Programming in Haskell – Homework Assignment 9

UNIZG FER, 2013/2014

Handed out: January 21, 2013. Due: January 31, 2013 at 23:59

Note: Define each function with the exact name and type specified. You can (and in most cases should) define each function using a number of simpler functions. Provide a type signature above each function definition and comment the function above the type signature. Unless said otherwise, a function may not cause runtime errors and must be defined for all of its input values. Use the `error` function for cases in which a function should terminate with an error message, and return the exact error message specified. Problems marked with a star ($\star$) are optional.

1. Define a simple utility for measuring the time an action spent executing. One way to get the current time is to use `Data.Time.Clock.POSIX.getPOSIXTime`.

   ```
   time :: IO () -> IO Double
   time (putStrLn "printing this line takes time") ⇒ return 0.00204
   time (mapM_ print [1..100000]) ⇒ return 4.608441
   ```

2. Define a simple grep-like utility.

   (a) Read a file and output only the lines containing some given string. (Hint: use `Data.List.isInfixOf`.)

   ```
   grep :: String -> FilePath -> IO ()
   $ echo \some\ntext\nhere\nokay\nnot okay" > lines.txt
   ghci> grep "o" "lines.txt"
   some
   okay
   not okay
   ghci> grep "ok"
   okay
   not okay
   ```

   (b) Use `System.Environment.getArgs` to read the command-line arguments. Pass them into the previously-defined grep action. You can use `System.Environment.withArgs` to simulate command-line arguments within the interpreter.

   ```
   grepWithArgs :: IO ()
   ghci> withArgs ["o", "lines.txt"] grepWithArgs
   some
   okay
   not okay
   ```

   (c) Lists of characters are not a very memory efficient representation of text. For most purposes, a better way to store text is by way of a mostly contiguous array. Reimplement `grep` using the `Text` type from the `Data.Text.Lazy` and

`Data.Text.Lazy.IO` modules. Where needed, use `Data.Text.Lazy.pack` to convert a `String` into a `Text` value.

```
grepText :: Data.Text.Lazy.Text -> FilePath -> IO ()
> grepText (Data.Text.Lazy.pack "okay") "lines.txt"
okay
not okay
```

Use the previously defined timing action to see which of the two greps is faster for large files.

3. Create an API for a simple text-based database (saved in a textual file). All database fields are of type `String`, while each row is a `[String]`. A database is abstracted as

   ```
   type Table = (FilePath,[String])
   ```

   where `FilePath` is filepath of the database file and `[String]` are the column labels. Feel free to define `Table` in some other way if it suits you better.

   Example of a database file:

   ```
   name surname age
   John James 23
   Robert Robertson 48
   Linda Lindson 31
   ```

   (a) Define a function that takes the table name and the column labels (which must be unique) and creates a file named `<tableName> ++ ".tbl"`. You may assume that column names don't contain spaces.
   `dbCreateTable :: String -> [String] -> IO Table`

   (b) Define a function that deletes the file holding the table.
   `dbDeleteTable :: Table -> IO ()`

   (c) Define a function that inserts a row into the table. Check whether the input is of correct length. If yes, write it to the file, otherwise throw an exception.
   `dbInsert :: Table -> [String] -> IO ()`

   (d) Define a select operation that takes a predicate and returns all rows that satisfy it.
   `dbSelect :: Table -> ([String] -> Bool) -> IO [[String]]`

   (e) Define a delete operation that deletes all rows that satisfy the given predicate.
   `dbDelete :: Table -> ([String] -> Bool) -> IO ()`

   (f) Define an update operation that updates all rows that satisfy the given predicate using the given update function.
   ```
   dbUpdate ::
     Table -> ([String] -> Bool) -> ([String] -> [String]) -> IO ()
   ```

   (g) Define a print function that prints the table to standard output. (Hint: Use `dbSelect`.)
   `dbPrintTable :: Table -> IO ()`

Corrections

`v2: 3: changed Table definition`