Ejercicios Scheme

Para cada función escriba su contrato, descripción, ejemplos y los tests necesarios, antes de su implementación.

1 Para soltar la mano

1. Represente la siguiente expresión en Scheme: $\frac{5+4+(2-(3-(6+\frac{4}{5})))}{3(6-2)(2-7)}$
2. Defina una función que tome como argumento tres números y retorne la suma de los cuadrados de los dos números mayores
3. El siguiente programa pretende sumar el valor de a, con el valor absoluto de b.¿Es correcto el programa? Argumente.

```
(define (a-suma-abs-b a b)
  ((if (> b 0) + -) a b))
```

4. Implemente la función factorial que calcula el factorial de un entero no negativo n.
5. Implemente la función fibs que calcula el k-ésimo número de Fibonacci.
6. Implemente la función my-expt que recibe dos enteros como argumento, el segundo siempre no negativo, y calcula el primero a la potencia del segundo.[1]
7. Implemente el algoritmo de Euclides para obtener el máximo común divisor entre dos enteros positivos
8. Implemente una función que determina si un entero positivo dado es o no primo. Retorne #t en el caso afirmativo, y #f en otro caso.[2]
9. Implemente la función suma-cuadrados que calcule la suma del cuadrado de los primeros k naturales

[1] Scheme trae incorporada la función *expt*

[2] Una función o expresión que retorna #t o #f es llamada *predicado*. Cualquier valor distinto de #f es considerado como verdadero. Una convención sintáctica de Scheme es finalizar el nombre de un predicado con el símbolo ?. Por ejemplo *primo?*, *null?*, *empty?*

10. Implemente la función suma-cos que calcule la suma del coseno de los primeros k naturales

11. Implemente la función suma-abs que sea una abstracción del proceso de sumar, y reimplemente suma-cuadrados y suma-cos

12. Refine la abstracción de suma-abs para que permita calcular una suma sobre un intervalo arbitrario de enteros positivos, luego redefina suma-abs, suma-cuadrados y suma-cos utilizando esta nueva abstracción

13. Refine la abstracción del ejercicio 12, de manera de que el incremento no sea siempre unitario. Por ejemplo, para sumar sobre todos los números pares , hasta cierta cota superior, se iniciaría el intervalo en 2 y el incremento sería 2

14. Implemente la función Celcius->Farenheit que convierte de grados Celsius a Farenheit

15. Implemente la función Dolar->Peso que convierte de dólares a pesos.

16. Implemente la función Euro->Peso un conversor de euros a pesos.

17. Implemente una abstracción para la conversión de unidades y reimplemente las funciones de los ejercicios 14, 15 y 16 utilizando dicha abstracción.

2 Listas

1. Implemente la función my-list-ref que retorne el k-ésimo elemento de una lista. ¿Qué debería pasar si se pide un elemento mayor al largo de la lista?[3]

2. Implemente la función *lista-fibs* que retorne la lista de los primeros k números de Fibonacci.

3. Implemente la función my-reverse que dada una lista, construya una con los elementos en orden inverso[4]

4. Implemente la función my-take que dada una lista y un entero n, retorne una lista con los primeros n elementos de la lista original[5]

5. Implemente la función my-drop que dada una lista y un entero n, retorne una lista con todos los elementos, excepto los primeros n de la lista original[6]

[3] Corresponde a la función *list-ref* de Scheme
[4] Corresponde a la función *reverse* de Scheme
[5] Corresponde a la función *take* de Scheme
[6] Corresponde a la función *drop* de Scheme

6. Implemente la función rango que dados dos enteros, retorne la sublista que contiene los elementos correspondientes al intervalo dado por estos enteros (Hint: puede usar my-take y my-drop para esto)

7. Implemente la función my-append, que dadas dos listas, retorne una lista con la concatenación de los elementos de estas listas[7]

8. Implemente la función list-sub que, dada una lista de números, retorne la resta sucesiva de sus elementos. Ej.: (list-sub '(1 2 3 4)) debe evaluarse como -8

9. Implemente la función list-add que, dada una lista de números, retorne la suma sucesiva de sus elementos. Ej.: (list-add '(1 2 3 4)) debe evaluarse como 10

10. Implemente la función list-mult que, dada una lista de números, retorne el producto de sus elementos.

11. Implemente la función my-fold que sea una abstracción de list-sub, list-add y list-mult, y reimplemente estas funciones utilizando my-fold

12. Implemente la función list-square que, dada una lista de números, retorne una lista con los cuadrados de cada elemento. Ej.: (list-square '(1 2 3 4) debe evaluarse en (1 4 9 16)

13. Implemente la función list-halve que, dada una lista de números, retorne una lista con la mitad de cada element. Ej.: (list-halve '(1.0 2.0 3.0 4.0)) debe evaluarse en (0.5 1.0 1.5 2.0) ¿Qué resultado obtiene si el parámetro es (1 2 3 4) en vez de (1.0 2.0 3.0 4.0)? ¿Puede explicar esto?

