UNR Universidad Nacional de Rosario

Universidad Nacional de Rosario

Facultad de Ciencias Exactas, Ingeniería y Agrimensura

Lenguaje de organización de tareas

Trabajo práctico final
Análisis del Lenguajes de Programación

Autor:
Arroyo, Joaquín

6 de febrero de 2023

Índice

1. Introducción

1.1. Idea general

En este lenguaje el usuario va a poder organizar sus tareas en carpetas y perfiles. Cada perfil va a contar con su propio directorio **root**, y dentro de cada directorio, el usuario va a poder moverse para insertar y acceder a la información que necesite; esta última puede filtrarse.

En lo que refiere a las **tareas**, estas tienen una serie de atributos

Nombre No puede ser nulo. Es único a nivel carpeta.

Descripción No puede ser nulo.

Bandera Esta se utilizará para marcar la tarea como terminada. En un principio se setea en false.

Timestamp Esta se utilizará para indicar fecha lítime de la tarea. Puede ser nulo.

Prioridad Esta se utilizará para indicar la prioridad de la tarea. Va a ser un valor natural, donde **1** indica la mayor prioridad y **0** indica que la tarea no tiene prioridad.

en lo que refiere a las **carpetas**, estas tienen los siguientes atributos

Nombre No puede ser nulo. Es único a nivel carpeta.

Subcarpetas Una lista de subarpetas.

Tareas Una lista de tareas.

y en lo que refiere a los **perfiles**, estos van a contar con

Nombre Es único a nivel perfil.

Directorio Contiene toda la información del perfil.

El usuario va a poder crear/cargar/eliminar perfiles, y siempre va a manipular la información de uno a la vez. El **sistema de perfiles**[4.3] es detallado mas abajo.

1.2. Funcionalidades generales

El usuario va a tener disponible las siguientes funcionalidades a partir de los comandos:

1. **Creación/Edición/Borrado** Sobre carpetas.
2. **Creación/Edición/Borrado** Sobre tareas.
3. **Navegar sobre el sistema de carpetas** A partir de una ruta.
4. **Creación/Borrado** Sobre perfiles.
5. **Navegar sobre el sistema de perfiles**
6. **Mostrar todo el contenido de una carpeta** Se van a mostrar tanto las subcarpetas como las tareas.
7. **Búsqueda sobre tareas** A partir de un sistema de filtros sobre los atributos de las tareas. El **sistema de filtros**[4.2] es detallado mas abajo.

2. Decisiones de diseño

Diseño de sistema de carpetas La estructura **Folder** contiene un nombre, un mapa que representa a las subcarpetas, y un mapa que representa a las tareas de la carpeta. El sistema va a contar con una sola carpeta, llamada **root**.
Para poder implementar el movimiento dentro del sistema, se lleva como estado, además de la carpeta actual, la carpeta root entera, ya que al avanzar sobre la estructura, vamos perdiendo información, y a partir de dicha carpeta, podremos volver a recuperar la información que necesitemos. Esto implica que cada vez que realicemos un cambio sobre el directorio actual, este se va a tener que replicar sobre el directorio root.

Diseño de sistema de perfiles Para el diseño de este sistema, se decidió utilizar archivos .json para codificar las estructuras involucradas en el sistema de carpetas. Para esto se utilizó el modulo externo **Data.Aeson**[6.4].

Diseño de parsers Para el diseño del parser de comandos y del parser de expresiones de filtro, se decidió utilizar el software Happy, ya que nos facilita el diseño de estos mismos, y además, ya fue utilizado en trabajos pasados.

Wrapper para representar la prioridad Se decidió crear una estructura cáscara con un único contructor [**P** Integer] representar a la prioridad, ya que de esta manera, podemos instanciar de una manera más sencilla el orden. En este caso, al utilizar operaciones de orden, queremos que la mayor prioridad sea 1, y la menor sea 0.

Wrapper para representar fecha y hora Esta estructura cáscara nos permite llevar una fecha [**D** LocalTime], una fecha nula [**Null**] o un error de parseo en la fecha [**Error**]. Esto último fue incluido ya que la fecha es parseada con la librería **Data.Time**[6.4] al momento de que el comando pasa por nuestro parser. Esto fue una optimización, ya que en un principio, la fecha era pasada como string, y luego era parseada cada vez que se necesitaba utilizar; por ejemplo en el evaluador de filtros (una vez por cada tarea a analizar). Ahora, antes de pasar por este evaluador, la expresión es chequeada, y en caso de que tenga un error, no pasamos por el evaluador; caso contrario, ya tenemos la fecha parseada.
En el caso del evaluador de comandos, esto no represento una mejora sustancial.

3. Instalación

Para la instalación, vamos a necesitar tener instalado **Stack**[6.1].
Una vez lo tengamos instalado, vamos a ingresar al directorio **TP-FINAL** y ejecutar los siguientes comandos:

```
$ stack setup
```

Este comando se encarga de instalar la versión correcta de GHC, instalar los paquetes necesarios y compilar el proyecto.

```
$ stack build
```

Este comando compila el proyecto.

