COMPSCI 1JC3

Introduction to Computational Thinking

Fall 2017

Assignment 4

Dr. William M. Farmer

McMaster University

Revised: November 6, 2017

Assignment 4 is a continuation of Assignment 3. Its purpose is to write a module in Haskell that involves I/O and manipulation of lists. The requirements for Assignment 4 and for Assignment 4 Extra Credit are given below. You are required to do Assignment 4, but Assignment 4 Extra Credit is optional. Please submit Assignment 4 as a single `.hs` file to the Assignment 4 folder on Avenue under Assessments/Assignments. If you choose to do Assignment 4 Extra Credit for extra marks, please submit it also as a single `.hs` file to the Assignment 4 Extra Credit folder on Avenue in the same place. Both Assignment 4 and Assignment 4 Extra Credit are due **November 17, 2017 before midnight.** Assignment 4 is worth 3% of your final grade, while Assignment 4 Extra Credit is worth 2 extra percentage points.

Although you are allowed to receive help from the instructional staff and other students, your submitted program must be your own work. Copying will be treated as academic dishonesty!

1 Background

A polynomial can be represented by the list of the coefficients in its standard form. That is, if

$$a_0 + a_1 * x^1 + a_2 * x^2 + \cdots + a_m * x^m$$

where $a_m \neq 0$ is the standard form of a nonzero polynomial p, then p can be represented by the list

$$[a_0, a_1, \ldots, a_m].$$

The zero polynomial can be represented by the empty list $[\,]$. Polynomials can be processed by manipulating their representations as lists. For example, two polynomials can be added by adding the corresponding components in their representations as lists.

We will call a list that represents a polynomial a *polynomial list.* Every list of numbers whose final value is not 0 is a polynomial list.

2 Assignment 4

The purpose of this assignment is to create a Haskell module for polynomial lists over the integers.

2.1 Requirements

1. The name of your Haskell file is `Assign_4_YourMacID.hs` where *YourMacID* is your actual MacID.
2. Your name, MacID, the date, and "Assignment 4" are given in comments at the top of your file.
3. The first uncommented line of the file should be

   ```
   module PolynomialList where
   ```

4. The file contains the following algebraic data type definition from Assignment 3:

   ```
   data Poly =
       X
     | Coef Integer
     | Sum Poly Poly
     | Prod Poly Poly
     deriving Show
   ```

5. The file includes a function `getPolyList` of type

   ```
   FilePath -> IO [Integer]
   ```

   that reads the coefficients of the standard form of a polynomial from a file in which there is one integer per line where the first integer is a_0, the second a_1, etc. and then returns the inputted integers as a polynomial list.

6. The file includes a function named `polyListValue` of type

   ```
   [Integer] -> Integer -> Integer
   ```

   such that, if `pl` is a polynomial list, `polyListValue pl n` is the value of the polynomial function represented by `pl` at `n`. Hint: use Horner's method to do the computation:

   $$a_0+a_1*x^1+a_2*x^2+\cdots+a_m*x^m = a_0+x*(a_1+x(a_2+\cdots+x*(a_m)))$$

7. The file includes a function named `polyListDegree` of type

   ```
   [Integer] -> Integer
   ```

   such that, if `pl` is a polynomial list, `polyDegree pl` is the degree of the polynomial represented by `pl`. The degree of the polynomial list `[]` should be undefined, since the degree of the zero polynomial is undefined.

8. The file includes a function named `polyListDeriv` of type

   ```
   [Integer] -> [Integer]
   ```

   such that, if `pl` is a polynomial list, `polyListDeriv pl` is the polynomial list that represents the derivative of the polynomial represented by `pl`. `polyListDeriv pl` thus symbolically differentiates a polynomial list `pl`.

9. The file includes a function named `polyListSum` of type

   ```
   [Integer] -> [Integer] -> [Integer]
   ```

   such that, if `pl` and `ql` are polynomial lists, `polyListSum pl ql` is the polynomial list that represents of the sum of the polynomials represented by `pl` and `ql`.

