Taller 1: Definición recursiva de programas e inducción (Racket)

Fundamentos de Lenguajes de Programación / 750095M / Grupo 01 / Prof. Robinson Duque / Monitor Juan Marcos Caicedo / 2019-2

1. **(5pts)** Elabore una función llamada *invert* que recibe un argumento: una lista **L**, sin embargo, esta lista **L** se compone de pares x, y que a su vez son listas (de tamaño 2). La función debe retornar una lista similar a **L**, con pares ordenados invertidos, es decir, y, x. *Ejemplos:*

```
> (invert '((a 1) (a 2) (1 b) (2 b)))
((1 a) (2 a) (b 1) (b 2))
> (invert '((5 9) (10 91) (82 7) (a e) ("hola" "Mundo")))
((9 5) (91 10) (7 82) (e a) ("Mundo" "hola"))
> (invert '(("es" "racket") ("genial" "muy") (17 29) (81 o)))
(("racket" "es") ("muy" "genial") (29 17) (o 81))
```

2. **(5pts)** Elabore una función llamada *down* que recibe como argumento una lista **L**, y lo que debe realizar dicha función es retornar una lista con cada elemento de **L** asociado a un nivel más de paréntesis comparado con su estado original en **L**. *Ejemplos:*

```
> (down '(1 2 3))
((1) (2) (3))
> (down '((una) (buena) (idea)))
(((una)) ((buena)) ((idea)))
> (down '(un (objeto (mas)) complicado))
((un) ((objeto (mas))) (complicado))
```

3. **(5pts)** Elabore una función llamada *list-set* que reciba tres argumentos: una lista **L**, un número **n** y un elemento **x**. La función debe retornar una lista similar a la que recibe (**L**), pero debe tener en la posición ingresada **n** (indexando desde cero) el elemento **x**. *Ejemplos:*

```
> (list-set '(a b c d) 2 '(1 2))
(a b (1 2) d)
> (list-set '(a b c d) 3 '(1 5 10))
(a b c (1 5 10))
```

4. **(5pts)** Elabore una función llamada *filter-in* que debe recibir dos argumentos: un predicado **P** y una lista **L**. La función retorna una lista que contiene los elementos que pertenecen a **L** y que satisfacen el predicado **P**. *Ejemplos:*

```
> (filter-in number? '(a 2 (1 3) b 7))
(2 7)
> (filter-in symbol? '(a (b c) 17 foo))
(a foo)
> (filter-in string? '(a b u "univalle" "racket" "flp" 28 90 (1 2 3)))
("univalle" "racket" "flp")
```

5. **(4pts)** Elabore una función llamada *list-index* que debe recibir dos argumentos: un predicado **P** y una lista **L**. La función retorna (desde una posición inicial 0) el primer elemento de la lista que satisface el predicado **L**. Si llega a suceder que ningún elemento satisface el predicado recibido, la función debe retornar **#f**. *Ejemplos:*

```
> (list-index number? '(a 2 (1 3) b 7))
1
> (list-index symbol? '(a (b c) 17 foo))
0
> (list-index symbol? '(1 2 (a b) 3))
#f
```

6. **(5pts)** Elabore una función llamada *swapper* que recibe 3 argumentos: un elemento **E1**, otro elemento **E2**, y una lista **L**. La función retorna una lista similar a **L**, sólo que cada ocurrencia anterior de **E1** será reemplazada por **E2** y cada ocurrencia anterior de **E2** será reemplazada por **E1** (Los elementos **E1** y **E2** deben pertenecer a **L**). *Ejemplos:*

```
> (swapper 'a 'd '(a b c d))
(d b c a)
> (swapper 'a 'd '(a d () c d))
(d a () c a)
> (swapper 'x 'y '(y y x y x y x x y))
(x x y x y x y y x)
```

