Programming in Haskell – Homework Assignment 1

UNIZG FER, 2014/2015

Handed out: October 5, 2014. Due: October 9, 2013 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.

1. Define `strlenInRange` as a function that takes a string and length boundaries and checks whether the string length is within those bounds (inclusive). If either of the length boundaries is negative, the function should return an error. To do so, call the `error` function: `foo x = error "Not implemented"`.

   ```
   strlenInRange "Duck" 0 4 ⇒ True
   strlenInRange "Duck" 4 0 ⇒ False
   strlenInRange "Aardvark" 1 5 ⇒ False
   strlenInRange any (-2) 7
   ⇒ error "String length cannot be a negative number"
   ```

2. You are given a list of numbers, an index and a value. Determine if an element of list with given index is larger than provided value. Use list index (subscript) operator `(!!)` or `genericIndex` function that returns element from list with the desired index. If the index is out of range you should return `False`. Numbers in the list and the value argument can be floating point numbers.

   ```
   isHereAGreater [1,3,5,7] 0 1 ⇒ False
   isHereAGreater [1,3,5,7] 1 1 ⇒ True
   isHereAGreater [1,3,5,7] 0 7 ⇒ False
   ```

3. Implement a `wordFilter` function using `do` notation. It needs two lines of user input – a sentence and a single word. All occurrences of the given word in the sentence should be removed and the sentence should be printed out. (*Hint:* use `words` to split sentence into a list of words and `unwords` to merge them with whitespace in between.

   ```
   > This is an example
   > is
   This an example
   ```

   ```
   > Just testing.
   > test
   Just testing.
   ```

4. Using *if-then-else*, define a function `ord3` that returns a sorted list of given numbers in ascending order.

```
ord3 5 2 1 ⇒ [1,2,5]
ord3 1 5 2 ⇒ [1,2,5]
```

5. Tuples are a natural way of representing vectors. However, for the representation to be of any use, operations over vectors need to be implemented. Using 2-tuples to represent vectors in $2D$ space, as `(x,y)`, implement the following functions:

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

   ```
   norm (3,4) ⇒ 5
   norm (-1,4) ⇒ 4.1231056
   ```

   (b) Define the `add` function that adds two vectors.

   ```
   add (1,4) (0,-2) ⇒ (1,2)
   add (8,3) (2,3) ⇒ (10,6)
   ```

   (c) Define the `scalarMult` function that takes a vector and a scalar and multiplies them.

   ```
   scalarMult (3,1) (-1) ⇒ (-3, -1)
   scalarMult (-5,0) 3 ⇒ (-15, 0)
   ```

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

   ```
   dot (-6,8) (5,12) ⇒ 66
   dot (-1,1) (3,3) ⇒ 0
   ```

6. Define the function `asciiRange` that takes two characters and returns a list of pairs *(character, ascii code)* for all characters in that range. Accomplish this using list ranges (`['a'..'z']`) and the `zip` function. Calling `asciiRange` with the first argument greater than the second should return an empty list. (*Hint:* use `Data.Char.ord` and `Data.Char.chr` to convert between a `Char` and its `Integer` representation.)

   ```
   asciiRange 'a' 'c' ⇒ [('a',97),('b',98),('c',99)]
   asciiRange 'E' 'E' ⇒ [('E',69)]
   asciiRange 'Z' 'A' ⇒ []
   ```

7. Using a list comprehension, define a function `incn` that increments each character of a `String` N times.

   ```
   incn 1 "abc" ⇒ "bcd"
   incn 3 "EF" ⇒ "HI"
   incn 0 "haskell" ⇒ "haskell"
   incn (-2) "noo" ⇒ "lmm"
   ```