Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información.
CI–3661 – Laboratorio de Lenguajes de Programación I.
Septiembre–Diciembre 2025

Tarea 1
Práctica de Haskell
(10 pts)

Problema 1: Palíndromo (2 pts)

Implementa una función que determine si una cadena es un palíndromo (se lee igual al derecho y al revés). La función debe devolver un simple valor booleano (`True` o `False`). **Implementa esto usando recursión explícita.**

Firma de tipo:

```haskell
esPalindromo :: String -> Bool
```

Problema 2: Producto de Elementos Pares en una Lista (2 pts)

Implementa una función **usando recursión explícita (sin usar `map`, `fold` o `filter` inicialmente)** que calcule el producto de todos los elementos en una lista de números enteros, pero solo incluyendo aquellos elementos que son **pares**. La función debe devolver 1 si la lista está vacía o no contiene números pares.

Firma de tipo:

```haskell
productoParesRec :: [Integer] -> Integer
```

Problema 3: Parseo Condicional con `Either` (2 pts)

Implementa una función que tome una lista de cadenas. Para cada cadena, intenta convertirla a un número entero (`Int`). Si la conversión es **exitosa**, devuelve ese número entero. Si la conversión **falla** (porque la cadena no es puramente numérica), devuelve la cadena original transformada a **mayúsculas**. La función debe devolver una lista de resultados, donde cada elemento es un `Either` que contiene `Left String` si no fue numérica o `Right Int` si fue exitosa.

Firma de tipo:

```haskell
parsearCondicional :: [String] -> [Either String Int]
```

Problema 4: Suma Acumulada Condicional (2 pts)

Implementa una función que reciba una lista de números de punto flotante y un umbral (`Float`). La función debe **filtrar** la lista, manteniendo solo los números que son **mayores** que el umbral, y luego calcular la **suma** de los números filtrados utilizando una operación de **plegado** (`fold`).

Firma de tipo:

```
sumaAcumuladaCondicional :: Float -> [Float] -> Float
```

Problema 5: Generación de Coordenadas Impares (2 pts)

Implementa una función que tome un número entero positivo N y genere una lista de **pares de coordenadas** (x, y) donde x e y son números enteros en el rango $[1, N]$ y la suma de x e y $(x + y)$ es un número **impar**. **Utiliza Listas por Comprensión** para la implementación.

Firma de tipo:

```
coordenadasImpares :: Int -> [(Int, Int)]
```

Problema 6: Descomposición Segura de Lista (2 pts)

Implementa una función **recursiva** que divida una lista en su cabeza (`head`) y el resto (`tail`), pero lo haga de manera segura utilizando el tipo `Maybe`. Si la lista está vacía, devuelve `Nothing`. Si la lista no está vacía, devuelve `Just` con una tupla que contiene el primer elemento y el resto de la lista.

Firma de tipo:

```
descomponerListaSegura :: [a] -> Maybe (a, [a])
```

Entrega

Se les suministrará un archivo 'tarea1.hs' para que usen como base para la implementación de las funciones aquí propuestas.

Esta asignación es de carácter individual y debe ser subida a *GitHub*. Su repositorio debe contener un archivo `README.md` identificado con el nombre y número de carnet del estudiante. La fecha límite de entrega es el **viernes 17 de octubre de 2025 a las 11:59 pm**.
