Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información.
CI–3661 – Laboratorio de Lenguajes de Programación.
Enero–Marzo 2015

# Haskinator

*(25 pts)*

En las profundidades del bosque de los mil y un monads, en la misteriosa cuna del gran río Curry, habita entre paredes 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 precisas preguntas. El misterioso 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 presiente que sus poderes se desvanecen con el tiempo y teme defraudar a quienes expectantes desean ser maravillados por sus predicciones. Con premura comienza a indagar maneras de conservar su preciado don antes que se le escape por completo. Entonces recuerda la historia de uno de sus tantos visitantes. El visitante, agradecido y soprendido con el oráculo, le había contado sobre un grupo de talentosos estudiantes que aprendían a programar en *Haskell*, el lenguaje de dioses y oráculos de eras pasadas.

Usando el poco poder que aún le quedaba, el gran Haskinator se comunica por medio de la telepatía con los profesores del Laboratorio de Lenguajes en la Universidad Simón Bolívar, para que encomienden a sus estudiantes la creación de un programa que logre simular sus afamadas 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 oráculo que corresponda a una respuesta positiva para la pregunta.
  - Un oráculo que corresponda a una respuesta negativa para la pregunta

En adición a la definición de oráculo, se deben suplir las siguientes funciones para manipularlo:

- **Funciones de construccion:**
  - Una función `crearPrediccion` que recibe una cadena de caracteres. Devuelve un oráculo de tipo `Prediccion` con la cadena suministrada como predicción.
  - Una función `crearPregunta` que recibe una cadena de caracteres y dos oráculos. Devuelve un oráculo de tipo `Pregunta` con la cadena suministrada como pregunta, el primer oráculo como respuesta positiva y el segundo como respuesta negativa.

- **Funciones de acceso:** (Usando buenas técnicas de definición de datos, estas funciones pueden resultar inmediatas).
  - Una función `prediccion` que recibe un oráculo y devuelve la cadena de caracteres, siempre y cuando sea una predicción (arroja un error de lo contrario).
  - Una función `pregunta` que recibe un oráculo y devuelve la cadena de caracteres, siempre y cuando sea una pregunta (arroja un error de lo contrario).
  - Una función `positivo` que recibe un oráculo y devuelve el oráculo que corresponde a una respuesta positiva, siempre y cuando sea una pregunta (arroja un error de lo contrario).
  - Una función `negativo` que recibe un oráculo y devuelve el oráculo que corresponde a una respuesta negativa, siempre y cuando sea una pregunta (arroja un error de lo contrario).
- **Funciones de modificación:**
  - Una función `obtenerCadena` que recibe un oráculo y una cadena de caracteres correspondiente a una predicción. Devuelve un valor de tipo `Maybe [(String, Bool)]`. Si la predicción suministrado no pertenece al oráculo, debe retornar `Nothing`. De lo contrario debe retornar `Just lista`, donde `lista` es una lista de tuplas (de cadenas de caracteres y booleanos) que corresponden a todas las preguntas que deben hacerse, a partir de la raíz del oráculo, para alcanzar la predicción suministrada y el valor de verdad (decisión positiva o negativa) de la misma. (*Nota: Si hay varias ocurrencias de una misma predicción, se debe devolver aquella que tome una decisión positiva lo más temprano posible.*)
  - Una función `obtenerEstadisticas` que recibe un oráculo. Devuelve una 3–tupla con los siguientes datos: El mínimo, máximo y promedio de preguntas que el oráculo necesita hacer para llegar a alguna predicción.
- **Instancias:**
  - Una instancia de la clase `Show`, para crear representaciones como cadenas de caracteres del oráculo. 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 oráculo 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 la firma 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 las siguientes funciones:

- Una función `main` que no recibe argumentos y devuelve un `IO ()`. 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 borran todos los datos del oráculo actual quedando el mismo vacío (*Observe que no hay definiciones para oráculos vacíos, por lo que es conveniente que se maneje un valor de tipo* `Maybe Oraculo`*, en vez de un oráculo directamente.*)
  - **Predecir:** Si esta opción es seleccionada, se comienza el proceso de predicción:
    - Si el oráculo es una pregunta, se plantea dicha pregunta al usuario.
    - Tomando en cuenta la respuesta del usuario, que puede ser únicamente *si* o *no*, se navega al sub–oráculo correspondiente.
    - 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 o no. De ser acertada, se termina la acción. De lo contrario, debe pedírsele al usuario que diga cual era la respuesta correcta, además de una pregunta que la distinga de la predicción hecha (que evalúe en verdadero para la respuesta correcta y en falso para la predicción que había realizado). Usando esta información, debe incorporarse la nueva pregunta al oráculo con la respuesta correcta como predicción del lado positivo, y la predicción original como predicción del lado negativo.
  - **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). (*Nota: De existir varias predicciones iguales a las cadenas suministradas se debe tomar, para cada una de ellas, la predicción que se alcance tomando una decisión positiva lo más pronto posible.*)
  - **Consultar estadísticas:** Si esta opción es seleccionada, se debe mostrar al usuario el mínimo, máximo y promedio de preguntas que el oráculo necesita hacer para llegar a alguna predicción. Si el oráculo está vacío, debe reportar al usuario que su consulta es inválida.

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 visible (cualquier función auxiliar que implemente, no debe ser visible al exterior del módulo). *Importante: No puede modificar la firma de la funcion propuesta.*

## Detalles de la Entrega

La entrega del proyecto consistirá de un único archivo `pH<carne1>-<carne2>.tar.gz`, donde `<carne1>` y `<carne2>` sin los carnés de cada uno de los miembros de su equipo, respectivamente. 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 que considere relevantes de su implementación.

El proyecto deberá ser entregado a *todos* los encargados del curso, a su direcciones de correo electrónico oficiales: ( rmonascal@gmail.com, emhn@.usb.ve, targen@gmail.com, jljb1990@gmail.com & dvdalilue@gmail.com ) a más tardar el Martes 3 de Febrero, a las 11:59pm. VET.

*Cualquier parecido con aplicaciones reales o ficticias*
*como* Akinator*, es completamente intencional,*
*pudiendo encontrarse la misma en la siguiente dirección:*
( http://es.akinator.com/ )