INF-253 Lenguajes de Programación
Tarea 4 2025-2: Extensiones a la nave!

Profesores: José Luis Martí Lara, Wladimir Ormazábal Orellana, Ricardo Salas Letelier
Ayudante coordinador: Bastián Ríos Lastarria
Ayudantes: Diego Duarte, Andrés Jablonca, Cristobal Tirado,
Martin Pardo, Carlos Bravo, Martín Palacios,
Blas Olivares, Sofía Ramirez, Benjamín Echeverria

20 de octubre de 2025

Índice

1. Introducción

Tras la finalización de tu videojuego, decides enviárselo a una editora para la publicación oficial de éste, la editora BlueTrident le encantó tu propuesta pero tiene ciertos comentarios y mejoras que hacerle, es por esto que decides realizar una expansión a tu juego utilizando el lenguaje de programación funcional Scheme para las ideas que comenta la empresa.

2. Descripción de la tarea

2.1. P1) Calibrador de telemetría (recursión simple y de cola, función `lambda`)

BlueTrident solicita un módulo de **calibración de telemetría** para las lecturas del sonar/eco de la nave. Las lecturas se representan como una lista de números (enteros o reales) y deben ser transformadas por una función de corrección parametrizable. Se deben implementar dos variantes: una con **recursión simple** y otra con **recursión de cola** que preserve el orden de las lecturas.

Objetivo: Aplicar una función de corrección a cada lectura, generando una nueva lista con las lecturas calibradas, preservando el orden original.

Firmas

```
(calibra-simple f lecturas) ; (recursión simple)
(calibra-cola f lecturas) ; (recursión de cola)
```

Parámetros

- `f`: función de una variable numérica, entregada como función `lambda` o nombre de función definido por el/la estudiante. Debe aceptar una lectura y retornar su valor calibrado.
- `lecturas`: lista (posiblemente vacía) de números que representan intensidades/retornos del sonar.

Comportamiento esperado

- Ambas funciones retornan una **nueva** lista con `(f x)` aplicado a cada elemento `x` de `lecturas`, en el **mismo orden** de aparición.
- Deben manejar correctamente la lista vacía.

Requisitos de recursión

- **Recursión simple** en `calibra-simple`: estructurar la solución descendiendo y reconstruyendo el resultado al volver de la llamada recursiva.
- **Recursión de cola** en `calibra-cola`: la *última* operación de la definición debe ser la llamada recursiva.

Restricciones

- **Prohibido**: `map`, `apply`, `filter`, cualquier `fold*`, `for-each`, `length`, mutación (`set!`), vectores.
- **Permitido**: función `lambda`, `cond`, `null?`, `pair?`, `car`, `cdr`, `cons`, `append`, `reverse`, operaciones aritméticas necesarias.

Casos de ejemplo

```
(define corr-sat (lambda (x)
(let ((y (+ x 5)))
(max 0 (min 100 y)))))

(calibra-simple corr-sat '(95 -3 50))
>'(100 2 55)
(calibra-cola corr-sat '(95 -3 50))
>'(100 2 55)

(calibra-simple (lambda (x) (if (< x 10) 0 x))
'(3 12 8 20)) ; => '(0 12 0 20)
(calibra-cola (lambda (x) (if (< x 10) 0 x))
'(3 12 8 20)) ; => '(0 12 0 20)

(calibra-simple (lambda (x) (if (negative? x) 0 (sqrt x)))
'(0 1 4 9 -5 16)) ; => '(0 1 2 3 0 4)
```

2.2. P2) Radar de recursos (listas simples y anidadas, recursión)

BlueTrident encarga la expansión *Radar Suite* para la nave exploradora. El módulo de radar debe procesar escaneos de sectores oceánicos representados como listas de símbolos y sublistas anidadas (misiones, subzonas y eventos). Se solicita implementar un **radar de recursos** que, dado un recurso objetivo (p.,ej., `titanio`, `cobre`, `litio`) y un escaneo que puede ser una lista simple o anidada, determine cuántas veces se *detecta* dicho recurso en toda la estructura.

Objetivo Implementar una función que calcule el número total de *detecciones* de un recurso objetivo dentro de un escaneo potencialmente anidado, respetando el paradigma funcional.

Firma

```
(radar-detecta recurso escaneo)
```

Parámetros

- `recurso`: símbolo que identifica el recurso (p.,ej., `titanio`, `cobre`, `litio`).
- `escaneo`: símbolo o lista (posiblemente **anidada**) de símbolos y/o sublistas que modela la jerarquía de lecturas del radar.

Comportamiento esperado La función debe recorrer **toda** la estructura de `escaneo` y retornar el total de ocurrencias de `recurso`, manejando correctamente:

- listas vacías,
- símbolos distintos al objetivo,
- anidación arbitraria de sublistas.

Restricciones

- Prohibido: `map`, `filter`, cualquier `fold*`, `length`, mutación (`set!`), `apply`, vectores.
- Permitido: `cond`, `symbol?`, `pair?`, `list?`, `null?`, `eq?`, `car`, `cdr`, `cons`, `append`.

