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

Fundamentos de Interpretación y Compilación de Lenguajes de Programación / 750017C / Grupo 01 / Prof. Robinson Duque / Mauricio Muñoz / 2024-1

Evaluación por Indicadores - RES 047

Los indicadores de logro a evaluar en este Taller son:

- IL 1.1.1 Construye datos recursivamente utilizando métodos de inducción y gramáticas BNF (ejercicios 1 a 13)
- IL 1.1.2 Utiliza gramáticas BNF para guiar la construcción de programas recursivos (ejercicios 14 a 18)

Este taller tiene un valor global de 5.44% distribuidos en (3.2%) para el IL 1.1.1 y (2.24%) para el IL 1.1.2. El taller se calificará con una nota en escala de 0 a 100 puntos (que luego se escalará de 0.0 a 5.0). El valor de cada indicador se muestra a continuación:

$$IL1.1.1 = \frac{3.2 * 100}{5.44} = 59 puntos(aprox)$$

$$IL1.1.2 = \frac{2.24 * 100}{5.44} = 41 puntos(aprox)$$

Cada ejercicio será probado con un conjunto de pruebas unitarias que deberá pasar para asignar la totalidad de los puntos indicados. En caso de no pasar las pruebas, el valor a asignar será de 0.0pts. Asegurese de respetar el contrato de cada función.

Ejercicios - IL 1.1.1 (59pts)

1. **(4.5pts)** 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. **(4.5pts)** 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. **(4.5pts)** 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. **(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. **(4.5pts)** Elabore una función llamada *mix* que debe recibir dos argumentos, una lista **L1** y otra lista **L2**. La función retorna una sola lista con los elementos cruzados entre ambas listas. *Ejemplos:*

```
> (mix '(1 2 3 4 5) '(6 7 8 9 10))
(1 6 2 7 3 8 4 9 5 10)
> (mix '(a b c) '(7 8 9))
(a 7 b 8 c 9)
> (mix '(1 (5 2) foo) '(h  (1 3) flp))
(1 h (5 2) (1 3) foo flp)
```

6. **(4.5pts)** 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. **(4.5pts)** 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. **(4.5pts)** 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. **(4.5pts)** Elabore una función llamada *reverse* que recibe como entrada una lista **L**, y retorna la misma lista con los elementos en posiciones invertidas. *Ejemplos:*

```
> (reverse '(2 3 8 6 1))
(1 6 8 3 2)
> (reverse '(1 2 3 4))
(4 3 2 1)
> (reverse '(h o l a m u n d o))
(o d n u m a l o h)
```

10. **(4.5pts)** Elabore una función llamada *flatten* que recibe como entrada una lista **L**, y retorna la lista "aplanada", es decir, sin ningún tipo de lista anidada interna conservando todos los elementos. *Ejemplos:*

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

11. **(4.5pts)** Elabore una función llamada *unzip* que recibe como entrada una lista de tuplas **L**, es decir, una lista de listas de dos elementos, y retorna dos listas resultantes de descomponer la lista original en dos, la primera con el primer elemento de las tuplas y la segunda con el segundo elemento de las tuplas. *Ejemplos:*

```
> (unzip  '((a 1) '(b 2) (c 3))
((a b c) (1 2 3))
> (unzip  '((1 2) (3 4) (5 6)))
((1 3 5) (2 4 6))
```

12. **(4.5pts)** Elabore una función llamada *scan* que recibe como entrada 3 parámetros: Una lista **L**, un elemento **n** y una función binaria **F**. El procedimiento scan empezará tomando el elemento **n** y aplicara la función binaria **F** con cada elemento de la lista de forma acumulativa, la respuesta de la función será una lista con cada resultado parcial empezando con el elemento **n**. *Ejemplos:*

```
> (scan '(1 2 3 4 5) 0 +)
(0 1 3 6 10 15)
> (scan '(2 4 8) 1 *)
(1 2 8 64)
```

13. **(5pts)** 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.

