Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información.
CI–3661 – Laboratorio de Lenguajes de Programación.
Septiembre–Diciembre 2017

Proyecto I: Haskinator

En las profundidades del bosque de los mil y un monads, en la apartada y escondida cuna del gran río Curry, vive en su choza de piedra el poderoso oráculo *Haskinator*. Con su vasto conocimiento, Haskinator es capaz de adivinar los pensamientos de aquellos que lo visitan, sin más que unas pocas y puntuales preguntas. El gran oráculo se ha hecho conocido y afamado por su gran poder, sorprendiendo y maravillando a cada viajero con la suerte de presenciarlo.

Sin embargo, Haskinator ahora siente que sus poderes se desvanecen en el tiempo y teme defraudar a quienes expectantes desean ser maravillados por él. Velozmente maquina una forma de conservar el don que se le escapa cuando recuerda la historia de uno de sus tantos visitantes. El visitante le había contado sobre un grupo de talentosos estudiantes que aprendían a programar en *Haskell*, el lenguaje de los antiguos dioses y oráculos de eras pasadas.

Usando el poco poder telepático que le quedaba, el gran Haskinator se comunica con los estudiantes y les encomienda la creación de un programa que logre emular sus legendarias capacidades de predicción.

Tipo de Datos: Oráculo

Un `Oraculo` debe representar el conocimiento de Haskinator, por tanto puede ser una `Pregunta` o una `Prediccion`.

- En el caso de las predicciones, debe incluir:
  - Una cadena de caracteres con la predicción en cuestión.
- En el caso de las preguntas, debe incluir:
  - Una cadena de caracteres con la pregunta en cuestión.
  - Un diccionario (no vacío) de opciones, que asocia cadenas de caracteres (las respuestas) con oráculos (el resto del proceso de predicción).

Tipo de Datos: Opciones

Una `Opcion` debe contener información de la opción escogida en la pregunta anterior y cómo continuar con la predicción. Por tanto, el conjunto de opciones viene a ser un diccionario con clave `String` y valor `Oraculo`. PAra esto, puede usarse el tipo de datos `Map`, incluido en la librería `Data.Map`.

```
type Opciones = Map String Oraculo
```

Nota: Es posible que deba importar y utilizar la librería `Data.Map` *de forma* `qualified`*.*

En adición a las definiciones de `Oraculo` y `Opciones`, se deben suplir las siguientes funciones para manipularlas:

- **Funciones de construcción:**
  - `crearOraculo :: String -> Oraculo`
    Recibe una cadena de caracteres y devuelve un oráculo que consiste únicamente de la cadena suministrada como predicción.
- **Funciones de acceso:** *(Nota: Usando buenas técnicas de definición de datos, algunas de estas funciones pueden resultar inmediatas).*
  - `prediccion :: Oraculo -> String`
    Recibe un oráculo y devuelve la cadena de caracteres asociada si el mismo es una predicción (arroja un error de lo contrario).
  - `pregunta :: Oraculo -> String`
    Recibe un oráculo y devuelve la cadena de caracteres asociada si el mismo es una pregunta (arroja un error de lo contrario).
  - `opciones :: Oraculo -> Opciones`
    Recibe un oráculo y devuelve la lista de opciones asociadas si el mismo es una pregunta (arroja un error de lo contrario).
    Nótese que el tipo Map representa diccionarios con clave String y valor
  - `respuesta :: Oraculo -> String -> Oraculo`
    Recibe un oráculo y una cadena de caracteres S, y devuelve el oráculo que corresponde a la respuesta asociada a la opción S; esto, si el mismo es una pregunta (arroja un error de lo contrario).
- **Funciones de modificación:**
  - `ramificar :: [String] -> [Oraculo] -> String -> Oraculo`
    Recibe un lista de cadenas de caracteres (representando opciones a una pregunta), una lista de oráculos y una cadena de caracteres (representando una pregunta). Devuelve un oráculo, de tipo pregunta, con la cadena suministrada y las opciones construidas.
- **Instancias:**
  - Una instancia de la clase `Show`, para crear representaciones como cadenas de caracteres de un `Oraculo` (y, por consiguiente, de una `Opcion`). Es importante que la representación escogida sea fácil de leer y convertir nuevamente en un oráculo, más allá de que sea legible para un humano.
  - Una instancia de la clase `Read`, para leer representaciones como cadenas de caracteres de un `Oraculo` (y, por consiguiente, de una `Opcion`) y construir un nuevo oráculo a partir de dicha información.