4. Manual de uso

Para utilizar el programa basta con haber realizado la instalación, y ejecutar el comando

```
$ stack exec TP-FINAL-exe
```

A partir de aquí, se detallan los comandos y funcionalidades disponibles del programa.

4.1. Comandos

Creación de tareas

```
$ newtask (<Name>, <Description>, <Timestamp -
    Optional>, <Priority - Optional>)
```

Podemos utilizar ignorar los campos opcionales para crear tareas, por ejemplo

```
$ newtask (Prueba, Esto es una prueba)
$ newtask (Prueba, Esto es una prueba, 1)
$ newtask (Prueba, Esto es una prueba, 2023/11/25
   14:30)
$ newtask (Prueba, Esto es una prueba, 2)
```

La tarea se va a crear sobre la carpeta que estemos parados.

Edicion de tareas

```
$ edittask <Name> set <Field> <Value>
$ completetask <Name>
```

Solo vamos a tener acceso a tareas que se encuentren en la carpeta que estamos parados.
$\langle Field \rangle$ puede tener los siguientes valores:

Name En este caso $\langle Value \rangle$ debe a ser una cadena alfanumérica, puede contener espacios.

Description En este caso $\langle Value \rangle$ debe a ser una cadena alfanumérica, puede contener espacios.

Completed En este caso $\langle Value \rangle$ debe ser **true** o **false**.

Timestamp En este caso $\langle Value \rangle$ debe ser una fecha de la forma **aaaa/mm/dd** y puede incluir o no la hora de la forma **hh:mm**.

Priority En este caso $\langle Value \rangle$ debe ser un número natural.

Borrado de tareas

```
$ deletetask <Name>
```

Solo vamos a tener acceso a tareas que se encuentren en la carpeta en la que estemos parados.

Creacion de carpetas

```
$ newdir <Name>
```

La carpeta se va a crear dentro de la carpeta en la que estemos parados.

Edicion de carpetas

```
$ editdir <Name>
```

Solo podremos editar el nombre de la carpeta sobre la que estamos parados.

Borrado de carpetas

```
$ deletedir <Name>
```

Solo vamos a tener acceso a carpetas que se encuentren en la carpeta en la que estemos parados.
Se va a borrar recursivamente todo el contenido de la carpeta.

Navegacion sobre el sistema de carpetas

```
$ cd <Route>
$ cd ..
```

⟨*Route*⟩ tiene que tener la siguiente forma ⟨*Name*⟩/⟨*Name*⟩/... o simplemente ⟨*Name*⟩ donde **Name** tiene que ser el nombre de una carpeta.
El segundo comando, nos permitira ir hacia atras en el directorio.

Mostrar el contenido de una carpeta

```
$ ls
```

Siempre se van a mostrar las carpetas por un lado, y las tareas por otro. Estas últimas se van a mostrar ordenadas de mayor a menor prioridad.

4.2. Sistema de filtros

Este sistema nos permite realizar búsquedas filtradas sobre las tareas, dichos filtros son expresiones booleanas las cuales van a involucrar a operadores booleanos, a los atributos de las tareas y a algunas funciones especiales.
El sistema provee el comando

```
$ search [-f] <Filter>
```

el cual retorna una lista de todas las tareas que se encuentren en la carpeta actual que cumplan con el filtro. En caso de utilizar la bandera -f, buscará recursivamente en las subcarpetas.
< *Filter* > tiene que ser una expresion de este sistema.
La sintaxis abstracta de las expresiones es la siguiente:

```
fieldstringatom ::= description
                  | name
boolatom ::=  true
            | false
timestring ::= nat/nat/nat
             | nat/nat/nat nat:nat
exp ::= fieldstringatom ilike var
      | fieldstringatom = var
      | completed = boolatom
      | timestamp = timestring
      | priority = nat
      | fieldstringatom != var
      | completed != boolatom
      | timestamp != timestring
      | priority != nat
      | timestamp <= timestring
      | priority <= nat
      | timestamp >= timestring
      | priority >= nat
      | timestamp < timestring
      | priority < nat
      | timestamp > timestring
      | priority > nat
      | exp and exp
      | exp or exp
      | not exp
```

Por ejemplo

```
$ search timestamp >= 2023/11/25 and name ilike examen
$ search -r priority > 2 and timestamp <= 2023/02/28
```

4.3. Sistema de Perfiles

Este sistema le permite al usuario no solo persistir la información que fue manipulando mientras utilizaba el programa, si no también le da versatilidad a la hora de utilizar y ordenar la información.
En un principio, se el programa inicia con el perfil **default**, y luego podremos crear/eliminar/cargar otros perfiles.
Los perfiles son guardados en archivos **.json**; y, al salir y entrar al programa, siempre vamos a tener cargado el último perfil utilizado. Esto es posible ya que el sistema guarda esa información cada vez que se carga un nuevo perfil.

Creación de perfiles

```
$ newprofile <Name>
```

Este comando va a crear un nuevo perfil con el nombre recibido, solo si no existe otro perfil con el mismo nombre.

