The University of Melbourne
School of Computing and Information Systems
COMP30020
Declarative Programming
Second Semester, 2018

Assignment 1

Due Wednesday, 15 August 2018, 5:00PM

Worth 0% (optional assignment)

The objective of this project is to practice your Haskell programming skills. You will write a few fairly simple Haskell functions.

The Assignment

You will implement the following Haskell functions.

1. `elementPosition :: Eq t => t -> [t] -> Int`
   `elementPosition` takes an element `elt` and a list `lst`, and returns the number of elements of `lst` that must be skipped over to find the first occurrence of `elt`. That is, it returns the zero-based index of the first occurrence of `elt` on `lst`. It should return the length of the list if `elt` is not on `lst` at all. For example:

   ```
   elementPosition 3 [1,2,3,4,5]    should return  2
   elementPosition 'e' "elephant"   should return  0
   elementPosition 'p' "elephant"   should return  3
   elementPosition 'z' "elephant"   should return  8
   ```

2. `everyNth :: Int -> [t] -> [t]`
   `everyNth` takes a number `n` and a list `lst` and returns a list of every `n`th element of `lst`, beginning with the `n`th element of `lst`. For example:

   ```
   everyNth 4 "elephant"                  should return   "pt"
   everyNth 2 [2,3,5,7,11,13,17]          should return   [3,7,13]
   everyNth 1 [2,3,5,7,11,13,17]          should return   [2,3,5,7,11,13,17]
   everyNth 0 [2,3,5,7,11,13,17]   should report an error
   ```

3. `sumLater :: Num a => [a] -> [a]`
   `sumLater` takes a list of numbers and returns a list of the partial sums of following elements. That is the result is the same length as the input, and for each element of the input, the corresponding output value is the sum of that number and all the numbers following it on the list. For example:

   ```
   sumLater [1,2,3,4,5]         should return   [15,14,12,9,5]
   sumLater [5,4,3,2,1]         should return   [15,10,6,3,1]
   sumLater [13,11,9,7,5,3,1]   should return   [49,36,25,16,9,4,1]
   ```

4. `sumEarlier :: Num a => [a] -> [a]`
   `sumEarlier` takes a list of numbers and returns a list of the partial sums up to each element. That is the result is the same length as the input, and for each element of the input, the corresponding output value is the sum of all elements up to and including that element on the list. For example:

`sumEarlier [1,2,3,4,5]`	should return	`[1,3,6,10,15]`
`sumEarlier [5,4,3,2,1]`	should return	`[5,9,12,14,15]`
`sumEarlier [1,3,5,7,9,11,13]`	should return	`[1,4,9,16,25,36,49]`

   Clearly this can be implemented as `reverse (sumLater (reverse input))`, but please don't implement it this way. The point of this exercise is to learn how to deal with two different styles of recursion: one where the work is done before the recursive call (tail recursion), and one where the work is done after the recursive call (head recursion).

You must call your source file `Assignment1.hs` or `Assignment1.lhs` (note the capitalisation), and it must begin with the module declaration:

```
module Assignment1 (elementPosition, everyNth, sumLater, sumEarlier) where
```

I will post on the LMS a test driver program called `Assignment1Test` which is substantially similar to the test driver I will use for testing your code. I will compile and link your code for testing using the following command (or similar):

```
ghc -O2 --make Assignment1Test
```

Assessment

This assignment will not be assessed for credit; that is, it is worth 0% of your final mark. To make it useful as practice for later, assessed work, your code will be tested for correctness.

Your submission will only be checked for correctness. For this assignment, code quality will not be considered. However, for your own sanity, I do recommend commenting your code and programming it carefully, paying due attention to programming technique.

Note that timeouts will be imposed on all tests. Test cases will be rather small, so the timeouts should only affect you if you create an infinite recursion (infinite loop).

Submission

You must submit your project through the *Maat* (Melbourne Automated Appraisal Tool) system by following the link from the LMS Projects page using your web browser. You may drag-and-drop your file onto the box for your assignment1file, or click the "Choose" button and use your browser's file selection dialog. Either way, once you have provided your file, click the "Save Submission" button to submit.

Important: you must wait while *Maat* tests your code. Once it has completed the appraisal, the window will refresh to show the test results. If any tests fail, the display will show which of the assigned functions failed their tests, but you will not see the failing test case, so you will need to revisit your code, and reexamine this project specification, to try to see what is wrong with your code.

It is your responsibility to look over the feedback provided for your submission.

If the test results show any problems, you may correct your code and submit again. You may submit as often as you like; only your final submission will be assessed.