CPSC 449: Assignment 2

Fall 2020

Consult the D2L site for the due date

1. We represent ***(single-variate) polynomials*** as lists of integer coefficients. For example, the list **`[1, 2, 0, 3]`** represents the polynomial $1 + 2x + 3x^3$. Specifically, the zero polynomial is represented by the empty list, and a non-zero polynomial is represented by a list of integers for which the last element is non-zero (i.e., the coefficient of the highest-degree term is non-zero).

   ```
   type Poly = [Integer]
   ```

   In the following, you are expected to demonstrate your ability to formulate primitive and/or general recursions.

   (a) *[10%]* Develop a Haskell function:

   ```
   addpoly :: Poly -> Poly -> Poly
   ```

   such that **`(addpoly`** P_1 P_2**`)`** returns the sum $(P_1 + P_2)$ of polynomials P_1 and P_2. For example:

   ```
   (addpoly [1,2] [3,4]) ⇝ [4,6]
   (addpoly [1, 1, 1, 1] [-1 1]) ⇝ [0,2,1,1]
   (addpoly [] [1, 3]) ⇝ [1, 3]
   (addpoly [] []) ⇝ []
   ```

   (b) *[15%]* Develop a Haskell function:

   ```
   multpoly :: Poly -> Poly -> Poly
   ```

   such that **`(multpoly`** P_1 P_2**`)`** returns the product $(P_1 \times P_2)$ of polynomials P_1 and P_2. For example:

   ```
   (multpoly [1,1] [1,1]) ⇝ [1,2,1]
   (multpoly [1, 1, 1, 1] [-1 1]) ⇝ [-1,0,0,0,1]
   (multpoly [2,1] [1,3]) ⇝ [2,7,3]
   (multpoly [1,1] (multpoly [1,1] [1,1])) ⇝ [1,3,3,1]
   (multpoly [1,2,1] []) ⇝ []
   ```

2. (a) *[10%]* Implement a function for merging two *sorted* list of **`Integer`**s:

   ```
   mergeLists ::  [Integer] -> [Integer] -> [Integer]
   ```

That is, given two sorted lists (i.e., in ascending order), the function returns a list containing all the elements in the argument lists in sorted order (i.e., ascending order). An example is the following:

```
mergeLists [1,3,5] [2,4,6] ⇝ [1,2,3,4,5,6]
```

(b) *[10%]* Implement a function that slits a given list into two lists, such that the first returned list contains those elements of the argument list at an even index, and the second list contains those at the odd index. Indices are zero based (i.e., the head of a list has index zero).

```
splitList ::  [Integer] -> ([Integer], [Integer])
```

The following is an example.

```
splitList [4,6,3,1,2,10] ⇝ ( [4,3,2], [6,1,10] )
splitList [7, 1, 4] ⇝ ( [7,4], [1] )
splitList [4] ⇝ ( [4], [] )
splitList [] ⇝ ( [], [] )
```

(c) *[5%]* Implement a function that performs merge sort.

```
mSort ::  [Integer] -> [Integer]
```

Your implementation must call the **mergeLists** and **splitList** functions implemented in parts (a) and (b).

3. (a) *[10%]* Prove that the implementation of merge sort in 2(c) always terminates.

(b) Consider the following function.

```
mystery :: [[Integer]] -> Integer
mystery []           = 0
mystery ((x:xs):ys) = x + mystery (xs:ys)
mystery ([]:ys)      = mystery ys
```

i. *[5%]* Give a conjecture of what the **mystery** function returns.

ii. *[10%]* Prove that the **mystery** function terminates for all inputs.

4. Prove, by structural induction, for all finite lists **xs**, we have:

```
length xs = length (reverse xs)                    (*)
```

You may assume the following equations for **reverse** and **length**:

```
reverse []        = []                             (reverse.1)
reverse (x:xs)    = (reverse xs) ++ [x]            (reverse.2)

length []         = 0                              (length.1)
length (x:xs)     = 1 + (length xs)                (length.2)

length (xs ++ ys) = (length xs) + (length ys)      (length.3)
```

Adhere to the following steps in your proof.

(a) *[5%]* State the two proof goals (i.e., base case and induction step).

(b) *[5%]* Prove the base case.

(c) *[15%]* Prove the induction step.