Trabajo Práctico Final

Análisis de Lenguajes de Programación

Julio Joaquín Güella
Legajo: G-5061/1
julioguella@hotmail.com

Introducción

RecipeManager es una aplicación que provee una interfaz por consola para gestionar una base de datos referida a los alimentos consumidos en una vivienda, haciendo énfasis en los valores nutricionales de estos.
Los datos a manejar (definidos en el módulo Types.hs) se agrupan principalmente en:

- `Ingredientes` : Elementos empleados para preparar las recetas. Constan de un nombre único, una cantidad (en una unidad arbitraria) y una fecha de vencimiento.
- `Recetas` : Lista de ingredientes y pasos que sirven para elaborar cierta comida. Constan de un nombre único, una lista de nombres de ingredientes con sus respectivas cantidades, la lista de pasos a seguir, y una lista de palabras claves para clasificarlas.
- `Valores nutricionales` : Información propia de cualquier alimento, que permite ver el aporte nutricional de este al ser consumido. Constan del nombre del ingrediente en cuestión, una cantidad de este y los valores en gramos de los macronutrientes encontrados en dicha cantidad.

Para contar las calorías que aporta un ingrediente, utilizaremos el sistema Atwater, que considera que 1 gramo de carbohidratos o proteínas equivale a 4 kilocalorías y 1 gramo de grasas/lípidos a 9 kilocalorías.

Instalación

Usando Stack

Para instalar la aplicación se deberá poseer la herramienta Stack, que puede ser descargada siguiendo las instrucciones de su página[1]. Los archivos con el código pueden obtenerse desde el repositorio en GitHub[2] clonando el repositorio con **git clone**.
Luego, para instalarlo usando la herramienta Stack, basta con abrir una terminal y ejecutar lo siguiente **dentro de la carpeta del repositorio**:

```
$ stack setup
$ stack install
```

El binario se copiará en el directorio `$HOME/.local/bin`, por lo tanto es recomendable revisar que dicho directorio se encuentre en la variable `PATH`.

Paquetes utilizados

Los paquetes que utilicé para hacer el trabajo son: pretty, dates, readline, ansi-terminal, parsec y directory, los cuales deberían instalarse automáticamente gracias a Stack siguiendo los pasos anteriormente mencinados.

[1] `https://docs.haskellstack.org/en/stable/README/`
[2] `https://github.com/juliojg/RecipeManager`

Detalles

- La existencia de que dos funciones sean similares en el módulo Main (readevalprint y readevalprintRM) se debe a la distinción que hice entre que el programa o está esperando que se cargue un inventario o está en un inventario, motivo por el cual también distingo los comandos propios de cada situación en el módulo Types.

- Considero iguales a dos recetas si poseen el mismo nombre, independientemente de sus otros campos. Motivo por el cual solo puede existir una con un nombre determinado.

- Decidí no distinguir entre unidades de medida para los ingredientes[3], quedando a discreción del usuario el que obtenga resultados consistentes debido a esto (para lo cual deberá utilizar la misma cuando trate con el mismo ingrediente.)

- Considero distintos a dos ingredientes si poseen fecha de vencimiento distinta, motivo por el cual puede aparecer mas de uno con el mismo nombre en el inventario (si uno posee el mismo nombre y vencimiento que otro ya ingresado, se sumará su cantidad a la que ya existe.)

- Los datos decidí almacenarlos en listas debido a su simplicidad. Cada vez que se agrega un ingrediente a la lista de ingredientes de un inventario, las funciones encargadas de ello lo hacen de manera tal que queden agrupados los que tengan mismo nombre, y que cada uno de estos grupos esté ordenado de forma creciente según su fecha de vencimiento.

- Siempre que se elimine un ingrediente del inventario, si el usuario posee muchas unidades del mismo con distinto vencimiento, se asume que quiere eliminar aquella que venza antes, por lo que dicha ocurrencia tendrá prioridad a la hora de ser eliminada.

- Para este trabajo consideraremos los valores de una comida preparada como la suma de los valores de los ingredientes que la componen.

