



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

Fundamentos de Interpretación y Compilación de Lenguajes de Programación / Grupo 01 / Prof. Robinson Duque / Emily Nuñez / 2026-I

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 15)

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:

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

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

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 como argumentos: una lista **L** y un predicado **P**. La 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 , sólo cuando ambos elementos de la lista cumplan con un predicado **P**. *Ejemplos:*

```
> (invert '((3 2) (4 2) (1 5) (2 8)) even?)
((2 4) (8 2))
> (invert '((5 9) (10 90) (82 7) ) multiplo5? )
((90 10))
> (invert '((6 9) (10 90) (82 7) ) odd? )
()
```

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 cuatro argumentos: una lista **L**, un número **n**, un elemento **x** y un predicado **P**. 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** sólo si el elemento original de la lista cumple con el predicado **P**. *Ejemplos:*

```
> (list-set '(5 8 7 6) 2 '(1 2) odd?)
(5 8 (1 2) 6)
> (list-set '(5 8 7 6) 2 '(1 2) even?)
(5 8 7 6)
> (list-set '(5 8 7 6) 3 '(1 5 10) mayor5? )
(5 8 7 (1 5 10) )
> (list-set '(5 8 7 6) 0 '(1 5 10) mayor5? )
(5 8 7 6)
```

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 (*palindrome?* palabra) que determine si una palabra es palíndromo. La función recibe como parámetro una lista de símbolos o caracteres *palabra* y retorna **#t** si la palabra se lee igual de izquierda a derecha que de derecha a izquierda, y **#f** en caso contrario. La solución debe realizarse de manera recursiva.

Ejemplos:

```
> (palindrome? '(r a d a r))
#t
> (palindrome? '(n e u q u e n))
#t
> (palindrome? '(h o l a))
#f
```

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

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

10. **(4.5pts)** Elabore una función llamada `(balanced-parentheses? L)` que determine si una lista de símbolos que representa paréntesis está correctamente balanceada. La función recibe como parámetro una lista `L` cuyos elementos pueden ser los símbolos `(` y `)`. La función debe retornar `#t` si los paréntesis están correctamente balanceados y `#f` en caso contrario. Un conjunto de paréntesis está balanceado si:

- Cada paréntesis de apertura tiene su correspondiente paréntesis de cierre.
- En ningún punto del recorrido la cantidad de paréntesis de cierre supera a la de apertura.

La solución debe ser recursiva.

Ejemplos:

```
> (balanced-parentheses? '(()()))
#t
```

```
> (balanced-parentheses? '((((())))
#t
```

```
> (balanced-parentheses? '(()))
#f
```

```
> (balanced-parentheses? '())(())
#f
```

11. **(4.5pts)** 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. **(4.5pts)** 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. **(5pts)** Elabore una función llamada (`operate lrators lrand`s) donde *lrators* es una lista de funciones binarias de tamaño n y *lrand*s 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 *lrand*s.

```
> (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)

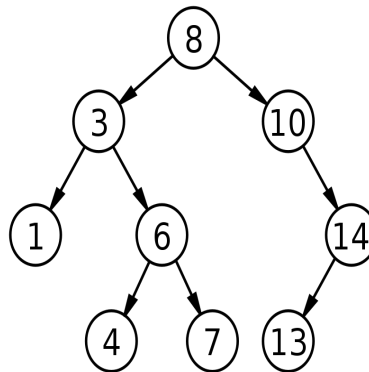
```

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 `(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. (8pts) Elabore una función llamada (`hanoi n origen auxiliar destino`) que resuelva el problema clásico de las Torres de Hanoi. La función recibe como parámetros un número entero positivo `n` (cantidad de discos) y tres símbolos que representan las torres: `origen`, `auxiliar` y `destino`. La función debe retornar una lista con la secuencia de movimientos necesarios para trasladar los `n` discos desde la torre `origen` hasta la torre `destino`, utilizando la torre `auxiliar`. Cada movimiento debe representarse como una lista de dos elementos: (`desde hacia`).

Ejemplos:

```
> (hanoi 1 'A 'B 'C)
((A C))
> (hanoi 2 'A 'B 'C)
((A B) (A C) (B C))
> (hanoi 3 'A 'B 'C)
((A C) (A B) (C B) (A C) (B A) (B C) (A C))
```

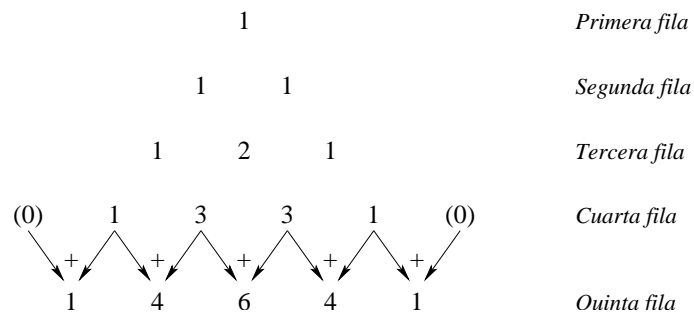
17. (8pts) Elabore una función llamada (`coin-change monto monedas`) que calcule el número total de formas distintas de obtener un monto dado utilizando una lista de denominaciones de monedas. La función recibe como parámetros:

- `monto`: un número entero no negativo.
- `monedas`: una lista de números enteros positivos que representan las denominaciones disponibles.

La función debe retornar el número de combinaciones posibles para obtener exactamente el valor `monto`. La solución debe implementarse utilizando definición recursiva. *Ejemplos:*

```
> (coin-change 5 '(1 5))
2
> (coin-change 5 '(1 2 5))
4
> (coin-change 10 '(2 5 3 6))
5
```

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:

$$\begin{array}{r}
 (1 \ 4 \ 6 \ 4 \ 1) = \\
 (0 \ 1 \ 3 \ 3 \ 1) + \\
 (1 \ 3 \ 3 \ 1 \ 0)
 \end{array}$$

```

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

```

Aclaraciones

1. El taller es en grupos de mínimo dos (2) alumnos, máximo tres (3).
2. La solución del taller debe ser subida al campus virtual en la fecha especificada allí mismo. Se debe subir al campus virtual en el enlace correspondiente a este taller un archivo comprimido **.zip** que siga la convención *Código de Estudiante1-Código de Estudiante2-Código de Estudiante3-Taller1FLP20261.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 documentar los procedimientos que hayan implementado como solución a los problemas, las expresiones BNF de las estructuras que se están utilizando, y de igual manera para las funciones auxiliares, con ejemplos de prueba (mínimo 2 pruebas). Por ejemplo, si se pide un procedimiento *remove-first* debe ir así:

```
;; remove-first :  
;; Propósito:  
;; S x L -> L' : Procedimiento que remueve la primera  
;; ocurrencia de un símbolo S en una lista de símbolos 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. **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.
6. En caso de tener dudas, puede consultar con la monitora **Emily Nuñez** - emily.nunez@correounivalle.edu.co en el horario de atención que se defina (en el Laboratorio del Grupo de Investigación Avispa, 3er piso) o vía correo electrónico.

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.