7. **(4pts)** Elabore una función llamada *cartesian-product* que recibe como argumentos 2 listas de símbolos sin repeticiones **L1** y **L2**. La función debe retornar una lista de tuplas que representen el producto cartesiano entre **L1** y **L2**. Los pares pueden aparecer en cualquier orden. *Ejemplos:*

```
> (cartesian-product '(a b c) '(x y))
((a x) (a y) (b x) (b y) (c x) (c y))
> (cartesian-product '(p q r) '(5 6 7))
((p 5) (p 6) (p 7) (q 5) (q 6) (q 7) (r 5) (r 6) (r 7))
```

8. **(4pts)** Elabore una función llamada *mapping* que debe recibir como entrada 3 argumentos: una función unaria (que recibe un argumento) llamada **F**, y dos listas de números **L1** y **L2**. La función debe retornar una lista de pares **(a,b)** siendo **a** elemento de **L1** y **b** elemento de **L2**, cumpliéndose la propiedad que al aplicar la función unaria **F** con el argumento **a**, debe arrojar el número **b**. Es decir, se debe cumplir que $\mathbf{F(a) = b}$. (Las listas deben ser de igual tamaño). *Ejemplos:*

```
> (mapping (lambda (d) (* d 2)) (list 1 2 3) (list 2 4 6))
((1 2) (2 4) (3 6))
> (mapping (lambda (d) (* d 3)) (list 1 2 2) (list 2 4 6))
((2 6))
> (mapping (lambda (d) (* d 2)) (list 1 2 3) (list 3 9 12))
()
```

9. **(4pts)** Elabore una función llamada *inversions* que recibe como entrada una lista **L**, y determina el número de inversiones de la lista **L**. De manera formal, sea $A = (a_1 a_2 ... a_n)$ una lista de n números <u>diferentes</u>, si $i < j$ (posición) y $a_i > a_j$ (dato en la posición) entonces la pareja $(i\ j)$ <u>es una inversión de A</u>. *Ejemplos:*

```
> (inversions '(2 3 8 6 1))
5
> (inversions '(1 2 3 4))
0
> (inversions '(3 2 1))
3
```

10. **(5pts)** Elabore una función llamada *up* que recibe como entrada una lista **L**, y lo que debe realizar la función es remover un par de paréntesis a cada elemento del nivel más alto de la lista. Si un elemento de este nivel no es una lista (no tiene paréntesis), este elemento es incluido en la salida resultante sin modificación alguna. *Ejemplos:*

```
> (up '((1 2) (3 4)))
(1 2 3 4)
> (up '((x (y)) z))
(x (y) z)
```

11. **(4pts)** Elabore una función llamada *zip* que recibe como entrada tres parámetros: una función binaria (función que espera recibir dos argumentos) **F**, y dos listas **L1** y **L2**, ambas de igual tamaño. El procedimiento *zip* debe retornar una lista donde la posición n-ésima corresponde al resultado de aplicar la función **F** sobre los elementos en la posición n-ésima en **L1** y **L2**. *Ejemplos:*

```
> (zip + '(1 4) '(6 2))
(7 6)
> (zip * '(11 5 6) '(10 9 8))
(110 45 48)
```

12. **(4pts)** Elabore una función llamada *filter-acum* que recibe como entrada 5 parámetros: dos números **a** y **b**, una función binaria **F**, un valor inicial **acum** y una función unaria **filter**. El procedimiento *filter-acum* aplicará la función binaria **F** a todos los elementos que están en el intervalo $[a, b]$ y que a su vez todos estos elementos cumplen con el predicado de la función **filter**, el resultado se debe ir conservando en **acum** y debe retornarse el valor final de **acum**. *Ejemplos:*

```
> (filter-acum 1 10 + 0 odd?)
25
> (filter-acum 1 10 + 0 even?)
30
```

13. **(7pts)** Elabore una función llamada `(operate lrators lrands)` donde *lrators* es una lista de funciones binarias de tamaño n y *lrands* es una lista de números de tamaño $n+1$. La función retorna el resultado de aplicar sucesivamente las operaciones en lrators a los valores en *lrands*.

```
> (operate (list + * + - *) '(1 2 8 4 11 6))
102
> (operate (list *) '(4 5))
20
```

