Proyecto 1 – Aplicación de Programación Funcional en Haskell

Fecha de entrega: 20/02

1. Descripción general

Este proyecto tiene como objetivo integrar los conceptos fundamentales de la **programación funcional** vistos hasta el momento mediante el desarrollo de un programa pequeño, bien tipado y correctamente estructurado en **Haskell**.

El énfasis del proyecto **no está en la complejidad algorítmica**, sino en:

- Modelado correcto de datos
- Uso disciplinado de funciones puras
- Razonamiento declarativo
- Separación clara entre lógica y efectos
- Aplicación explícita de recursividad y pattern matching
- Manejo básico de errores con `Maybe`

2. Objetivo del proyecto

Al finalizar este proyecto, el estudiante será capaz de:

- Diseñar un modelo de datos usando **tipos algebraicos**
- Implementar la lógica principal mediante **funciones puras**
- Resolver problemas usando **recursividad estructural**
- Aplicar **pattern matching** de forma correcta y exhaustiva
- Manejar errores simples sin excepciones, usando `Maybe`
- Integrar la lógica funcional con una interacción mínima por consola o archivo
- Explicar el comportamiento del programa desde una perspectiva **declarativa**

3. Requisitos mínimos (obligatorios)

El proyecto **debe cumplir todos** los siguientes puntos:

3.1 Tipos de datos

- Definir **al menos un tipo de dato algebraico propio**
- Usar el tipo en la lógica principal del programa

Ejemplo:

```
data SensorStatus = Active | Inactive | Error
```

3.2 Funciones puras

- La lógica principal debe estar implementada con **funciones puras**
- No debe haber IO mezclado con la lógica

Ejemplo correcto:

```
isOperational :: SensorStatus -> Bool
isOperational Active = True
isOperational _      = False
```

3.3 Uso explícito de recursividad

- Debe haber al menos **una función recursiva definida por el estudiante**
- No se aceptan soluciones que dependan exclusivamente de `map`, `filter`, etc.

Ejemplo:

```
countActive :: [SensorStatus] -> Int
countActive [] = 0
countActive (Active:xs) = 1 + countActive xs
countActive (_:xs) = countActive xs
```

3.4 Pattern matching

- Uso explícito de pattern matching sobre:
    - Tipos definidos por el usuario
    - Listas o estructuras recursivas
- Debe ser **exhaustivo**

Ejemplo:

```
describeStatus :: SensorStatus -> String
describeStatus Active   = "Sensor activo"
describeStatus Inactive = "Sensor inactivo"
describeStatus Error    = "Sensor en error"
```

3.5 Manejo básico de errores con Maybe

- Al menos una función debe devolver Maybe
- No usar valores mágicos (-1, "error", etc.)

Ejemplo:

```
safeDivide :: Double -> Double -> Maybe Double
safeDivide _ 0 = Nothing
safeDivide x y = Just (x / y)
```

3.6 Interacción mínima (IO)

- El programa debe:
    - Leer un valor por consola **o**
    - Escribir un resultado **o**
    - Leer/escribir un archivo simple
- El `main` debe limitarse a **coordinar IO**

Ejemplo:

```
main :: IO ()
main = do
  putStrLn "Ingrese un número:"
  input <- getLine
  print (safeDivide 10 (read input))
```

4. Restricciones importantes

No usar:

- Programación imperativa simulada
- Variables mutables
- Excepciones
- Librerías avanzadas no vistas en clase

Sí se espera:

- Código legible
- Tipos explícitos
- Comentarios explicativos
- Enfoque declarativo

5. Entregables

- **Un archivo .hs**
- Nombre del archivo:
- Proyecto1_NombreApellido_NombreApellido.hs
- Comentarios explicando:
    - El modelo de datos
    - Las funciones principales
    - Las decisiones de diseño

6. Criterios de evaluación (resumen)

Criterio	Peso
Modelado de datos (tipos algebraicos)	20%
Funciones puras y separación de IO	20%
Recursividad y pattern matching	20%
Manejo de errores con Maybe	15%
Correctitud y claridad del código	15%
Documentación y estilo	10%

7. Ejemplos de temas permitidos

Algunos ejemplos (no obligatorios):

- Simulación simple de sensores
- Gestión de estados de un sistema
- Evaluación de reglas o decisiones
- Procesamiento simbólico sencillo
- Validación de datos
- Pequeños sistemas de clasificación

Ideas de proyecto sugeridas

Evaluador de expresiones aritméticas (recomendado)

Modelo funcional de expresiones y su evaluación recursiva.

- Excelente para tipos algebraicos
- Directamente conectado con reescritura simbólica

Ejemplo:

- Entrada: expresión construida con tipos
- Salida: Just resultado o Nothing

Simulación funcional básica (cola de espera)

Simulación donde el estado es un valor inmutable transformado por eventos.

Ejemplo:

- Entrada: lista de eventos
- Salida: estado final de la cola

Mini sistema de gestión funcional (tareas)

Gestión simple de tareas usando listas y tipos algebraicos.

Ejemplo:

- Entrada: acción + lista de tareas
- Salida: lista modificada o Nothing

Analizador simple de texto

Clasificación básica de palabras y números.

- Sin librerías externas
- Sin parsing complejo

Propuestas NO válidas

- Programas imperativos con bucles
- Uso extensivo de estado mutable
- Interfaces gráficas
- Proyectos sin tipos algebraicos propios
- Código donde la lógica está en `main`

8. Relación con el cálculo lambda

Este proyecto se fundamenta en los conceptos vistos en clase:

- **Funciones como abstracciones**
- **Aplicación como sustitución**
- **Evaluación por reducción**
- **Forma normal**
- **Ausencia de estado**
- **Razonamiento por reescritura**

El código debe poder explicarse como una **secuencia de transformaciones de expresiones**, no como una secuencia de instrucciones.

9. Referencias bibliográficas recomendadas

Principales

- Hutton, G. *Programming in Haskell*. Cambridge University Press.
- Pierce, B. *Types and Programming Languages*. MIT Press.
- Bird, R. *Introduction to Functional Programming using Haskell*.

Complementarias

- Lipovača, M. *Learn You a Haskell for Great Good!*
- Thompson, S. *Haskell: The Craft of Functional Programming*
- Barendregt, H. *The Lambda Calculus: Its Syntax and Semantics*

Recursos en línea

- https://wiki.haskell.org
- https://hoogle.haskell.org
- https://learnyouahaskell.com