INF-253 Lenguajes de Programación
Tarea 4: Scheme

Profesor: José Luis Martí, Jorge Díaz Matte, Rodrigo Salas Fuentes
Ayudante Cátedras: Gonzalo Severín Muñoz, Jhossep Martínez Velásquez
Ayudante Coordinador: Álvaro Rojas Valenzuela
Ayudante Tareas: Ricardo Barriga Vera, Bryan González Ramírez, Bastián Salomón Ávalos, Cristian Tapia Llantén, Cristóbal Tirado Morales

28 de mayo de 2024

How about a nice cup of Liber-tea!

1. SchemeDivers

Estás aburrido de programar en los mismos lenguajes de programación? Crees que en Java tienen muchas clases? Pues ya no más! Bienvenido estudiante de Informática USM al equipo de logística e información de los SchemeDivers. Aquí tu importante misión es crear funciones en el lenguaje Scheme para apoyar a nuestros defensores de la Supertierra y repartir democracia administrada por la galaxia.

2. Funciones a Implementar

1. **Liberación Prioritaria**

   - **Sinopsis:** `(liberar lista)`
   - **Característica Funcional:** Funciones simples, listas simples, recursión simple.
   - **Descripción:** Un SchemeDiver siempre va en busca del planeta donde más lo necesitan. Tu misión es obtener el nombre del planeta con el menor porcentaje de liberación, para ello te entregan una lista de pares que contiene el nombre y el porcentaje de liberación respectivamente.
     Nota: Si dos planetas son el menor porcentaje de liberación, muestra cualquiera de ellos.

- **Ejemplos:**

```
> (liberar '(("Malevelon Creek" 0.0) ("Rend" 43.0154) ("Caladan" 100.0)
("Trantor" 68.59)))
("Malevelon Creek")

> (liberar '())
()
```

2. **Número de Catalán**

- **Sinopsis:** `(catalan_simple n)` y `(catalan_cola n)`
- **Característica Funcional:** Recursión simple, recursión cola.
- **Descripción:** Los números de Catalán tiene una gran importancia en los problemas de combinatorias. Por ejemplo, dice cuantas son las formas de ordenar n pares de paréntesis bien balanceados. Tu misión es calcular el n-ésimo número de Catalán usando la siguiente fórmula recursiva:

$$C_n = \begin{cases} 1, & n = 0 \\ \frac{2(2n-1)}{n+1} C_{n-1}, & n > 0 \end{cases}$$

  Se debe implementar dos funciones que realicen la operación de esa fórmula, donde `(catalan_simple n)` debe realizar recursión simple y `(catalan_cola n)` debe realizar recursión de cola.

- **Ejemplos:**

```
> (catalan_simple 0)
1

> (catalan_cola 0)
1

> (catalan_simple 5)
42

> (catalan_cola 6)
132
```

3. **Descifrar clave secreta**

   - **Sinopsis:** `(descifrar mensajes lista)`
   - **Característica Funcional:** Funciones lambda, recursión simple, recursión cola.
   - **Descripción:** Se han captado n mensajes encriptados de los autómatas de diferentes tamaños enumerados desde 1 hasta n. Se cree que es una clave para alguna super-arma. Después de una ardua investigación, se identifica que al guardar los mensajes en una matriz, cada fila puede pasar por un proceso de descifrado diferente y obtener un número entero entre 0 y 9 incluidos.
     El descifrado de una fila i de tamaño m se realiza de la siguiente manera. Dado una lista de n funciones f enumerados desde 1 hasta n, el dígito d_i se calcula como:

     $$d_i = \left\lfloor \frac{1}{m} \sum_{j=1}^{m} f_i(a_{ij}) \right\rfloor \mod 10$$

     Por ende la clave es la lista $d_1, d_2, \ldots, d_n$. Tu misión es programar dos funciones que puedan descifrar la clave de la super-arma. La primera es `(descifrar_simple mensajes lista)` en la que debe usar recursión simple. Y `(descifrar_cola mensajes lista)` en la que debe usar recursión de cola.

     Nota: Se asegura que no se van a provocar divisiones por 0. Los mensajes se entregan en formato de lista de listas.
   - **Ejemplos:**

```
> (descifrar_simple '((5 2 3) (3 2 4 2 25)) (list (lambda (x) (* x 5))
(lambda (x) (/ (* x 2) (+ x 2)))))
(6 1)

> (descifrar_cola '((5 2 3) (3 2 4 2 25)) (list (lambda (x) (* x 5))
(lambda (x) (/ (* x 2) (+ x 2)))))
(6 1)

> (descifrar_simple '() '())
()

> (descifrar_cola '() '())
()
```