El tipo de datos y las funciones asociadas deberán estar contenidas en un módulo de nombre `Oraculo`, plasmadas en un archivo de nombre `Oraculo.hs`. Es importante que todas las definiciones pedidas sean visibles para potenciales importadores de tal módulo y que únicamente tales definiciones sean visibles (cualquier función auxiliar que implemente, no debe ser visible al exterior del módulo). *Importante: No puede modificar la estructura del oráculo ni de las funciones propuestas.*

Cliente: Haskinator

El programa principal debe poder interactuar con el usuario, planteando preguntas y proponiendo predicciones. Concretamente, este cliente debe implementar la siguiente función:

- `main :: IO()`

  Recibe argumentos y devuelve un `IO ()` (equivalente a un `void` en C). La función debe mantener internamente un oráculo e interactuar con el usuario de la siguiente forma: Debe pedir al usuario repetidamente que ingrese una opción de entre las disponibles y luego pasar a ejecutarla. Las diferentes opciones se muestran a continuación.

  - **Crear un oráculo nuevo:** Si esta opción es seleccionada, se debe pedir al usuario una predicción y almacenar la misma como la única predicción del oráculo.
  - **Predecir:** Si esta opción es seleccionada, se comienza el proceso de predicción:
    - Si el oráculo es una pregunta, se plantea dicha pregunta al usuario y se le presenta el conjunto de respuestas posibles a la misma.
    - Tomando en cuenta la respuesta del usuario, que puede ser únicamente una de las respuestas presentadas o *ninguna*, se navega al sub–oráculo correspondiente.
      - Si el usuario responde *ninguna*, debe pedírsele al usuario la opciónque él esperaba y la respuesta correcta.
    - Al alcanzar una predicción (o si el oráculo inicial era una predicción) se le propone la misma al usuario.
    - El usuario puede entonces decidir si la predicción es acertada. De serlo, se termina la acción. De no serlo, debe pedírsele al usuario:
      - La respuesta correcta.
      - Una pregunta que la distinga de la predicción hecha.
      - La opción que lleva a la respuesta deseada.
      - La opción que corresponde a la respuesta incorrecta (la que arrojó Haskinator).

      Usando esta nueva información, debe incorporarse la nueva pregunta al oráculo.
  - **Persistir:** Si esta opción es seleccionada, se debe pedir un nombre de archivo al usuario y luego se debe almacenar la información del oráculo construido en el archivo suministrado.
  - **Cargar:** Si esta opción es seleccionada, se debe pedir un nombre de archivo al usuario y luego se debe cargarar la información al oráculo desde el archivo suministrado.
  - **Consultar pregunta crucial:** Si esta opción es seleccionada, se deben pedir dos cadenas de caracteres al usuario.
    - Si alguna de las dos no tiene una predicción correspondiente en el oráculo, entonces la consulta es inválida.
    - Si ambas se encuentran como predicciones en el oráculo, se debe imprimir la pregunta crucial que llevaría a decidir eventualmente por una predicción o la otra (si se analiza el oráculo como un árbol, vendría a ser el ancestro en común más bajo), además de la opción correspondiente para cada una de las predicciones involucradas.
  - **Salir:** Permite salir del menú de opciones y terminar la ejecución del programa.

