Compiladores, 2024-1
Práctica 1: Introducción a Haskell

Manuel Soto Romero
manu@ciencias.unam.mx

Javier Enríquez Mendoza
javiem@ciencias.unam.mx

Pedro Ulises Cervantes González
confundeme@ciencias.unam.mx

Braulio Aaron Santiago Carrillo
braulioa124@ciencias.unam.mx

Facultad de Ciencias, UNAM
Fecha de entrega: 01 de septiembre de 2023

1 Introducción

Haskell es un lenguaje de programación de propósito general con las siguientes características:

- Puramente funcional. Las funciones siempre producen la misma salida para la misma entrada, por lo que su comportamiento es predecible.
- Perezoso. Esta evaluación proporciona resultados según sea necesario, lo que permite, entre otras cosas, implementar estructuras de datos infinitas.
- Tipos de datos algebraicos. El sistema de tipos del lenguaje proporciona herramientas para definir tipos de datos propios con un alto grado de comodidad y seguridad.
- Tipados fuerte y estático. El sistema de tipos detecta muchos errores antes de que se ejecute el programa.
- Inferencia de tipos. La inferencia automática de tipos hace que el código sea más corto y permite a lxs desarrolladorxs ser más productivxs.
- Clases de tipos. La categorización de tipos en clases proporciona una sobrecarga segura de tipos.

Este será el lenguaje que usaremos durante el curso, por lo que el objetivo de esta práctica es conocer su funcionalidad para sacarle el mayor provecho. La práctica consiste en definir las siguientes funciones

2 pts Define la función **groupAnagrams** tal que recibe una lista de **String** y devuelve una lista con los anagramas agrupados. Un anagrama es una palabra o frase formada al reorganizar las letras de otra palabra o frase, utilizando todas las letras originales exactamente una vez.

```
groupAnagrams :: [String] -> [[String]]

    {- Ejemplo -}
    > groupAnagrams ["eat","tea","tan","ate","nat","bat"]
    > [["eat","tea","ate"],["tan","nat"],["bat"]]

    > groupAnagrams ["hello","","world","wldro","hlloe","a","aa"]
    > [["hello","hlloe"],[""],["world","wldro"],["a"],["aa"]]
```

2 pts Define la función **subsets** tal que recibe una lista de elementos únicos y devuelve el conjunto potencia.

```
subsets :: [a] -> [[a]]

    {- Ejemplo -}
```

```
> subsets [1,2,3]
> [[],[3],[2],[2,3],[1],[1,3],[1,2],[1,2,3]]

> subsets ['a','b','c','d']
> ["","d","c","cd","b","bd","bc","bcd","a","ad","ac","acd","ab","abd","abc","abcd"]
```

2 pts El elemento mayoritario es el elemento que aparece más de $\lfloor n/2 \rfloor$ veces, donde n es la longitud de la lista. Define la función **majorityElem** tal que recibe una lista y devuelve su elemento mayoritario.

La solución debe ser de complejidad $O(n)$ en tiempo y $O(1)$ en el espacio.

```
majorityElem :: Eq a => [a] -> a

    { - Ejemplo -}
    > majorityElem [3,2,3]
    > 3

    > majorityElem [2,2,1,1,1,2,2]
    > 2
```

2 pts Define la función **coins** tal que recibe una lista de monedas de diferentes denominaciones y una cantidad total de dinero, y devuelve si es posible completar la cantidad usando únicamente ese tipo de monedas.

```
coins ::  [Int] -> Int -> Bool

    { - Ejemplo -}
    > coins [2,5] 8
    > True

    > coins [2,4,6] 21
    > False
```

Considera la siguiente definición de árbol binario:

```
data BST a = Empty | Node a (BST a) (BST a) deriving Show
```

1 pto Define la función **isBST** tal que recibe un árbol binario y devuelve si es un árbol de búsqueda binario válido. Un **BST** válido se define de la siguiente manera:

(a) El subárbol izquierdo contiene solo valores menores que la raíz.

(b) El subárbol derecho contiene solo valores mayores que la raíz.

(c) Ambos subárboles deben ser árboles de búsqueda binarios.

```
isBST :: BST Int -> Bool

    { - Ejemplo -}
    > isBST (Node 3 (Node 1 Empty (Node 2 Empty Empty)) (Node 4 Empty Empty))
    > True

    > isBST (Node 3 (Node 1 Empty (Node 3 Empty Empty)) (Node 4 Empty Empty))
    > False
```

1 pto Define la función **kthElem** tal que recibe un árbol de búsqueda binaria y un número entero k, y devuelve el k-ésimo valor más pequeño.

```
kthElem :: BST a -> Int -> a

    { - Ejemplo -}
    > kthElem (Node 3 (Node 1 Empty (Node 2 Empty Empty)) (Node 4 Empty Empty)) 2
    > 2

    > kthElem (Node 5 (Node 3 (Node 2 (Node 1 Empty Empty) Empty) (Node 4 Empty Empty))
        (Node 6 Empty Empty)) 4
    > 4
```

2 Entrega

- La practica será entregada en equipos de máximo 5 integrantes.
- La entrega será por Google Classroom.
- Únicamente un miembro del equipo sube la solución de la practica. El resto debe indicar los integrantes de su equipo en un comentario privado.
- Únicamente anexar el archivo con extensión `.hs` con la solución. El nombre del archivo debe ser el nombre del integrante que subió el archivo empezando por apellidos.
- Deberás entregar el ejercicio a más tardar a las 23:59 del día indicado. Después de esta hora, el ayudante rechazará el ejercicio.