Universidad Simón Bolívar
Departamento de Computación
y Tecnología de la Información
CI-3661: Laboratorio de
Lenguajes de Programación

Prof. Fernando Torre Mora

Proyecto 1 Septiembre – Diciembre 2018

El sabio del laberinto

1 Introducción

En las profundidades del bosque de los mil y un monads, en la misteriosa cuna del gran río Curry, hay un gran laberinto custodiado por un sabio. Dicho sabio es un experto en laberintos acíclicos, y con una mirada ya conoce todos los caminos y recovecos de ellos.

El sabio permite a los viajeros preguntarle por su recorrido planeado, y les indica qué encontrarán. Si el recorrido no llega hasta un camino sin salida o a un tesoro, el sabio permite que indiquen cómo procederían hasta llegar a un punto que sí lo sea.

Al salir un viajero, el sabio le pregunta qué encontró, para actualizar lo que sabe del laberinto. Así se mantiene al día con cuáles tesoros han sido reclamados, cuáles tesoros nuevos han sido dejados, paredes derribadas y derrumbes. De vez en cuando, un viajero le cuenta de otro laberinto, y con su descripción el sabio ya puede informarles a otros viajeros sobre los otros grandes laberintos que encontrarán.

Fue así como el sabio supo del gran laberinto llamado Universidad Simón Bolívar, y del tesoro más grande en su interior: los estudiantes del Laboratorio de Lenguajes capaces de crear un programa que replique sus capacidades.

2 Requerimientos del programa

Cada laberinto se compone de trifurcaciones, es decir, divisiones de la vía en tres en las que se puede

- Voltear a la izquierda
- Voltear a la derecha
- Seguir recto

El usuario no puede especificar ninguna otra opción (por ejemplo, no puede dar media vuelta para devolverse por donde vino) ya que el sabio no presta atención a viajeros que tratan de pasarse de listos. Si una división carece de una de las opciones (por ejemplo, en una intersección en T no se puede seguir recto) y el usuario especifica esa opción faltante, el sabio pone ojos en blanco e ignora este paso del recorrido. Si el recorrido lo llevaría a un camino sin salida y el recorrido tiene aún más pasos, el sabio se ríe para sus adentros, perimitiendo que el visitante siga indicando su recorrido planeado pero sin prestarle atención ya que no tendría por dónde seguir. Si hay una división que regresaría al viajero a un punto ya visitado, el sabio lo ignora, ya que prefiere concebir los laberintos en términos clásicos.

El sabio siempre permite que los visitantes den todo su recorrido antes de responder, y si no los conduce hasta un tesoro o un camino sin salida, permite que den una lista de pasos adicionales.

El sabio distingue entre los diferentes tesoros. Si el recorrido pasa sobre un tesoro pero no termina en él, el sabio supone que el tesoro será ignorado por los viajeros.

2.1 El tipo de datos "Laberinto"

Para implementar el programa, debe especificar un tipo de datos que almacene los conocimientos del sabio sobre el laberinto. Específicamente, debe tener

- `Trifurcacion`, que deben incluir
  - Un `Maybe` laberinto indicando lo alcanzable al voltear a la **derecha**, o `Nothing` si no es posible voltear a la derecha
  - Un `Maybe` laberinto indicando lo alcanzable al voltear a la **izquierda**, o `Nothing` si no es posible voltear a la izquierda
  - Un `Maybe` laberinto indicando lo alcanzable al **seguir recto**, o `Nothing` si no es posible seguir recto
- `Tesoro`, que deben incluir
  - Un `String` con la descripción del tesoro
  - Un `Maybe` laberinto indicando qué encontrarán si pasan el tesoro por alto (siguen recto)

Adicionalmente a especificar el laberinto, deben proveer las siguientes funciones para poder manejarlo

- Funciones de Construcción
  - Una función que retorne un **camino sin salida** (es decir, una `Trifurcacion` donde todos los caminos conducen a `Nothing`)
  - Una función que reciba un `String` con la **descripción de un tesoro** y un **laberinto** indicando qué encontrarán si pasan por alto el tesoro, y retorne el `Tesoro`.
  - Una función que reciba una **`Trifurcacion`**, un **laberinto** y un **indicador** de cuál camino los relaciona (izquierda, derecha, recto), y retorne una `Trifurcacion` en la que se indique que dicho camino conduce a dicho laberinto.
- Funciones de acceso
  - Una función que reciba un **laberinto** y una **ruta** y retorne el **laberinto** que comienza en el punto al que conduce esa ruta
  - Una función que reciba un **laberinto** y retorne el **laberinto** que comienza al voltear a la **izquierda**
  - Una función que reciba un **laberinto** y retorne el **laberinto** que comienza al voltear a la **derecha**
  - Una función que reciba un **laberinto** y retorne el **laberinto** que comienza al seguir **recto**
- Instancias
  - Una instancia de la clase `Show`, para crear representaciones en forma de `String`. Es importante que la representación escogida sea fácil de leer y convertir nuevamente en un laberinto, más allá de que sea legible para un humano
  - Una instancia de la clase `Read`, para leer representaciones como cadenas de caracteres de un laberinto y construir un nuevo laberinto a partir de dicha información.