14. Implemente la función my-map, que sea una abstracción de list-square y list-halve. Luego reimplemente estas funciones utilizando my-map.[8]

3 Funciones de Orden Superior

1. Implemente la función `(define (detect list predicate)` que retorna el primer elemento de la lista que cumple el predicado, o falso en otro caso. No debe recorrer toda la lista.

2. Implemente la función `(define (filter list predicate)` que retorna la sublista de elementos que cumple con el predicado, o la lista vacía en otro caso.

3. Implemente la función `(define (reject list predicate)` que retorna la lista de elementos que **no** cumplen el predicado. Es el opuesto de `filter`.

[7] Corresponde a la función *append* de Scheme
[8] Corresponde a la función *map* de Scheme

4. Implemente la función `(define (all list predicate))` que retorna verdadero si todos los elementos de la lista cumplen el predicado, y falso en otro caso.

5. Implemente la función `(define (memoize f))` que retorna una nueva función con el mismo comportamiento que `f` pero con un caché para los resultados.

6. Implemente la función `(define (once f))` que retorna una nueva función que sólo puede ser llamada una vez. Invocaciones repetidas a la función modificada no tendrán efecto y simplemente retornarán el valor de la llamada original.

4 BNF

1. Una lista de elementos arbitrarios se define con la siguiente gramática BNF: list-of-any ::= () | (<any>, list-of-any>. Implemente usando representación de listas y representación define-type la siguiente interfaz para manipulación de listas:

   (a) Crear una lista vacía.

   (b) Crear una lista indicando el primer elemento y el resto de la lista.

   (c) Obtener el primer elemento de la lista.

   (d) Obtener la lista resultante de remover el primer elemento de la lista.

   (e) Retornar $\#t$ si la lista sólo contiene números, y $\#f$ en otro caso.

   (f) Retornar $\#t$ si la lista sólo contiene símbolos, y $\#f$ en otro caso.

   (g) Retornar $\#t$ si cada elemento de la lista cumple con un predicado dado, y $\#f$ en otro caso. Reescriba las funciones de los dos puntos anteriores usando esta función.

2. Un árbol binario con valores numéricos puede representarse usando la siguiente gramática BNF: BinTree ::= <number> |<number>, BinTree, BinTree. Implemente usando representación de listas y representación define-type las siguientes funciones para manipular árboles binarios.

   (a) (root-value t): retorna el valor de la raíz del árbol *t.*

   (b) (rightmost-value t): retorna el valor del nodo que está más a la derecha en el árbol.

   (c) (leftmost-value t): retorna el valor del nodo que está más a la izquierda en el árbol.

   (d) (contains? t v) : retorna #t si algún nodo del árbol t contiene el valor v, y #f en otro caso.

3. Se le ha encargado un software para el monitoreo del sistema de trenes[9]. Un tren consiste en una locomotora en un extremo, seguida por al menos un coche (que no sea locomotora) y finalmente una locomotora en el otro extremo del tren. Existen diversos tipos de coches: comedor, equipaje y dormitorio (también se considera la locomotora como un tipo de coche). Para cada coche se debe conocer su capacidad total de pasajeros y su capacidad utilizada. Los coches comedor poseen además la cantidad de menus disponibles. Los coches dormitorio poseen la cantidad de camas disponibles. Los coches de equipaje poseen el peso total utilizado por el equipaje, y la locomotora el peso total del carbón utilizado.

   Defina una sintáxis BNF e implemente utilizando define-type un tipo de dato abstracto para la representación de trenes. ***Estas definiciones deben permitir sólo configuraciones válidas de trenes***. Luego implemente las siguientes funciones:

   (a) Capacidad total de pasajeros.

   (b) Peso total del carbón.

   (c) Una función para saber cuántas personas quedarán sin cama para dormir la noche siguiente, asumiendo que en cada cama cabe sólo una persona.

   (d) Una función que determine la carga total promedio del tren, considerando como parámetro el peso promedio de un pasajero.

[9] Ej. 3 de Tarea 1 Primavera 2010