Programming in Haskell – Homework Assignment 1

UNIZG FER, 2015/2016

Handed out: October 16, 2014. Due: October 22, 2015 at 17:00

Note: Define each function with the exact name specified. You can (and in most cases you should) define each function using a number of simpler functions. 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. Problems marked with a star ($\star$) are optional.

Each problem is worth a certain number of points. The points are given at the beginning of each problem or subproblem (if they are scored independently). These points are scaled, together with a score for the in-class exercises, if any, to 10. Problems marked with a star ($\star$) are scored on top of the mandatory problems, before scaling. The score is capped at 10, but this allows for a perfect score even with some problems remaining unsolved.

1. *(2 pts)* Tuples are a natural way of representing vectors. However, to make this representation useful, we need it to support a set of basic vector operations. You will implement vectors in 2D space using 2-tuples, as `(x, y)`, by implementing the following functions:

   (a) Define a `norm` function that computes the Euclidean norm of a vector. (Hint: see the `sqrt` function.)

   ```
   norm (3, 4) ⇒ 5.0
   norm (0, 0) ⇒ 0.0
   norm (3, 0) ⇒ 3.0
   ```

   (b) Define a `normalize` function that returns a normalized version of a given vector.

   ```
   normalize (3, 4) ⇒ (0.6, 0.8)
   normalize (1, 1) ⇒ (0.7071, 0.7071)
   normalize (0, 0) ⇒ error "Cannot normalize null vector"
   ```

   (c) Define a `scalarMult` function that takes a scalar and a vector and returns their product.

   ```
   scalarMult 0 (3, 3) ⇒ (0, 0)
   scalarMult (-1) (2, 3) ⇒ (-2, -3)
   scalarMult 2 (1, 2) ⇒ (2, 4)
   ```

   (d) Define a `dot` function that computes the dot product of two vectors.

   ```
   dot (1, 1) (1, 1) ⇒ 2
   dot (3, 5) (2, 1) ⇒ 11
   ```

```
dot (2, -1) (0, 5) ⇒ -5
```

(e) Define the `cos'` function that computes the cosine of the angle between two vectors. Use the functions you defined in previous subtasks.

```
cos' (1, 0) (1, 0) ⇒ 1.0
cos' (1, 0) (0, 1) ⇒ 0.0
cos' (3, 2) (5, 4) ⇒ 0.9962
cos' (0, 0) (1, 1) ⇒ error "Null vector given"
cos' (1, 1) (0, 0) ⇒ error "Null vector given"
```

(f) Define the `areParallel` function that checks whether two given vectors are parallel, returning a boolean.

```
areParallel (1, 1) (1, 1) ⇒ True
areParallel (1, 1) (2, 1) ⇒ False
areParalell (1, 1) (2, 2) ⇒ True
```

2. *(2 pts)* Only three contestants are left in the final round of the Nintendo World Championship, each hoping to win the prestigious title. Each contestant plays a game and is then ranked according to their total score (a higher score means a higher rank). If two players obtain the same score, they are ranked lexicographically.

   Define the `finalRanking` function that takes the names and scores of *exactly* three players as 2-tuples `(name, score)` and returns them ranked, in the form of a list. The highest-ranked player is placed first (index 0) and the lowest-ranked player is placed last.

   Use `if-then-else` or guards in the function implementation.

```
finalRanking ("John", 250) ("Paul", 360) ("Ringo", 10)
⇒ ["Paul", "John", "Ringo"]
finalRanking ("Eden", 1000) ("Leo", 500) ("Cristiano", 500)
⇒ ["Eden", "Christiano", "Leo"]
```

3. *(4 pts)* Suppose there exist three directories on your hard drive named `A`, `B` and `C`. Each of those directories contains image, video and/or audio files. You need to determine the *minimal* number of files that need to be moved between the three directories so that each directory contains only one type of media (image, video or audio). Finally, you wish to name the directories according to the type of media they contain.

   To solve this task, write a program that reads data from the standard input (stdin). It should read in three lines: for directories A, B, and C, respectively. Each line contains the names of the files in that directory, delimited by spaces (Hint: `words` function). Suppose all images have the `.png` extension, all audio files have the `.mp3` extension and all video files have the `.mp4` extension. (Hint: `Data.List.isSuffixOf`.)

   The program should then output four lines: the first should contain the minimal number of move operations needed to solve the problem. The other three lines should contain the new names for the directories `A`, `B`, and `C` (`image`, `audio`, and `video`, in some order). (Another hint: List comprehensions).

Make sure that the code can be compiled and run as a binary (define a `main` function).

```
> ./task3
> img1.png img2.png s01ep01.mp4
> s01ep02.mp4 s01ep03.mp4 track01.mp3
> track02.mp3 s01ep04.mp4
3
image
video
audio
```

Corrections

Revision 1.1 – Renamed `cos` to `cos'`.
Revision 1.2 – Integer ⇒ Floating in `norm` function.
Revision 1.3 – Added examples for null vector argument.