la lista con solo su valor del nodo. Se le considera la raíz del árbol, el nodo que no tiene un nodo padre.
  El cuarto miembro es una cazadora de tesoros y exploradora de profundidades. Para la siguiente aventura, va a tener un mapa de forma de lista representando un árbol. Los nodos del árbol están enumerados desde el 1, y algunos van a tener el valor T, indicando que en ese nodo hay un tesoro. La exploradora requiere que construyas una función llamada `profundidades`, que recibe un árbol, y retorna una lista de las profundidades de los tesoros ordenada de menor a mayor.
  Nota: Sea una raíz s y un nodo u en un árbol, la profundidad del nodo u corresponde a la distancia del nodo s a u.

- **Ejemplos:**

```
> (profundidades '(1 (T (2 (T) (3))) (4 (5) (6)) (7 (T (8)))))
(1 2 2 3)
```

Explicación: El árbol se puede visualizar como:



Vemos que las profundidades de los tesoros T son $1, 2, 2$ y 3. Corresponden a las distancias cada T al nodo 1.

```
> (profundidades '(1 (6 (3) (2 (5))) (4 (7 (8) (9)))))
()
> (profundidades '())
()
```

3. Sobre la Entrega

- El código debe venir indentado y ordenado. De no existir orden, se realizará descuentos.
- Se debe entregar un archivo por cada ejercicio con la(s) función(es) solicitadas, con el siguiente formato de nombres:
  - `P1.rkt, P2.rkt, P3.rkt, P4.rkt`
  - README.txt
- Todas las funciones definidas por `define`, deben ir comentadas con el siguiente formato:

```
;; Descripción de la función
```

```
;;
;; a : Descripción del parámetro a
;; b : Descripción del parámetro b
```

- Se debe programar siguiendo el paradigma de la programación funcional, no realicen códigos que siguen el paradigma imperativo. Por ejemplo, se prohíbe el uso del for-each. El uso de funciones no permitidas implican un 0 en la pregunta.
  Para implementar las funciones, utilice DrRacket.
  - https://racket-lang.org/download/
- Todo código debe contener al principio #lang scheme
- El trabajo es individual obligatoriamente.
- **La entrega debe realizarse en un archivo comprimido en tar.gz y debe llevar el nombre: Tarea4LP_RolAlumno.tar.gz**.
  Ej.: `Tarea4LP_202273000-k.tar.gz`.
- El archivo **README.txt** debe contener **nombre y rol del alumno**, junto a instrucciones detalladas para la correcta **ejecución** del programa.
- La entrega es vía aula y el plazo máximo de entrega es hasta el **04 de noviembre a las 23:55 hora aula**.
- Las consultas se deben realizar mediante el foro de la tarea disponible en AULA.
- Por cada día (o fracción) de atraso se descontará 20 puntos. 10 puntos dentro de la primera hora.
- Las copias serán evaluadas con nota 0 y se informará a las respectivas autoridades.

4. Clasificación

4.1. Entrega

Para la clasificación de su tarea, se debe realizar una entre con requerimientos mínimos que otorgarán hasta 30 pts base. Luego se le entregará puntaje dependiendo de los otros requerimientos que llegue a cumplir.

4.1.1. Entrega Mínima

Para obtener el puntaje mínimo en la entrega, el programa de cumplir con los siguientes requerimientos:

- Implementa **El Extraviado** correctamente. (15 pts)
- Implementa ambas funciones de **Serie de Taylor del Seno y Coseno** correctamente. (15 pts)

NOTA: Las funciones deben entregar respuestas correctas en el formato denotado en sus ejemplos

4.1.2. Entrega

Luego de cumplir con la Entrega Mínima, puede obtener más puntaje cumpliendo con los siguientes puntos (puede haber puntaje parcial por cada punto):

- Implementación de funciones para la correcta resolución de los problemas (Total 70 pts)
  - Implementa **Composición Rotacional** usando recursión simple y/o recursión de cola, y utiliza las funciones lambda correctamente. (30 pts)
  - Implementa **Las Profundidades de los Tesoros** correctamente. (40 pts)

Para todas las funciones, existe puntaje parcial acorde a los casos de pruebas que resuelve.

4.2. Descuentos

- Falta de orden (Max -20 pts)
- Falta de comentarios (-10 pts c/u, Max -30 pts)
- Falta de README (-20 pts)
- Falta de alguna información obligatoria en el README (-5 pts c/u)
- Día de atraso (-20 pts por día (o fracción), -10 pts dentro de la primera hora)
- Mal nombre en algún archivo entregado o información errónea (-5 pts c/u)

En caso de existir nota negativa, esta será remplazada por un 0.