Casos de ejemplo

```
(radar-detecta 'titanio 'titanio)
>1
(radar-detecta 'titanio 'cobre)
>0
(radar-detecta 'titanio '())
>0
(radar-detecta 'titanio '(titanio cobre titanio))
>2
(radar-detecta 'diamante
'(sector-a (cueva-1 diamante)
(cueva-2 (nodo diamante diamante)) coral))
>3
(radar-detecta 'cobre
'(scan ((cobre) ((cobre titanio) arena)) (baliza cobre)))
>3
```

Posibles dudas que puedan aparecer

- Si `escaneo` es **símbolo**: retornar `1` si `(eq? escaneo recurso)`, en otro caso `0`.
- Si `escaneo` es **lista vacía**: retornar `0`.
- Si es **lista no vacía**: sumar recursivamente el resultado en `(car escaneo)` y en `(cdr escaneo)`.

2.3. P3) Traducción de códigos en planos (map, filter, assoc, función `lambda`)

El editor considera que el uso de los planos es muy simple, es por esto que en la nueva versión, los planos vendrán encriptados en código tendrás que crear un traductor que convierta dichos códigos en nombres legibles y descarte las entradas desconocidas.

Objetivo Dada una **tabla de traducción** (alist) y una lista de **códigos** presentes en los planos, producir una lista *ordenada* con los *nombres legibles* correspondientes, eliminando los códigos sin traducción.

Firma

```
(traducir-codigos tabla codigos)
```

Parámetros

- `tabla`: alista de pares `(codigo . nombre)`. Ej.: `'((A . titanio) (B . cobre) (C . litio))`.
- `codigos`: lista (posiblemente vacía) de símbolos-código presentes en los planos.

Comportamiento esperado

- Conservar solo las traducciones existentes (descartar códigos desconocidos).
- Preservar el orden de aparición de `codigos`.

Construcciones obligatorias

- Uso de función `lambda` (p.,ej., en la función que se pasa a `map` o `filter`).
- Uso de `map`, `filter` y `assoc`.

Restricciones

- **Prohibido**:`fold*`, `apply`, `for-each`, mutación (`set!`), vectores.
- **Permitido**: función `lambda`, `map`, `filter`, `assoc`, `cond`, `null?`, `pair?`, `car`, `cdr`.

Casos de ejemplo

```
(define tabla '((A . titanio) (B . cobre) (C . litio)))

(traducir-codigos tabla '(B X A C X))
> '(cobre titanio litio)

(traducir-codigos '((Z . panel-solar) (Y . escaner))
'(Y Y X Z))
> '(escaner escaner panel-solar)

(traducir-codigos '() '(A B C))
> '()
```

2.4. P4) Corte de potencia perezoso (vectores, *delay*/*force*)

El sistema de autopiloto combina el empuje de los *thrusters* con pesos de rumbo para verificar si la **potencia parcial** acumulada supera un umbral de seguridad antes de completar el cálculo total. Para evitar trabajo innecesario, la evaluación debe ser **perezosa**: sólo se computan los términos requeridos hasta detectar el cruce del umbral.

Objetivo Dadas las entradas $v_1, v_2 \in \mathbb{R}^n$ (vectores de igual longitud) y un umbral $\tau \in \mathbb{R}$, definir para cada $k \in \{1, \ldots, n\}$ la suma parcial

$$S_k \;=\; \sum_{i=1}^{k} v_1[i] \cdot v_2[i].$$

La función debe retornar el índice con base 1

$$k^* \;=\; \text{mín}\,\{\, k \in \{1, \ldots, n\} \mid S_k > \tau \,\},$$

si tal k^* existe; en caso contrario, retornar 0. Además, se exige **evaluación perezosa**: cada término $v_1[i] \cdot v_2[i]$ debe encapsularse con `delay` y forzarse con `force` únicamente al calcular S_k; el algoritmo debe detenerse en el primer k que cumpla $S_k > \tau$, sin evaluar términos posteriores.

Firma

```
(indice-corte v1 v2 umbral)
```

Retorna entero >= 0, es 0 si no supera el umbral.

Parámetros

- `v1`, `v2`: vectores numéricos (#(...)) de igual longitud.
- `umbral`: número real (comparación estricta "`>`").

Comportamiento esperado

- Se recorre `v1` y `v2` en paralelo, acumulando el producto punto parcial.
- Al detectar que la suma parcial > `umbral`, se detiene y retorna el **índice comenzando desde 1** del término que produjo el cruce.
- Si al finalizar no se supera el umbral, retorna `0`.
- La implementación debe envolver cada término en una **promesa** y **forzarla** sólo al utilizarla, evitando evaluar términos posteriores al corte.

Restricciones

- **Obligatorio**: uso explícito de `delay` y `force` para lograr evaluación perezosa.
- **Permitido**: función `lambda`, `cond`, `let/let*`, `vector?`, `vector-length`, `vector-ref`, operaciones aritméticas.
- **Prohibido**: `vector-set!`, materializar listas intermedias innecesarias, `apply`, `map`/`filter`/`fold*`, mutación global.