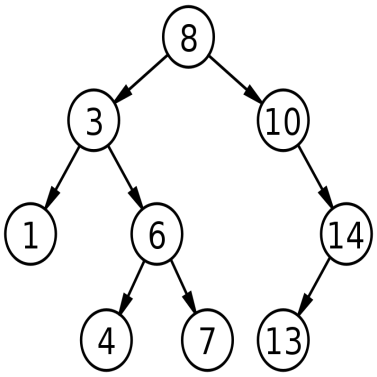
Ejercicios -IL 1.1.2 (41%)

14. **(8pts)** 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. **(8pts)** Elabore una función llamada `inorder` que toma un árbol binario y retorna una lista con los elementos del árbol correspondientes a recorrerlo inorder. En el recorrido inorder los nodos se visitan de la forma (izquierda, raiz, derecha). *Ejemplos:*

```
(in-order '(21 (0 (4 () ()) (4 () ()))
               (52 (14 (6 (8 () ()) ())
                       (5 () ()))
                   (19 () ()))))
```

```
(4 0 4 21 8 6 14 5 52 19)

> (inorder '(14 (7 () (12 () ()))
                        (26 (20 (17 () ())
                                ())
                            (31 () ()))))
(7 12 14 17 20 26 31)
```

En este caso dado que el árbol es un árbol binario de busqueda (BTS) al hacer el recorrido inorder se obtiene una lista con los elementos ordenados ascendentemente.

16. **(8pts)** Dada la siguiente gramática sobre operaciones binarias:

```
<OperacionB>::= <int>
            ::= (<OperacionB> 'suma <OperacionB>)
            ::= (<OperacionB> 'resta <OperacionB>)
            ::= (<OperacionB> 'multiplica <OperacionB>)
```

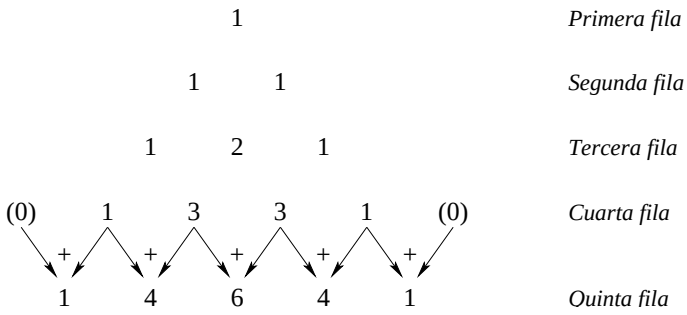
Implementa una función llamada `(Operar-binarias operacionB)` que recibe como parámetro una operación binaria válida y retorna el resultado de hacer las operaciones suma, resta y multiplicación correspondientes.

```
> (Operar-binarias  4)
4
> (Operar-binarias  '(2 suma 9) )
11
> (Operar-binarias  '(2 resta 9) )
-7
> (Operar-binarias  '(2 multiplica 9) )
18
> (Operar-binarias  '( (2 multiplica 3) suma (5 resta 1 ) ) )
10
> (Operar-binarias  '( (2 multiplica (4 suma 1) )
                       multiplica
                     ( (2 multiplica 4) resta 1 ) ) )
70
```

17. **(8pts)** Elabore una función llamada `(prod-scalar-matriz 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:

```
(1 4 6 4 1) =
(0 1 3 3 1) +
(1 3 3 1 0)
```

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

Aclaraciones

1. El taller es en grupos de dos (2) estudiantes.

2. La solución del taller debe ser subida al campus virtual a más tardar el día indicado en el campus virtual, se debe subir al campus virtual en el enlace correspondiente a este taller un archivo comprimido **.zip** que siga la convención *CódigodeEstudiante1-CódigodeEstudiante2-CódigodeEstudiante3-Taller1FLP20241.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 '())
```

LA NO INCLUSIÓN DE LA DOCUMENTACIÓN DEL CÓDIGO, CONLLEVARÁ A UN DESCUENTO DEL 20% EN CADA EJERCICIO.

5. **Importante**: Recuerde mantener la nomenclatura de las funciones que se piden (no les cambie el nombre), así como el orden y la cantidad de parámetros, en caso de que considere necesario añadir un parámetro, recurra al uso de funciones auxiliares.

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.

2. Las entregas tarde serán penalizadas así: (-1pt de la nota final obtenida) 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.