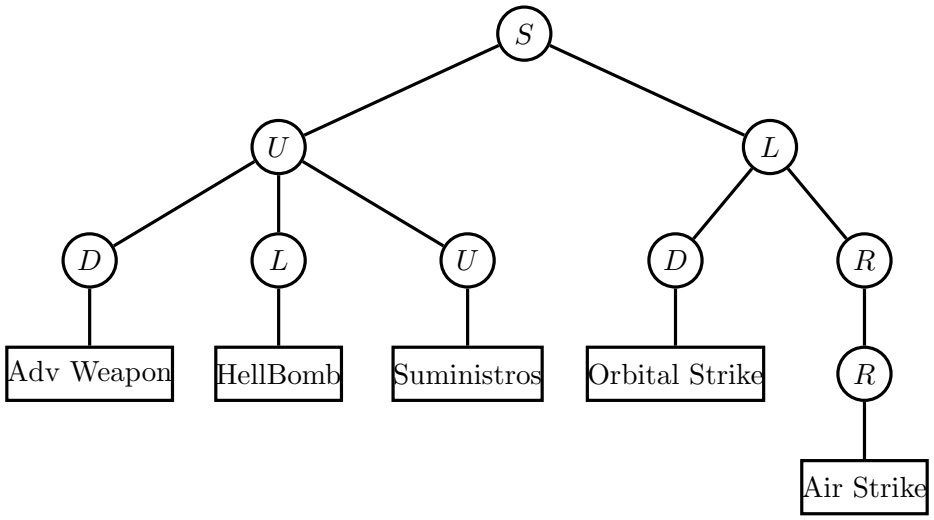
4. **StrataTree**

   - **Sinopsis:** `(stratatree secuencia arbol)`
   - **Característica Funcional:** Funciones simples, listas anidadas (estructura de árbol), recursión simple
   - **Descripción:** Un árbol puede ser representado por una lista mediante: `(Valor_nodo árbol_1 árbol_2 ...)`
     Este árbol representa todos los caminos válidos de una secuencia de `U` (Up), `D` (Down), `L` (Left) y `R` (Rigth). Esta secuencia lleva al nombre de un StrataTree. Tu misión es dada una `secuencia` y un `arbol`, obtener el StataTree correspondiente a esa secuencia, si no existe, debe entregar *false*.
     Nota: todas las secuencias válidas siempre comienzan con `S` (Start). Se asegura que la secuencia es única y siempre tiene asociado a un solo StrataTree. La secuencia y/o el árbol de entrada pueden ser vacíos.
   - **Ejemplos:**

     ```
     > (stratatree '(S U D) '(S (U (D ("Adv Weapon")) (L ("HellBomb")) (U
     ("Suministros"))) (L (D ("Orbital Strike")) (R (R ("Air Strike")))))
     ("Adv Weapon")
     ```

     Explicación:
     De esta llamada a la función, el árbol es `(S (U (D ("Adv Weapon")) (L ("HellBomb")) (U ("Suministros"))) (L (D ("Orbital Strike")) (R (R ("Air Strike"))))`

     Este árbol se puede visualizar como:



Por ende, las secuencias válidas de este ejemplo son: `(S U D)`, `(S U L)`, `(S U U)`, `(S L D)` y `(S L R R)`, y el StrataTree es la hoja que lleva la secuencia. Cualquier otra secuencia debe entregar `false`.
Entonces la secuencia entregada `(S U D)` es una secuencia válida y su StrataTree es `("Adv Weapon")`.

```
> (stratatree '(S L R R) '(S (U (D ("Adv Weapon")) (L ("HellBomb"))
(U ("Suministros"))) (L (D ("Orbital Strike")) (R (R ("Air Strike"))))))
("Air Strike")

> (stratatree '(S L R U) '(S (U (D ("Adv Weapon")) (L ("HellBomb"))
(U ("Suministros"))) (L (D ("Orbital Strike")) (R (R ("Air Strike"))))))
false

> (stratatree '(S U) '(S (U (D ("Adv Weapon")) (L ("HellBomb"))
(U ("Suministros"))) (L (D ("Orbital Strike")) (R (R ("Air Strike"))))))
false
```

3. Sobre la Entrega

- Se deberá 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`
- 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.
  Para implementar las funciones, utilice DrRacket.
  - https://racket-lang.org/download/
- Todo código debe contener al principio #lang scheme
- Puede crear funciones que no estén especificadas para utilizar en los problemas planteados, pero solo se revisará que la función pedida funcione y el problema esté resuelto con las características funcionales planteadas en el enunciado.
- Cuidado con el orden y la indentación de su tarea, llevará descuento de los más 20 puntos.
- Las funciones implementadas y que no estén en el enunciado debe ser comentadas de la siguiente forma:

```
;; Descripción de la función
;;
;; a : Descripción del parámetro a
;; b : Descripción del parámetro b
```

Se harán descuentos por función no comentada.

- Se debe trabajar de forma individual obligatoriamente.
- **La entrega debe realizarse en un archivo comprimido en tar.gz y debe llevar el nombre: Tarea4LP_RolAlumno.tar.gz**.
- **El archivo README.txt debe contener nombre y rol del alumno, e instrucciones detalladas para la correcta utilización**.
- La entrega será vía aula y el plazo máximo de entrega es hasta el **13 de junio a las 23:55 hora aula**.
- Por cada día de atraso se descontarán 20 puntos (10 puntos dentro la primera hora).
- Las copias serán evaluadas con nota 0 y se informarán a las respectivas autoridades.

4. Clasificación

4.1. Entrega

Para la calificación de su tarea, debe realizar una entrega con requerimientos mínimos que otorgarán 30 pts base, luego se entregará puntaje dependiendo de los otros requerimientos que llegue a cumplir.

4.1.1. Entrega Mínima

- Implementa **Liberación Prioritaria**. (15 pts)
- Implementa ambas funciones de **Número de Catalán**. (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:

- Implementación de funciones para la correcta resolución de los problemas (Total 70 pts)
    - Implementa **Decifrar Clave Secreta** de forma recursiva simple y utiliza las funciones lambda. (15 pts)
    - Implementa **Decifrar Clave Secreta** de forma recursiva de cola y utiliza las funciones lambda. (20 pts)
    - Implementa **StrataTree** correctamente. (35 pts)

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

4.2. Descuentos

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