TC2006 – Lenguajes de Programación

Programación Básica en Racket

Para realizarse en equipos de 2 integrantes (3 estrellas)

I. Programa las siguientes funciones **SIN** utilizar listas.

1. Programar la función **mayor** que reciba 4 argumentos numéricos y regrese el mayor.

   Probar con:

   ```
   > (mayor 7 5 3 6)          => 7
   > (mayor 1 2 3 4)          => 4
   > (mayor 9 9 9 9)          => 9
   ```

2. Programar la función **paronon** que reciba 4 argumentos numéricos y regrese un símbolo que indique si hay más pares, más nones o un empate.

   Probar con:

   ```
   > (paronon 7 5 3 1)        => nones
   > (paronon 1 2 3 4)        => empate
   > (paronon 2 6 7 8)        => pares
   ```

3. Programar el función recursiva **serie** que calcule la suma de n términos de la serie:

$$\left(\frac{2}{1}+1\right)+\left(\frac{2}{2}+1\right)+\left(\frac{2}{3}+1\right)+\cdots+\left(\frac{2}{n}+1\right)=\sum_{a=1}^{a=n}\left(\frac{2}{a}+1\right)$$

   Probar con:

   ```
   > (serie 1)        => 3
   > (serie 4)        => 8.1666666
   ```

4. Programar la función **fibo3** que regrese el n-ésimo elemento de la serie ampliada de Fibonacci: 1,1,1,3,5,9,17,31,… donde cada elemento después de los primeros 3 se calcula sumando sus 3 predecesores.

   Probar con:

   ```
   > (fibo3 2)            => 1
   > (fibo3 8)            => 31
   ```

5. Obtener la versión terminal de la función fibo3 como **fibo3t**. En este caso se requerirá de más de 1 argumento adicional en la función auxiliar. Se puede comparar tiempos de ejecución con la función **(time <llamada>).**

   Probar con:

   ```
   > (fibo3t 2)       => 1
   > (fibo3t 8)       => 31
   ```

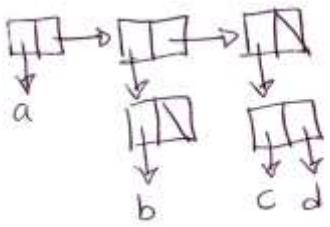
II. Programa las siguientes funciones **CON** listas.

6. A continuación se muestran listas en diferentes formas de representación: visual, física (en forma gráfica) o con constructores.

   Para cada inciso encuentra las 2 representaciones equivalentes que faltan.

   a) `(cons (cons (cons 'a 'b) '())(cons 'c (cons 'd '())))`

   b)



   c) `(1 (2 . 3) . 4)`

7. Programar la función recursiva **multiplica** que regresa la lista que resulta de multiplicar un número por los elementos de una lista.
   Probar con:
   ```
   >(multiplica 4 '(-3))          => (-12)
   >(multiplica 4 '(1 2 3 4 5))   => (4 8 12 16 20)
   ```

8. Programar la función recursiva **entero** que regresa el número entero representado por una lista de dígitos. Si el primero es negativo, el número resultante también lo debe de ser.
   Probar con:
   ```
   >(entero '(3 4 7))             => 347
   >(entero '(-2 4 3 1)           => -2431
   ```

9. Programar la función recursiva **intercala** que regresa la lista resultante de intercalar los elementos de dos listas hasta que alguna de las listas se acabe.
   Probar con:
   ```
   >(intercala '(1 2 3) '(4 5 6))  => (1 4 2 5 3 6)
   >(intercala '(1 2 3 4) '(5 6))  => (1 5 2 6)
   >(intercala '(1 2) '(3 4 5 6))  => (1 3 2 4)
   ```

10. Programar el predicado recursivo **profundidad?** que determine si en una lista (primer argumento) existe algún elemento a una profundidad dada (segundo argumento). Asumir que el nivel mínimo es 1.

    Probar con:
    ```
    >(profundidad? '(a b c) 2)          => #f
    >(profundidad? '(a (b) c) 2)        => #t
    >(profundidad? '(0 (1 (2) 1) 0) 3)  => #t
    ```

11. Programar la función recursiva **tabla** que cree una tabla con N renglones (primer argumento) y M columnas (segundo argumento) llena con el mismo valor V (tercer argumento). Asumir que los valores dados para N y M son mayores que 0.
    Probar con:

```
>(tabla 1 1 'a)     => ((a))
>(tabla 2 4 0)      => ((0 0 0 0)(0 0 0 0))
```

12. Programar la función recursiva **concatena** que concatene todas las sublistas de una lista posiblemente imbricada. Debe eliminar todos los elementos que no sean sublistas.

    Probar con:

```
>(concatena '((1 2)(a b)(3 4)))          => (1 2 a b 3 4)
>(concatena '(a (b) c))                  => (b)
>(concatena '(0 (1 (2) 1) (0 (1 (2))))) => (1 (2) 1 0 (1 (2)))
```