[3]Siendo "Grams" el nombre que elegí para esta unidad genérica, por ser el que consideré más común

Manual de uso

Lanzando la aplicación

La aplicación puede iniciarse desde una terminal de la siguiente forma:

```
$ RecipeManager
```

Luego de esto verá un mensaje de bienvenida junto al prompt, lo que le dejará utilizar el intérprete de los comandos del programa. Al guardar un inventario, de no existir se creará en el lugar donde se está ejecutado el programa una carpeta `saves/`, que contendrá los archivos con los datos de todos los inventarios que se creen, motivo por el cual, de querer cargar estos, deberá ejecutar el programa desde el mismo directorio o llevar los archivos de esta carpeta a la creada en el nuevo lugar de ejecución.

Uso del intérprete

La aplicación puede estar o bien `esperando a que se cargue un inventario` o `dentro de uno`, los comandos de archivos se refieren al primer caso, mientras que los de inventario al segundo.

Comandos de archivos

- `new_inv` nombre · Crea un inventario con el nombre especificado.
- `load` nombre · Carga un inventario ya creado y guardado con el nombre especificado (todos los inventarios guardados por este programa acaban en ".rcpm")
- `quit` · Cierra la aplicación.
- `help` · Muestra en pantalla los comandos disponibles, junto a una explicación de su función.

Comandos del inventario

- `add_ing` nombre-cantidad-dia/mes/año · Agrega cierta cantidad de un ingrediente al inventario junto con su vencimiento, siempre y cuando este haya sido ingresado previamente en la tabla de valores nutricionales.
- `add_rcp` nombre -i i_1[;i_2...] -p p_1;[p_2;...] -t t_1;[t_2;...] -f · Agrega una receta a la lista del inventario (Donde los términos i_i tienen la forma "nombre_de_ingrediente-cantidad", p_i de cualquier parrafo sin ";", y t_i una palabra.)
- `add_t` nombre-cantidad-carbohidratos proteinas grasas · Agrega un ingrediente a la tabla de valores nutricionales del inventario.
- `import_t nombre` · Añade las entradas de la tabla de valores nutricionales del inventario que se le pasa como argumento al actual.

- `rm_t` nombre · Elimina un ingrediente de la tabla de valores nutricionales del inventario, junto a todas las entradas que existan de estos en el inventario.
- `rm_ing` nombre-cantidad · Elimina cierta cantidad de un ingrediente del inventario.
- `rm_rcp` nombre · Elimina una receta de la lista.
- `showr` nombre · Muestra una receta y sus pasos en pantalla, de existir.
- `check` · Revisa el vencimiento de los ingredientes en el inventario, mostrando aquellos vencidos, de existir.
- `display` · Muestra en pantalla los ingredientes y recetas del inventario, junto con el diario de calorias consumidas.
- `display_t` . Muestra los ingredientes cuyos dayos han sido ingresados a la tabla de valores.
- `need_food` [+ {$cond_1$[, $cond_2$,...]} [- {tag_1[, tag_2, ...]}]] · Muestra una lista de las recetas preparables con los ingredientes disponibles, pudiendo además especificar ciertas condiciones que estas deben cumplir. (Los términos $cond_i$ tienen la forma " </ > cantidad carb/prot/grasas/cal " o tag_i, y los tag_i tienen la forma de una palabra.)
- `i_eat` nombre . Elimina del inventario los ingredientes necesarios para preparar cierta receta (siempre y cuando se disponga la cantidad requerida de estos) y agrega al diario la cantidad de calorías de la receta a la consumida en el día.
- `save` . Guarda el inventario actual, generando un archivo en la carpeta */save* del directorio donde se está ejecutando la aplicación.
- `close` . Cierra el inventario, permitiendo cargar/crear otro o salir de la aplicación.
- `help` · Muestra en pantalla los comandos disponibles, junto a una explicación de su función.