10. The file includes a function named `polyListProd` of type

    ```
    [Integer] -> [Integer] -> [Integer]
    ```

    such that, if `pl` and `ql` are polynomial lists, `polyListProd pl ql` is the polynomial list that represents of the product of the polynomials represented by `pl` and `ql`.

11. The file includes a function named `polyListToPoly` of type

    ```
    [Integer] -> Poly
    ```

    such that, if `pl` is a polynomial list, `polyListToPoly pl` is a polynomial of type `Poly` whose standard form is represented by `pl`.

12. The file includes a function named `polyToPolyList` of type

    ```
    Poly -> [Integer]
    ```

    such that `polyToPolyList p` is the polynomial list that represents the standard form of `p`.

13. Your file can be imported into GHCi and all of your functions perform correctly.

2.2 Testing

Include in your file a test plan for all the functions mentioned above. The test plan must include at least three test cases for each function. Each test case should have following form:

`Function:` Name of the function being tested.
`Test Case Number:` The number of the test case.
`Input:` Inputs for function.
`Expected Output:` Expected output for the function.
`Actual Output:` Actual output for the function.

In addition, your test plan must include at least one QuickCheck case for each of the functions `polyListValue`, `polyListDegree`, `polyListDeriv`, `polyListSum`, and `polyListProd`. Each QuickCheck case should have following form:

`Function:` Name of the function being tested.
`Property:` Code defining the property to be tested by QuickCheck.
`Actual Test Result:` Pass or Fail.

The test plan should be at the bottom of your file in a comment region beginning with a `{-` line and ending with a `-}` line.

3 Assignment 4 Extra Credit

The purpose of this assignment is to write Haskell program that differentiates a polynomial inputted by the user.

3.1 Requirements

1. The name of your Haskell file is `Assign_4_ExtraCredit_`*`YourMacID`*`.hs` where *YourMacID* is your actual MacID.
2. Your name, MacID, the date, and "Assignment 4 Extra Credit" are given in comments at the top of your file.
3. The first uncommented line of the file should be

   ```
   module PolyDiff where
   ```

4. The file contains the following algebraic data type definition from Assignment 3 Extra Credit:

   ```
   data Poly a =
       X
     | Coef a
     | Sum (Poly a) (Poly a)
     | Prod (Poly a) (Poly a)
     deriving Show
   ```

5. The file includes the functions `polyDeriv` and `polyAsList` from Assignment 3.

6. The file includes a function named `polyParse` of type

   ```
   String -> Poly Integer
   ```

   that takes a string like `"1 + 2x + 5x^2"` as input and returns the member of the type `Poly Integer` that the string represents as output.

7. The file includes a function `getPoly` of type

   ```
   FilePath -> IO (Poly Integer)
   ```

   that reads a string like `"1 + 2x + 5x^2"` from a file and then returns a member of the type `Poly Integer` that the string represents as output.

8. The file includes a function named `polyPrettyPrint` of type

   ```
   Poly Integer -> String
   ```

   that takes a member of the type `Poly Integer` as input and returns a nicely formatted string presentation of it as output.

9. Using `polyPrettyPrint`, the file defines `Poly Integer` as a instance of the type class `Show`.

10. The file includes a function named `polySimp` of type

    ```
    Poly Integer-> Poly Integer
    ```

    that takes a member `p` of the `Poly Integer` as input and returns a simplified member `q` of `Poly Integer` as output such that `p` and `q` represent the same polynomial function.

11. The file includes a `getPolyAndDiff` function that, of type

    ```
    FilePath -> IO String
    ```

    that reads a string like `"1 + 2x + 5x^2"` from a file and returns a nicely formatted string that represents the derivative in simplified form of the polynomial represented by the input string.

12. Your file successfully loads into GHCi and all of your functions perform correctly.

3.2 Testing

Include in your file a test plan for all the functions mentioned above. The test plan must include at least three test cases for each function and one QuickCheck case for each of the functions `polyParse`, `polyPrettyPrint`, `polyDeriv`, and `polySimp`.