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 `Haskinator`, plasmado en un archivo de nombre `Haskinator.hs`. Es importante que únicamente la función `main` sea visibles (cualquier función auxiliar que implemente, no debe ser visible al exterior del módulo). *Importante: No puede modificar la estructura de la función propuesta.*

Ejemplos de interacciones con Haskinator

Si se tiene un árbol de la siguiente forma:

```
Es un lenguaje de programación?
  - Si: A qué paradigma pertenece?
    - Imperativo: A quién pertenece el lenguaje?
      - Oracle: Java
      - Microsoft: C#
    - Funcional: Haskell
    - Lógico: Prolog
  - No: HTML
```

a) Primero supongamos que el usuario está pensando en `Haskell`:

Haskinator:	`Es un lenguaje de programación?` `Si / No`
Usuario:	`Si`
Haskinator:	`A qué paradigma pertenece?` `Imperativo / Funcional / Lógico`
Usuario:	`Funcional`
`Haskinator:`	`Predicción: Haskell` `Si / No`
Usuario:	`Si`

b) Supongamos ahora que el usuario está pensando en `Go`:

Haskinator:	`Es un lenguaje de programación?` `Si / No`
Usuario:	`Si`
Haskinator:	`A qué paradigma pertenece?` `Imperativo / Funcional / Lógico`
Usuario:	`Imperativo`
Haskinator:	`A quién pertenece el lenguaje?` `Oracle / Microsoft`
Usuario:	`ninguna`
`Haskinator:`	`He fallado! Cuál era la respuesta correcta?`
Usuario:	`Go`
`Haskinator:`	`A quién pertenece el lenguaje?`
Usuario:	`Google`

Luego de esto, el árbol quedaría actualizado de la siguiente forma:

```
Es un lenguaje de programación?
  - Si: A qué paradigma pertenece?
    - Imperativo: A quién pertenece el lenguaje?
      - Oracle: Java
      - Microsoft: C#
      - Google: Go
    - Funcional: Haskell
    - Lógico: Prolog
  - No: HTML
```

c) Supongamos por último que el usuario está pensando en `CSS`:

Haskinator:	`Es un lenguaje de programación?` `Si / No`
Usuario:	`No`
Haskinator:	`Predicción: HTML` `Si / No`
Usuario:	`No`
`Haskinator:`	`He fallado! Cuál era la respuesta correcta?`
Usuario:	`CSS`
`Haskinator:`	`Qué pregunta distingue a CSS de las otras opciones?`
Usuario:	`Qué tipo de lenguaje es?`
`Haskinator:`	`Cuál es la respuesta a "Qué tipo de lenguaje es?" para CSS?`
Usuario:	`De definición de estilo`
`Haskinator:`	`Cuál es la respuesta a "Qué tipo de lenguaje es?" para HTML?`
Usuario:	`De marcado`

Luego de esto, el árbol quedaría actualizado de la siguiente forma:

```
Es un lenguaje de programación?
  - Si: A qué paradigma pertenece?
    - Imperativo: A quién pertenece el lenguaje?
      - Oracle: Java
      - Microsoft: C#
      - Google: Go
    - Funcional: Haskell
    - Lógico: Prolog
  - No: Qué tipo de lenguaje es?
    - De marcado: HTML
    - De definición de estilo: CSS
```

Detalles de la Entrega

La entrega del proyecto consistirá de un único archivo `p1_<carné1>&<carné2>.tar.gz`, donde `<carné1>` y `<carné2>` son los números de carné de los integrantes de su equipo. Por ejemplo, si el equipo está conformado por `00-00000` y `11-11111`, entonces su entrega debe llamarse: `p1_00-00000&11-11111.pdf`.

Tal archivo debe ser un directorio comprimido que contenga únicamente los siguientes elementos:

- Los archivos `Oraculo.hs` y `Haskinator.hs` descritos anteriormente.
- Un archivo `makefile` para compilar correctamente los códigos fuentes. Debe generarse un ejecutable `haskinator` que no reciiba argumentos y ejecute la función `main` del módulo `Haskinator.`
- Un archivo `Readme` con los datos de su equipo (integrantes, carné, etc.) y los detalles relevantes de su implementación.

La tarea deberá ser entregada al prof. Ricardo Monascal *únicamente* a su dirección de correo electrónico oficial: (rmonascal@usb.ve) a más tardar el Miércoles 25 de Octubre, a las 11:59pm. VET.