Organización del código

RecipeManager/

Carpeta principal de la aplicación, contiene subdirectorios que explicaremos a continuación y los siguientes archivos:

- **RecipeManager.cabal**: archivo con las dependencias necesarias para instalar la aplicación con Stack.
- **stack.yaml**, **Setup.hs**, **src/**, **test/**, **ChangeLog.md**: archivos y directorios generados por la herramienta Stack para funcionar correctamente.
- **LICENSE**
- **README.md**

RecipeManager/app/

Esta carpeta contiene los módulos **.hs** que conforman la aplicación. A continuación explicaré de qué se trata cada uno.

Types.hs

En este módulo están definidas las estructuras empleadas por la aplicación, el árbol de sintaxis abstracta de nuestro lenguaje y distintas instancias para los tipos definidos. Ciertos tipos fueron redefinidos con la sentencia `type` para darles un nombre más acorde al contexto.
Para utilizar el mismo tipo de datos tanto para los ingredientes del inventario como para los necesarios para recetas (estos últimos sin vencimiento), se encapsuló la fecha de expiración de un ingrediente con la mónada Maybe.

Parser.hs

En éste módulo están definidos mediante la librería Parsec [4] los analizadores sintácticos encargados tanto de traducir los comandos que ingresa el usuario a unos con los que la aplicación puede trabajar (o retornar un error si lo ingresado no era un comando válido), como de intepretar los archivos ".rcpm" que genera el programa para poder cargar inventarios ya creados y guardados anteriormente.

Pretty.hs

Este módulo emplea la librería Pretty[5] y provee la función que se encarga de guardar un inventario para posteriormente cargarlo. Para esto genera un archivo de texto de tal manera que pueda ser interpretado por un analizador del módulo Parser para recuperar el estado que llevaba previamente el programa.

Commands.hs

En este módulo se encuentran definidas las funciones que manipulan el estado que lleva el programa (junto a otras auxiliares necesarias para esto), cada una puede corresponderse con un comando que el usuario puede efectuar, dicha correspondencia puede observarse en las funciones handleComm y handleCommRM del módulo Main.

Monads.hs

Define la mónada empleada para estructurar la aplicación, dicha mónada es StateError (inspirada en la definida en clase) y se encarga de realizar acciones IO, manejar errores y llevar el estado del programa, siendo esto posible al haber sido instanciada respectivamente en las clases MonadIO (del módulo `Control.Monad.IO.Class`), MonadError[6] y MonadState (ambas vistas durante el cursado)

[4] https://hackage.haskell.org/package/parsec
[5] https://hackage.haskell.org/package/pretty
[6] Se agregó el método *catchError* inspirado en la función catch del paquete Control.Exception que permite al programa recuperarse de un error.

Main.hs

Módulo encargado de la interfaz de usuario y donde se encuentra definido el bucle que le brinda interactividad al programa. Se encarga de pasar la entrada del usuario a los analizadores, de asociar los comandos de usuario a sus operaciones correspondientes y de manejar los errores que estas pueden lanzar para proseguir la ejecución.

RecipeManager/app/save

Ejecutando el programa en la carpeta app/ se generaría esta carpeta, la cual posee el archivo *example.rcpm* a modo de ejemplo de un inventario ya creado.

RecipeManager/doc/

Aquí se encuentran los archivos referidos a este informe.

Referencia bibliográfica

- **Wikipedia**
  `https://en.wikipedia.org/wiki/Atwater_system`

- **Stack Overflow**
  `https://stackoverflow.com/questions/4063592/how-can-i-write-a-state-monad-that-does-e`
  `https://stackoverflow.com/questions/45435845/chaining-state-monad`

- **Hackage**
  `https://hackage.haskell.org/package/dates-0.2.2.1/docs/Data-Dates.html`
  `https://hackage.haskell.org/package/parsec`
  `https://hackage.haskell.org/package/pretty`
  `https://hackage.haskell.org/package/base-4.10.1.0/docs/Control-Exception.html`
  `https://hackage.haskell.org/package/transformers-0.3.0.0/docs/Control-Monad-IO-Class.`

Comentarios finales

Agradezco a:

- Guido De Luca por ayudarme con el uso de Stack.
- Luis Dzikiewicz e Iván Ernandorena, por sus consejos y ayuda.