El tipo de datos y las funciones asociadas deberán estar contenidas en un módulo de nombre `Laberinto`, plasmadas en un archivo de nombre `Laberinto.hs`. Es importante que todas las definiciones pedidas sean visibles para potenciales importadores de tal módulo. **Importante**: No puede modificar la estructura del laberinto ni la firma de las funciones propuestas.

2.2 Programa Cliente

El programa principal debe permitir interactuar con el usuario, recibiendo rutas e indicando qué se encuentra al seguirlas. Concretamente, el cliente debe implementar las siguientes funciones:

- Una función `main` que recibe argumentos y devuelve un `IO()`. La función debe mantener internamente un laberinto e interactuar con el usuario. La interacción se logra pidiéndole repetidamente que ingrese una opción disponible y luego pasar a ejecutarla. Las diferentes opciones se muestran a continuación
    - **Comenzar a hablar de un laberinto nuevo:** Si esta opción es seleccionada, se reemplaza el laberinto en memoria con un laberinto vacío y se pide una ruta que puede ser seguida en el mismo. Se usa esta ruta para poblar el laberinto inicialmente. La ruta debe suponerse terminada en un camino sin salida y no contener tesoros.
    - **Preguntar ruta:** Si esta opción es seleccionada, se recorre la ruta dada en el laberinto y se indica qué hay en el punto que se alcanza. Si no se alcanza un tesoro o un camino sin salida, la opción es reemplazada por las siguientes dos opciones
        - **Continuar ruta:** Si esta opción es seleccionada, se recorre el laberinto que comienza al final de la ruta anterior usando la ruta especificada por el usuario y se indica qué se alcanza
        - **Preguntar ruta nueva:** Si esta opción es seleccionada, se sigue la ruta desde el inicio del laberinto
    - **Reportar pared abierta:** Si esta opción es seleccionada, se recibe un camino, luego se recorre el camino hasta alcanzar una pared (un `Nothing`). La ruta dada a partir de ese momento se convierte en el laberinto alcanzable por esa dirección.
    - **Reportar derrumbe:** Si esta opción es seleccionada, se recibe un camino y una dirección (izquierda, derecha o recto). Se sigue el laberinto hasta ese punto y se elimina el laberinto alcanzable en la dirección dada.
    - **Reportar tesoro tomado:** Si esta opción es seleccionada, se recibe una ruta. Se reemplaza el tesoro al final de la ruta con el laberinto accesible pasando al tesoro de largo. Si al final de la ruta dada no había un tesoro, el usuario debe recibir un mensaje pertinente y el laberinto no debe ser cambiado
    - **Reportar tesoro hallado:** Si esta opción es seleccionada, se recibe una ruta. Se agrega el tesoro como alcanzable desde la trifurcación inmediatamente anterior, y el laberinto anteriormente alcanzable desde la misma como alcanzable siguiendo recto desde el tesoro. Si ya existe un tesoro en ese punto, el usuario debe recibir un mensaje pertinente
    - **Dar nombre al laberinto:** Si esta opción es seleccionada, se debe pedir un nombre de archivo al usuario, y luego se debe almacenar el laberinto en el archivo suministrado
    - **Hablar de un laberinto de nombre conocido:** Si esta opción es seleccionada, se debe cargar el laberinto en el archivo suministrado

Es altamente recomendable que la implementación de su función `main` esté dividida en diversas otras funciones que se encarguen de cada posible acción, por motivos de modularidad y legibilidad. El cliente debe estar contenido en un módulo de nombre `Sabio`, plasmado en un archivo de nombre `Sabio.hs`.

3 Requerimientos de la entrega

Todos sus códigos deben estar debidamente documentado en formato Haddock. Debe entregar su proyecto en un archivo comprimido (tgz, zip, etc) que contenga únicamente los siguientes elementos:

- Los archivos `Laberinto.hs` y `Sabio.hs` descritos anteriormente
- Un archivo `makefile` para compilar correctamente los códigos fuentes. Debe generarse un ejecutable `Sabio` que no reciba argumentos y ejecute la función `main` del módulo `Sabio`.
- Un archivo `Readme.txt` con los datos de su equipo (integrantes, carnet, etc.) y los detalles que considere relevantes de su implementación que no pertenezcan al Haddock
- Al menos un laberinto de ejemplo. El(los) ejemplo(s) incluido(s) debe(n) poder cargarse sin errores usando la opción "Hablar de un laberinto de nombre conocido" de la función `main`.

El proyecto debe ser subido al Moodle de la materia en la sección marcada como "Proyecto 1" hasta el domingo, 28 de octubre. Sólo deberá efectuar una entrega por grupo.

4 Evaluación

El proyecto tiene una ponderación de 30 puntos. Se asignarán

- 15 puntos por código
  - 2 puntos por su tipo de datos "Laberinto" tal y como se especifica en la sección 2.1
  - 9 puntos por las funciones para manejar el tipo de datos "Laberinto (1 pt c/u)
  - 2 puntos por su función `main`
  - 1 punto por utilizar correctamente el *monad* de `IO` poder guardar en un archivo
  - 1 punto por utilizar correctamente el *monad* de `IO` poder cargar desde un archivo
- 10 puntos por ejecución (1 pt por cada opción del cliente)
- 5 puntos por su documentación de Haddock (½ pt por función)

El programa debe correr sin errores. La documentación de Haddock debe poder generarse sin errores.