Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación
Lenguajes de Programación (6204)
Semestre 1-2025

Proyecto #1 – Haskell
Draft Beers

En una fiesta, se cuenta con 3 barriles de cerveza de diferentes capacidades. Los barriles están conectados entre sí mediante tubos que permiten transferir cerveza de uno a otro. Además, cada barril tiene una salida que permite servir cerveza directamente en vasos.

Su objetivo como especialista en programación funcional es diseñar un programa que permita determinar cuántos litros de cerveza deben agregarse entre los barriles para servir exactamente **n** vasos de cerveza desde cualquiera de las salidas. El programa debe considerar las siguientes restricciones:

1. Cada vaso corresponde a un litro de cerveza.
2. Cada barril tiene una capacidad máxima y una cantidad inicial de cerveza.
3. Los barriles están conectados de la siguiente manera: A <-> B <-> C.
4. Se puede agregar cerveza desde los barriles A y C.
5. Los vasos se sirven desde cualquier barril que cumpla con la cantidad de cerveza solicitada.
6. Las transferencias entre barriles deben respetar las capacidades máximas de los mismos.
7. Se debe minimizar la cantidad de cerveza a agregar (mejor solución).

Ejemplo:

- Barril A: Capacidad 1L, contiene 1L.
- Barril B: Capacidad 7L, contiene 2L.
- Barril C: Capacidad 4L, contiene 3L.

Objetivo: Servir exactamente 6 vasos de cerveza desde un solo barril (1L por vaso).

Resultado esperado: agregar 4L de cerveza desde el Barril A para servir los 5 vasos de cerveza desde el Barril B.

Parte 1 – Inicialización de barriles

La representación de un barril se hará mediante un par (capacidad máxima, cantidad actual).

```
type Barrel = (Int, Int)
```

Escriba la función **initialBarrels** para recibir los valores de cada barril y devuelva una terna de *Barrel*.

```
initialBarrels :: Barrel -> Barrel -> Barrel  -> (Barrel, Barrel, Barrel)
```

Caso de prueba:

1) **initialBarrels (10, 10) (7, 0) (3, 0)**

Retorno: ((10, 0), (7, 0), (3, 0))

Parte 2 – Existe solución

Verificar si el estado actual de los barriles garantiza una posible solución. Implemente la función **iSolution**, que reciba como entrada los tres barriles, el objetivo de cervezas y devuelva un booleano indicando si hay solución.

iSolution:: (Barrel, Barrel, Barrel) -> Int -> Bool

Casos de prueba:

1) **iSolution ((10, 1), (7, 3), (2, 2)) 2**

Retorno: True

2) **iSolution ((3, 3), (5, 2), (7, 1)) 5**

Retorno: False

Parte 3 – Añadir cerveza

Implemente la lógica para añadir cerveza a un barril, respetando las capacidades máximas de los barriles y las cantidades actuales de cerveza. Si el barril sobrepasa su capacidad debe devolverse la cantidad de cerveza a transferir al barril vecino.

addBeer :: Int -> Barrel -> (Barrel, Int)

Casos de prueba:

1) **addBeer 5 (20, 15)**

Retorno: ((20, 20), 0)

2) **addBeer 5 (4, 3)**

Retorno: ((4, 4), 4)

Parte 4 – Mejor solución

Implementar la función **findBestSolution** para determinar la cantidad de cerveza óptima que debe agregarse en los barriles, con el fin de alcanzar una cantidad específica de vasos de cerveza en uno de los tres barriles. La función recibe como entrada un entero **n**, que representa la cantidad total de cerveza requerida, y los tres barriles. La función debe devolver una tupla donde la primera posición indica los litros de cervezas requeridos, y la segunda muestra el estado final de los tres barriles antes de servir las cervezas.

findBestSolution :: Int -> (Barrel, Barrel, Barrel) -> (Int, (Barrel, Barrel, Barrel))

Casos de prueba:

1) **findBestSolution 4 ((10, 6), (7, 0), (4, 0))**

Retorno: (0, ((10, 6), (7, 0), (4, 0)))

2) **findBestSolution 5 ((10, 3), (7, 0), (4, 0))**

Retorno: (2, ((10, 5), (7, 0), (4, 0)))

3) **findBestSolution 6 ((10, 2), (7, 5), (4, 2))**

Retorno: (3, ((10, 2), (7, 6), (4, 4)))

Notas:

- Reutilizar las funciones **iSolution** y **addBeer**.
- Si ocurre un desborde en el barril B, la cantidad de cerveza a transferir debe ir al barril vecino que tenga menor cantidad.
- Puede llegar a perderse cerveza si la cantidad total de cerveza excede la capacidad total de los barriles.

Puntuación:

- Parte 1, 1 pto.
- Parte 2 y 3, 2 ptos c/u.
- Parte 4, 14 ptos.
- Documentación o Readme, 1 pto.

Consideraciones para la entrega

- Realice las validaciones que considere necesarias.
- Sólo se pueden utilizar operaciones del Prelude de Haskell.
- Las funciones solicitadas deben estar definidas dentro de un único archivo llamado **`Proyecto1.hs`** y las funciones deben tener el mismo nombre y número de argumentos descritos en el enunciado.
- Las respuestas de las funciones deben ser iguales a las solicitadas en el enunciado, de no ser así el script de pruebas arrojará error y se calificará como incorrecto.
- Utilice comentarios en Haskell para separar las soluciones de las preguntas y documentar lo que considere necesario. En Haskell hay dos tipos de comentarios: línea (`--`) y bloque (`{-` y `-}`). Incluya un comentario de encabezado con sus datos en el archivo.
- No está permitido utilizar soluciones de terceros (incluyendo soluciones generadas por LLMs). Cualquier material consultado para ideas de solución o documentación debe ser referenciado, indicando para qué sirvió. Esto lo pueden documentar en un archivo en formato *markdown* (**`README.md`**) o directamente como comentarios en **`Proyecto1.hs`**. No están permitidos documentos en ningún otro formato.
- El proyecto es **estrictamente en parejas** y la fecha de entrega será el **domingo 25/05/2025**, hasta las 11:59 pm (VET).
- El proyecto se debe adjuntar por e-mail a la dirección `ldpucv@gmail.com` usando como asunto: `[Proyecto Haskell] <Apellido(s)> <Nombre(s)>, <Cédula1> - <Apellido(s)> <Nombre(s)>, <Cédula2>`.

José Yvimas, Mayo 2025