CA320 Computability and Complexity

Lab 3: Defining Functions on Lists

Aim

The aim of this lab is to help you learn to define your own functions on lists, especially using recursion and list patterns.

1. Product of the Elements of a List

Define a recursive function `myProduct :: [Int] -> Int` that multiplies together all the elements of a list of integers.

To check your answer is correct (and to register your progress), you should save your program to a file named `myProduct.hs`, and drag and drop this file to the upload link on the following page: https://ca320.computing.dcu.ie/einstein

2. Converting a String to Upper Case

Haskell has a pre-defined function `toUpper :: Char -> Char` that converts single lower case letters to upper case. This function has to be imported from `Data.Char` by including the following within your script:

```
import Data.Char(toUpper)
```

Experiment with this function to find out exactly how it behaves (i.e. try it on a variety of characters).

Define a recursive function `stringToUpper :: String -> String` that converts all the lower case letters in a string to upper case, and leaves the others unchanged (remember that in Haskell the type `String` is a synonym for `[Char]`).

To check your answer is correct (and to register your progress), you should save your program to a file named `stringToUpper.hs`, and drag and drop this file to the upload link on the following page: https://ca320.computing.dcu.ie/einstein

3. Shortest List

Define a recursive function `shortest :: [[a]] -> [a]` that takes a list of lists and returns the shortest of these lists (it should return [] if the list of lists is empty). For example:

```
> shortest [[1,2,3],[1,2],[1,2,3,4,5],[4,3,2,1]]
[1,2]
```

To check your answer is correct (and to register your progress), you should save your program to a file named `shortest.hs`, and drag and drop this file to the upload link on the following page: https://ca320.computing.dcu.ie/einstein

4. Palindrome

Define a recursive function `isPalindrome :: Eq a => [a] -> Bool` that takes a list and checks if it is a palindrome. For example:

```
> isPalindrome [1,7,5,5,7,1]
```

```
True
> isPalindrome "madam"
True
```

To check your answer is correct (and to register your progress), you should save your program to a file named `isPalindrome.hs`, and drag and drop this file to the upload link on the following page: https://ca320.computing.dcu.ie/einstein

5. Adding Two Polynomials

A polynomial in a single variable `x` can be represented rather simply by a list of its coefficients. For example:

`[1,7,5,2]`	represents	$2x^3 + 5x^2 + 7x + 1$
`[42,2,1]`	represents	$x^2 + 2x + 42$
`[-3,0,0,0,1]`	represents	$x^4 - 3$
`[0,-2,0,4]`	represents	$4x^3 - 2x$

Notice how the list index for each element corresponds to the exponent of the term.

Two polynomials can be summed by adding the coefficients of corresponding terms. For example, the sum of $2x^3 + x^2 + 1$ and $3x^4 + 4x^2 - 7$ is $3x^4 + 2x^3 + - 6$.

Define a recursive function `sumPoly :: [Int] -> [Int] -> [Int]` that sums two polynomials that are represented as above. Take care with the case of polynomials with different degrees. For example:

```
> sumPoly [1,7,5,2] [42,2,1]
[43,9,6,2]
> sumPoly [-3,0,0,0,1] [1,7,5,2]
[-2,7,5,2,1]
> sumPoly [0,-2,0,4] [1,7,5,2]
[1,5,5,6]
```

To check your answer is correct (and to register your progress), you should save your program to a file named `sumPoly.hs`, and drag and drop this file to the upload link on the following page: https://ca320.computing.dcu.ie/einstein

6. Evaluating a Polynomial

Define a recursive function `evalPoly :: Int -> [Int] -> Int` that, given a value for `x` and a polynomial, will calculate the value of the polynomial for that value of `x` . For example:

```
> evalPoly 3 [1,7,5,2]
121
> evalPoly (-2) [0,-2,0,4]
-28
> evalPoly 4 (sumPoly [0,-2,0,4] [1,7,5,2])
485
```

There are many ways to do this, but an identity that you may find helpful is the following:

$a_n x^n + \ldots + a_2 x^2 + a_1 x + a_0 = a_0 + x(a_1 + x(a_2 + x(\ldots a_n) \ldots))$

To check your answer is correct (and to register your progress), you should save your program to a file named `evalPoly.hs`, and drag and drop this file to the upload link on the following page:
https://ca320.computing.dcu.ie/einstein