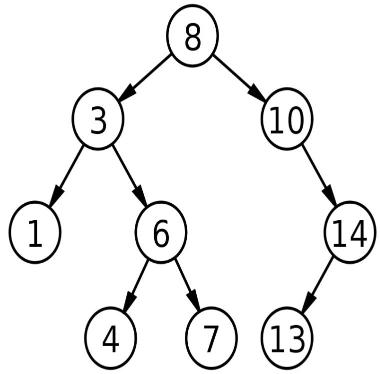
En el ejemplo anterior, el resultado es 102 puesto que $(((((1 + 2) * 8) + 4) - 11) * 6) = 102$ y 20 puesto que $(4 * 5) = 20$.

14. **(6pts)** Elabore una función llamada *path* que recibe como entrada dos parámetros: un número **n** y un árbol binario de búsqueda (representando con listas) **BST** (el árbol debe contener el número entero **n**). La función debe retornar una lista con la ruta a tomar (iniciando desde el nodo raíz del árbol), indicada por cadenas left y right, hasta llegar al número **n** recibido. Si el número **n** es encontrado en el nodo raíz, el procedimiento debe retornar una lista vacía. *Ejemplo:*

```
> (path 17 '(14 (7 () (12 () ()))
                (26 (20 (17 () ())
                        ())
                    (31 () ()))))
(right left left)
```

Nota aclaratoria: Para el ejercicio se utiliza la representación de Árbol Binario de Búsqueda con Listas en Racket, y podría representarse con la ayuda de la siguiente gramática BNF:

```
<árbol-binario> := (árbol-vacío) empty
                := (nodo) número <árbol-binario> <árbol-binario>
```



Es decir que este Árbol Binario de Búsqueda, representado en Racket con listas y usando la anterior gramática, sería:

```
'(8 (3 (1 () ()) (6 (4 () ()) (7 () ()))) (10 () (14 (13 () ()) ())))
```

15. **(6pts)** Elabore una función llamada `(count-odd-and-even arbol)` que toma un árbol binario y retorna una lista con dos elementos correspondientes a la cantidad de pares e impares en arbol.

```
> (count-odd-and-even '(14 (7 () (12 () ()))
                          (26 (20 (17 () ())
                                  ())
                              (31 () ()))))
(4 3)
```

El resultado del ejemplo anterior es la lista (4 3) puesto que en el árbol suministrado hay 4 pares y 3 impares.

16. **(9pts)** Elabore una función llamada `(simpson-rule f a b n)` que cálcula la integral de una función `f` entre los valores `a` y `b` mediante la regla de Simpson. Dicha regla utiliza la siguiente aproximación:

$$\frac{h}{3} \cdot \big(y_0 + 4y_1 + 2y_2 + 4y_3 + 2y_4 + \cdots + 2y_{n-2} + 4y_{n-1} + y_n\big)$$

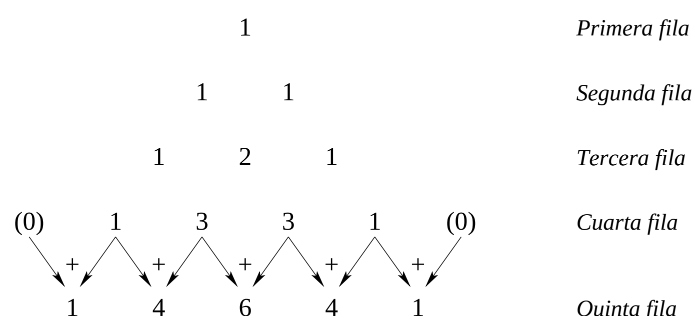
donde $h = \frac{(b-a)}{n}$ para algún entero par `n` y $y_k = f(a + kh)$.

```
> (simpson-rule (lambda (x) (* x (* x x))) 1 5 8)
156
> (simpson-rule (lambda (x) x) 1 5 12)
12
```