Casos de ejemplo

```
; suma parcial: 30 > 25 -> corta en el primero
(indice-corte '#(10 0 0 0) '#(3 9 9 9) 25)
> 1

; parciales: 1,2,3,4,5,105 -> corta en el sexto
(indice-corte '#(1 1 1 1 1 100 1 1) '#(1 1 1 1 1 1 1 1) 50)
> 6

; parciales: 5, 10, 15 == umbral (no lo supera)
(indice-corte '#(5 5 5) '#(1 1 1) 15)
> 0

; total 6 < 10 (no lo supera)
(indice-corte '#(1 2 3) '#(1 1 1) 10)
> 0
```

3. Sobre la entrega

- El código debe venir indentado y ordenado.
- Se debe entregar los siguientes archivos:
  - P1.rkt, P2.rkt, P3.rkt, P4.rkt
  - README.txt

▶ **El Uso de IA está prohibido**. Si se detecta, la tarea recibirá nota 0 sin derecho a apelación. (aplica para la totalidad de la entrega)

- Si no existe orden en el código habrá descuento.
- Todas las funciones deben ir comentadas con el siguiente formato:
  ”’
  Parametro a: descripción del parámetro a
  Parametro b : descripción del parámetro b
  Breve descripción de la función y lo que retorna retorno
  ”’
  Se harán descuentos por funcion no comentada
- Se debe programar siguiendo el paradigma de la programación funcional, no realizar códigos que siguen el paradigma imperativo. Por ejemplo, se prohíbe el uso de for-each. Para implementar las funciones utilice DrRacket. https://racket-lang.org/download/
- Todo código debe contener al principio #**lang scheme**
- Todos los archivos deben ser de la extensión **.rkt**
- Pueden crear funciones auxiliaresque no estén especificadas para utilizar en los problemas planteados, pero solo se revisará que la función pedida funcione y el problema esté resuelto con las características funcionales planteada en el enunciado.

- La entrega debe realizarse en un archivo comprimido en tar.gz y debe llevar el nombre: Tarea4LP_RolAlumno.tar.gz.
  Ej.: Tarea4LP_202473000-k.tar.gz.
- El archivo README.txt debe contener el nombre y rol del alumno e instrucciones detalladas para la correcta utilización de su programa.
- La entrega será vía aula y el plazo máximo de entrega es hasta el día **Miércoles 05 de noviembre, 23:59 horas, vía aula.**
- Por cada día de atraso se descontarán 20 puntos (10 puntos dentro de la primera hora)
- Las copias y Uso de código generado por IA serán evaluadas con nota 0 y se informarán a las respectivas autoridades.
- Solo se pueden realizar consultas respecto a la tarea hasta 2 días antes de la entrega.
- Las consultas del enunciado se debe realizarse mediante el foro de la tarea disponible en AULA.

4. Calificación

4.1. Entrega

Para la calificación de su tarea, debe realizar una entrega con requerimientos mínimos que otorgarán 30 pts base, luego se entregará puntaje dependiendo de los otros requerimientos que llegue a cumplir.

4.1.1. Entrega Mínima (total 30 pts)

- Implementar correctamente las dos funciones recursivas de P1 (`calibra-simple` y `calibra-cola`). (30 pts)

Nota: Las funciones deben devolver las respuestas correctas en el formato que se indica en los ejemplos. **De obtener los 30 puntos de la entrega mínima, se procederá a la revisión de las siguientes secciones de la tarea.**

4.1.2. Asignación de puntajes general (total 70 pts)

- Implementación de funciones para la correcta resolución de los casos a evaluar (Total 70 pts)
  - Implementar correctamente la función P2 (`radar-detecta`). (20 pts)
  - Implementar correctamente la función P3 (`traducir-codigos`). (20 pts)
  - Implementar correctamente la función P4 (`indice-corte`). (30 pts)

Para todas las funciones, existe puntaje parcial acorde a los casos de prueba que resuelvan.

4.2. Descuentos

- Código no compila (-100 pts)
- Uso de funciones prohibidas (cada función afectada recibe 0 pts)
- Falta de comentarios sobre cada `define` (–5 pts c/u, máx. –20 pts)
- Falta de README (–20 pts)
- Falta de información obligatoria en el README (nombre, rol, instrucciones) (–5 pts c/u)
- Falta de orden o mala indentación en el código (–5 a –20 pts según gravedad)
- Mal nombre en los archivos entregados (–5 pts c/u; –10 pts si es el `.tar.gz`)
- Uso de código generado por IA, nota máxima 30 y en casos graves implicará un 0 y se le informará a las respectivas autoridades.
- Entrega tardía (–10 pts si es dentro de la primera hora; –20 pts por cada día o fracción de retraso)
- Uso de código y/o texto generado por IA implicará un 0 y se le informará a las respectivas autoridades.
- Entrega tardía (–10 pts si es dentro de la primera hora; –20 pts por cada día o fracción de retraso)
- En caso de existir nota negativa, esta será remplazada por un 0.