Taller 2: Procesamiento de datos complejos, listas arbitrariamente largas y recursión. Fundamentos de Programación

Carlos Andres Delgado S, Msc
carlos.andres.delgado@correounivalle.edu.co

Septiembre de 2020

Reglas del taller

Importante: El no cumplimiento de alguna de las normas aquí expuestas le traerá reducción en la nota o la anulación de su taller.

1. El taller debe ser entregado antes del día **Miércoles, 16 de Septiembre a las 23:59:59** hora de Colombia del por el enlace dispuesto en el campus virtual. Se permiten entregas tardías, pero se descuenta 0.15 en la nota por hora o fracción de retraso. Por ejemplo, si entrega a partir delas 12:00:01 am se aplicará una penalización de 0.15, si lo entrega a partir de las 01:00:01 se aplicará 0.3 y así sucesivamente.

2. Entregue un sólo archivo comprimido. No entregue archivos comprimidos dentro de archivos comprimidos.

3. Debe entregar el código fuente organizado en carpetas dentro del primer nivel del archivo comprimido, no cree una jerarquía compleja difícil de revisar.

4. No se permite copiar código de Internet ni de sus compañeros. Si se encuentra código copiado el taller será anulado por completo a todas las partes involucradas.

5. Debe implementar el taller en Dr Racket.

6. El taller puede ser realizado por grupos de hasta de 3 personas. En los contratos debe incluir los nombres y códigos de todos los integrantes. Si olvida incluir alguno, no aceptarán reclamaciones después.

7. Todas las funciones principales deben llevar todos los elementos de la metodología vista el clase. Las funciones auxiliares únicamente el contrato.

```
;Autores: Juanito Perez, 1902321. Pepita Gomez, 1954545, Juanita Delgado, 1914547
;Fecha: 31-Diciembre-2020
;Contrato: operacionTres: numero,numero,numero->numero
;Propósito: Esta función calcula la operación (a+(b*c))
;Ejemplos
;(operacionTres 1 2 3) 7
;(operacionTres 2 3 4) 14
;Codigo
(define (operacionTres a b c)
    (+ a (auxiliar b c))
)
;Pruebas
(check-expect (operacionTres 1 2 3) 7)
(check-expect (operacionTres 2 3 4) 14)
;Recuerde usar (check-within (funcion 1 2 3) 3 0.1) en casos de irracionales

;Contrato: auxiliar: numero,numero -> numero
(define (auxiliar x y)
     (* x y)
)
```

8. Utilice los nombres de funciones especificados en el enunciado e incluya al menos 8 ejemplos en cada ejercicio que sean diferentes a los especificados por el docente.

1. Listas y recursión [60 puntos]

Una tienda de barrio mantiene su inventario con una lista de productos, un producto es una estructura con cuatro campos:

1. Un símbolo que representa el código del producto
2. Un símbolo que representa el nombre del producto
3. Una estructura llamada disponibilidad, la cual contiene
   a) Un número que indica el precio
   b) Un número que indica cuantas unidades del producto se encuentran disponibles

1. [**20 puntos**] Realice la función consultar, la cual recibe una lista de productos y el código de un producto (un símbolo) y retorna una estructura (que usted establece) con el nombre, el precio y la cantidad del producto correspondiente. Se solicita usar la metodología vista en el curso y dejar al menos 3 ejemplos en el código para la respectiva revisión.

2. [**20 puntos**] Realice la función cargarProductos este recibe una lista de cargar productos que viene desde un archivo de texto. Esta función retorna una lista de productos.

   Para este punto observe el archivo ListaProductos.txt, ejecute en Dr Racket lo siguiente:

```
(define lista1 (read-words/line "data/ListaProductos.txt"))
```

   El archivo de texto debe estar almacenado dentro un directorio llamado data a partir de la ubicación del archivo .rkt, a esto se le conoce como ruta relativa, en las cuales se considera como punto de partida la ubicación del archivo a ejecutar.

Para este punto investigue el paquete de enseñanza batch-io `https://docs.racket-lang.org/teachpack/2htdpbatch-io.html`.

Se solicita usar la metodología vista en el curso y dejar al menos 3 ejemplos en el código para la respectiva revisión.

3. **[20 puntos]** Realice la función venderProducto que recibe como parámetros una lista de productos y un producto, y retorna una lista de productos actualizada vendiendo el producto (en este caso el numero de existencias será la cantidad de productos a vender). Si la cantidad de un producto llega a 0 se debe borrar de la lista. **Posteriormente, la haga una función auxiliar que almacene la respuesta de esta función en un archivo de texto.**

   Se solicita usar la metodología vista en el curso y dejar al menos 3 ejemplos en el código para la respectiva revisión.

2. Datos complejos: árboles [40 puntos]

Un árbol es una estructura recursiva, que se implementa como estructuras dentro de estructuras, por ejemplo para el siguiente árbol:

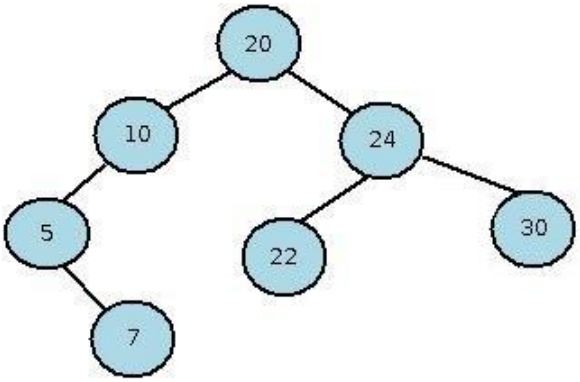

Figura 1: Ejemplo árbol

Los nodos se pueden implementar de esta forma:

```
(define-struct arbol (valor hijo-izq hijo-der))
```

El árbol de la figura puede ser implementado así:

```
(make-arbol 20
        (make-arbol 10
                (make-arbol 5
                        empty
                        (make-arbol 7 empty empty)
                )
                empty
        )
        (make-arbol 24
                (make-arbol 22 empty empty)
                (make-arbol 30 empty empty)
        )
)
```

Realice las siguientes funciones:

4. [**20 puntos**] Una función que reciba dos árboles binarios, retorna verdadero si ambos son iguales, en caso contrario falso. Se solicita usar la metodología vista en el curso y dejar al menos 3 ejemplos en el código para la respectiva revisión.

5. [**20 puntos**] Una función que reciba un árbol binario de números y una lista de números, la función debe determinar si todos los elementos de la lista están en el árbol ,en tal caso, la función debe devolver verdadero, de lo contrario debe devolver falso. Se solicita usar la metodología vista en el curso y dejar al menos 3 ejemplos en el código para la respectiva revisión.