Eliminacion de perfiles

```
$ deleteprofile
```

Este comando va a eliminar el perfil sobre el cual estamos parados. No se puede eliminar el perfil **default**.

Carga de perfiles

```
$ load <Name>
```

Este comando va a cargar el perfil, siempre y cuando exista un perfil con el nombre recibido.

Guardado de perfiles

```
$ save
```

Este comando va a guardar el perfil sobre el cual estamos parados.

4.4. Sistema de exportación de tareas

El programa provee el comando

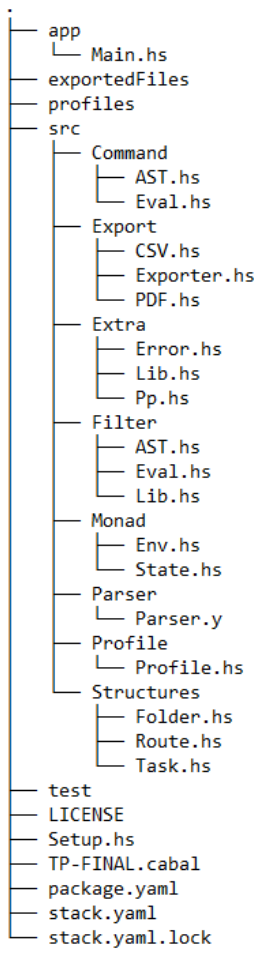
```
$ export <FileType>
```

el cual nos permite exportar las tareas de la carpeta en la que estemos parados, hacia un archivo de tipo $\langle FileType \rangle$.
Actualmente el programa nos permite exportar hacia los siguientes tipos de archivos:

- **pdf**
- **csv**

Los archivos se crearan en la carpeta **exportedFiles** con el nombre $tasks.\langle FileType \rangle$ donde **FileType** va a ser el tipo de archivo elegido.

5. Organización de archivos



En el directorio **app** se define el módulo **main**, que define el ejecutable final.
En el directorio **exportedFiles** se van a guardar los archivos exportados con el sistema de exportación.
En el directorio **profiles** se van a guardar los perfiles que defina el usuario, y un archivo que contiene el ultimo perfil cargado.
En el directorio **src** se definen los siguientes modulos:

Command En este módulo se encuentra el AST de los comandos, y su evaluador.

Export En este módulo se encuentra la definición del exportador general, y los exportadores a los respectivos tipos de archivos.

Extra En este módulo se encuentra la definición de los errores, del pretty printer, y una libreria con funciones genericas.

Filter En este módulo se encuentra el AST de las expresiones de los filtros, su evaluador y funciones útiles relacionadas a los filtros.

Monad En este módulo se encuentra la definición de la mónada **State**, y de la estructura **Env** (junto a sus funciones), la cuál es utilizada por la mónada mencionada anteriormente.

Parser En este módulo se encuentra la definición del **Parser**, el cual define el parser de comandos y el parser de las expresiones de filtros.

Profile En este módulo se van a encontrar la definición de las funciones/instancias que permiten manipular los perfiles.

Structures En este módulo se van a encontrar las definiciones tanto de las estructuras **Folder**, **Route** y **Task**, como las definiciones de sus respectivas funciones.

En el directorio **test** podrían encontrarse tests.
El resto de los archivos son parte de la configuración del proyecto.

6. Bibliografía, software de terceros

6.1. Stack

Es una herramienta que nos permite desarrollar aplicaciones con Haskell.
Para instalarlo visitar docs.haskellstack.org.

6.2. Happy

Es un generador de parsers para Haskell y fue utilizado para generar los distintos parsers utilizados en el programa. Happy puede trabajar junto a un analizador lexicográfico (una función que divide la entrada en tokens, que son las unidades básicas de parseo) proporcionado por el usuario.
Para leer mas sobre **Happy** puede visitar haskell-happy.readthedocs.io.

6.3. Hindent

Herramienta utilizada para identación automática de los archivos Haskell. Para mas información visitar github.com/mihaimaruseac/hindent.

6.4. Referencias

- hackage.haskell.org/package/mtl-2.3.1/docs/Control-Monad-Except
- hackage.haskell.org/package/text-2.0.1/docs/Data-Aeson
- hackage.haskell.org/package/bytestring-0.11.4.0/docs/Data-ByteString-Lazy
- hackage.haskell.org/package/cassava-0.5.3.0/docs/Data-Csv
- hackage.haskell.org/package/text-2.0.1/docs/Data-List
- hackage.haskell.org/package/text-2.0.1/docs/Data-Map
- hackage.haskell.org/package/text-2.0.1/docs/Data-Text
- hackage.haskell.org/package/time-1.12.2/docs/Data-Time
- hackage.haskell.org/package/base-4.17.0.0/docs/GHC-Generics
- hackage.haskell.org/package/HPDF-1.6.0/docs/Graphics-PDF
- hackage.haskell.org/package/haskeline-0.8.2/docs/System-Console-Haskeline
- hackage.haskell.org/package/directory-1.3.8.0/docs/System-Directory
- hackage.haskell.org/package/ansi-wl-pprint-0.6.9/docs/Text-PrettyPrint-ANSI-Leijen