17. **(9pts)** Elabore una función llamada `(prod-scalar-matrix mat vec)` que recibe una matriz `mat` representada como una lista de listas y un vector `vec` representado como una lista y retorna el resultado de realizar la multiplicación matriz por vector.

```
> (prod-scalar-matriz '((1 1) (2 2)) '(2 3))
'((2 3) (4 6))
> (prod-scalar-matriz '((1 1) (2 2) (3 3)) '(2 3))
'((2 3) (4 6) (6 9))
```

18. **(9pts)** Elabore una función llamada `(pascal N)` que retorna la fila `N` del triangulo de Pascal. A continuación se muestra las primeras cinco filas del triangulo de Pascal:



Ayuda: Note que la fila `N` depende de la anterior, por ejemplo la quinta fila:

$$\begin{aligned}
\texttt{(1 4 6 4 1)} &= \\
\texttt{(0 1 3 3 1)} &+ \\
\texttt{(1 3 3 1 0)} &
\end{aligned}$$

```
> (pascal 5)
(1 4 6 4 1)
> (pascal 1)
(1)
```

Aclaraciones

1. El taller es en grupos de máximo tres (3) alumnos.

2. La solución del taller debe ser subida al campus virtual a más tardar el día **14 de Noviembre (23:59)**. Se debe subir al campus virtual en el enlace correspondiente a este taller un archivo comprimido **.zip** que siga la convención *CódigodeEstudiante1-CódigodeEstudiante2-CódigodeEstudiante3-Taller1FLP20192.zip*. Este archivo debe contener el archivo **ejercicios-taller1.rkt** que contenga el desarrollo de los ejercicios.

3. En las primeras líneas del archivo **ejercicios-taller1.rkt** deben estar comentados los nombres y los códigos de los estudiantes participantes.

4. También deben de documentar los procedimientos que hayan implementado como solución a los problemas, las expresines BNF de las estructuras que se están utilizando, y de igual manera las funciones auxiliares, con ejemplos de prueba (mínimo 2 pruebas). Por ejemplo, si se pide un procedimiento *remove-first* debe ir así:

```
;; remove-first :
;; Proposito:
;; S x L -> L' : Procedimiento que remueve la primera
;; ocurrencia de un simbolo S en una lista de simbolos L.
;;
;;<lista> := ()
;;        := (<valor-de-scheme>  <lista>)

(define remove-first
   (lambda (s los)
      (if (null? los)
          '()
          (if (eqv? (car los) s)
              (cdr los)
              (cons (car los)
                    (remove-first s (cdr los)))))))

;; Pruebas
```

```
(remove-first 'a '(a b c))
(remove-first 'b '(e f g))
(remove-first 'a4 '(c1 a4 c1 a4))
(remove-first 'x '())
```

5. En caso de tener dudas, puede consultar con el profesor **Robinson Duque** (Martes 10-11am, Jueves 4-5pm, o con cita previa) o consultar con el monitor **Juan Marcos Caicedo** en el horario de atención de los Lunes de 2 pm a 4 pm (en el Laboratorio del Grupo de Investigación Avispa, 3er piso).

Entregas Tardías o por Otros Medios

1. Este taller **sólo se recibirá a través del campus virtual**. Adicionalmente, sólo se evaluarán los documentos solicitados en el punto 2 de la sección anterior. Cualquier otro tipo de correo o nota aclaratoria será descartado. Sólo se aceptan envíos por fuera del horario establecido bajo excusas de fuerza mayor validadas a través de la dirección de la Escuela.

2. Las entregas tarde serán penalizadas así: (-1pt) por cada hora de retraso o fracción. Por ejemplo, si usted realiza su entrega y el campus registra las 24:00 (i.e., 1min después de la hora de entrega), usted está incurriendo en la primer hora de retraso. Asegurese con mínimo dos horas de anticipación que el link de carga funciona correctamente toda vez que es posible incurrir en una entrega tardía debido a los tiempos de respuesta.