TC2006 – Lenguajes de Programación

Programación Avanzada en Racket

Para realizarse en equipos de 2 integrantes (3 estrellas)

1. Programar la función recursiva **unos** que, dado un arreglo ordenado de números binarios, donde todos los ceros aparecen antes de todos los unos, cuente y regrese la cantidad de unos en el arreglo.
   Probar con:
   ```
   >(unos '(0 0 0 0 1 1 1))      => 3
   >(unos '(0 0 1 1 1 1 1))      => 5
   ```

2. Programar la función recursiva **invierte** que, dada una matriz de números binarios, regrese una matriz similar, pero donde todos los ceros aparezcan como unos y todos los unos como ceros. Asumir que la matriz más pequeña contiene al menos un elemento.
   Probar con:
   ```
   >(invierte '((1)))                 => ((0))
   >(invierte '((0 1 0)(1 0 1)))      => ((1 0 1)(0 1 0)))
   ```

3. Programar la función recursiva **parcial** que, dados un número de calificación y una tabla de alumnos con registros en formato: **(matrícula (nombre) (calificaciones))**, regrese la matrícula y la calificación del parcial especificado**.** Regresar la matrícula y el símbolo NO para un alumno que no tenga la calificación especificada. El número de alumnos y la cantidad de calificaciones de cada alumno puede variar.
   Probar con:
   ```
   >(parcial 2 '((a0111111 (Jorge Perez) (100 100))
                 (a0222222 (Gloria Flores) (90))
                 (a0333333 (Ramiro Mendez) (90 60 90)) ...))
   => ((a0111111 100) (a0222222 NO) (a0333333 60) ...)
   ```

4. Programar la función recursiva **subarbol** que, dados el identificador de un nodo y un árbol binario, regrese el subárbol que se genera a partir del nodo especificado (este nodo aparecerá como la raíz de ese subárbol). Regresar una lista vacía si el nodo especificado no existe.
   Probar con:
   ```
   >(subarbol 'b '(a(b(c()())(d()()))(e()(f(g()())()))))
      => (b(c()())(d()()))
   >(subarbol 15 '(8(5(2()())(7()()))(9()(15(11()())()))))
      => (15(11()())())
   ```

5. Programar la función recursiva **adyacentes** que, dados un identificador de nodo y un grafo representado como nodos y aristas, regrese los nodos adyacentes al nodo especificado (nodos con los que tiene una arista).
   Probar con:
   ```
   >(adyacentes 3 '((1 2 3 4)((1 2)(1 4)(2 3)(2 4)(3 4))))  => (2 4)
   >(adyacentes 'a '((a b c)((a b)(c b)(c a))))              => (b c)
   ```

Resuelve los siguientes problemas SIN utilizar funciones de orden superior. Debes de subir a blackboard el archivo fuente (texto) que contenga todas las funciones.

6. Programar la función recursiva **aplica-listas** que aplique una lista de funciones binarias a cada elemento correspondiente en dos listas del mismo tamaño para obtener una lista de sublistas con los resultados de cada operador.
   Probar con:
```
> (aplica-listas (list + - * /) '(1 2 3) '(4 5 6))
   => ((5 7 9)(-3 -3 -3)(4 10 18)(1/4 2/5 1/2))
> (aplica-listas (list cons list append) '((a b)) '((c d)))
   => ((((a b) c d))(((a b)(c d)))((a b c d)))
```

7. Programar el predicado recursivo **alguno?** que verifique si al menos un par elementos correspondientes de dos listas cumplen con un predicado binario.
   Probar con:
```
> (alguno? < '(9 2 10) '(6 7 8))  => #t
> (alguno? (lambda (x y) (negative? (- x y))) '(5 8 10) '(4 5 6))
   => #f
> (alguno? > '(3 1 15) '(6 7 8))  => #t
```

8. Programar la función **verbosa** que genere un procedimiento unario que despliegue el valor de su argumento y su resultado además de regresarlo.
   Probar con:
```
> (define vinc (verbosa (lambda (x) (+ x 1))))
> (vinc 5)
   => Entrada = 5
      Salida = 6
    6
> (define vcdr (verbosa cdr))
> (vcdr '(1 2 3))
   => Entrada = (1 2 3)
      Salida = (2 3)
      (2 3)
```

9. Programar la función de orden superior **filtra** que sin utilizar recursividad explícita elimine de una matriz de números todos los elementos que NO cumplan una condición unaria que se le pase como argumento. No utilizar el primitivo **filter**.
   Probar con:
```
> (filtra negative? '((1 -2 3 4)(-5 6 -7 -8))) => ((-2)(-5 -7 -8))
> (filtra (lambda (x) (> x 5)) '((4 9)(1 2)(10 7))) => ((9)()(10 7))
```

10. Programar la función de orden superior **impares** que sin utilizar recursividad explícita extraiga todos los números impares dentro de una matriz.
    Probar con:
```
> (impares '((1 2 3)(4 5 6)))      => (1 3 5)
> (impares '((2 4)(2 1)(3 2)))     => (1 3)
```

11. Programar la función de orden superior **inserta** que sin utilizar recursividad explícita que inserte un valor después de cada elemento de una lista.

Probar con:

```
> (inserta 1 '(1 2 3 4))      => (1 1 2 1 3 1 4 1)
> (inserta 'a '(b (c) (d e))) => (b a (c) a (d e) a)
```

12. Programar la función de orden superior **multifnc** que sin utilizar recursividad explícita aplique una lista de funciones unarias a una lista de valores para obtener una matriz de resultados.

    Probar con:

```
> (multifnc '(sqr sqrt (lambda (v) (/ 1 v))) '(1 4 9))
   => ((1 16 81)(1 2 3)(1 1/4 1/9))
> (multifnc '(car cdr list) '((1 2)(a b)))
   => ((1 a)((2)(b))(((1 2))((a b))))
```