

Homework 2 — assigned Monday 26 February — due Friday 9 March

General instructions

A skeleton Haskell source file `homework2.hs` will be provided with the type signatures of all the functions you are to write. Please edit that file and submit.

The skeleton file will start with `import` declarations for all Haskell libraries that you are allowed to use in your solution. You must not add any other `import` declarations.

Wherever this aids clarity, use Haskell's predefined higher-order functions. Wherever this aids clarity, use the point-free style of definition.

2.1 Simple functions on numbers (10pts)

(See Project Euler, <https://projecteuler.net/problem=14>.)

The following iterative sequence is defined for the set of positive integers:

$$\begin{aligned} n &\rightarrow \frac{n}{2}, & n \text{ even} \\ n &\rightarrow 3n + 1, & n \text{ odd} \end{aligned}$$

Using the rule above and starting with 13, we generate the following sequence:

$$13 \rightarrow 40 \rightarrow 20 \rightarrow 10 \rightarrow 5 \rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1$$

It can be seen that this sequence (starting at 13 and finishing at 1) contains 10 terms. Although it has not been proven yet (Collatz Problem), it is thought that all starting numbers finish at 1.

Define a function, `collatz :: [Int] -> Int`, that takes in a list of starting numbers and returns the one which gives rise to the longest Collatz sequence. For example, `collatz [1..20]` should evaluate to 19.

If multiple starting numbers have the same sequence length, your function should return the largest of them. For example, 18 and 19 both produce a sequence length of 21, and so 19 is reported.

NB. Once the sequence starts, the terms can become quite large. Your code must be prepared to handle this possibility.

2.2 Simple functions on lists and strings (10pts)

Write a function `haskellFileNames :: [String] -> [String]`, which, given a list of strings, will extract those strings which look like Haskell source file names (they end in `.hs` or `.lhs`). (We'll consider `this.is.a.file.hs` to be a good file name.)

2.3 Simple functions on lists (10pts)

Write a function `select :: (t -> Bool) -> [t] -> [a] -> [a]`, which takes a predicate and two lists as arguments and returns a list composed of elements from the second list in those positions where the predicate holds when applied to the element in the corresponding position of the first list. For example, `select even [1..26] "abcdefghijklmnopqrstuvwxyx"` evaluates to `"bdfhjlnprtvxz"`.

2.4 Simple functions on lists and numbers (10pts)

Write a function `prefixSum :: [Int] -> [Int]`, which takes a list of numbers as its argument and returns a list of sums of all prefixes of the list. For example, `prefixSum [1..10]` evaluates to `[1,3,6,10,15,21,28,36,45,55]`.

2.5 Simple functions on lists and numbers (10pts)

Write a function `numbers :: [Int] -> Int`, which takes a list of integers as its argument and returns the integer which has those numbers as digits. For example, `numbers [1..4]` evaluates to 1234.

2.6 Using lists for arithmetic: writing recursive functions over lists (50pts)

Numerals can be represented as lists of integers. For instance, decimal numerals can be expressed as lists of integers from 0 to 9. The integer 12345678901234567890 might be represented as the Haskell list `[1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0] :: [Int]`. However, the representation should allow a radix (base) other than 10 as well.

We use the following type abbreviation:

```
type Numeral = (Int, [Int])
```

where the first component of the pair is the radix and the second is the list of digits.

The above example number is then represented as:

```
example = (10, [1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0])
```

Write the following functions:

- (10pts) `makeLongInt :: Integer -> Int -> Numeral`, such that `makeLongInt n r` computes the list representation of the integer n in radix r . You can assume that $n \geq 0$, and that $r > 1$. For example, `makeLongInt 123 10` should evaluate to `(10, [1,2,3])`.
- (10pts) `evaluateLongInt :: Numeral -> Integer`, such that `evaluateLongInt (r, l)` converts a numeral back to a Haskell integer. You can assume that l is a valid list for radix r . For example, `evaluateLongInt (10, [1,2,3])` should evaluate to 123.

3. (10pts) `changeRadixLongInt :: Numeral -> Int -> Numeral`, such that `changeRadixLongInt n r` computes the representation of the same number as n in a new radix r . For example, `changeRadixLongInt (10, [1,2,3]) 8` should evaluate to `(8, [1,7,3])`; on the other hand, `changeRadixLongInt (10, [1,2,3]) 16` should evaluate to `(16, [7,11])`. The computation should be carried out without the use of Haskell's built-in Integer arithmetic.
4. (10pts) `addLongInts :: Numeral -> Numeral -> Numeral`, such that `addLongInts a b` computes the sum of the numbers given by the numerals a and b . If a and b use the same radix, that radix should be used for the result. If a and b use different radices, the result should use the larger one. For example, `addLongInts (10, [1,2,3]) (3, [1])` should evaluate to `(10, [1,2,4])`. The computation should be carried out without the use of Haskell's built-in Integer arithmetic.
5. (10pts) `mulLongInts :: Numeral -> Numeral -> Numeral`, such that `mulLongInts a b` computes the product of the numbers given by the numerals a and b . If a and b use the same radix, that radix should be used for the result. If a and b use different radices, the result should use the larger one. For example, `mulLongInts (10, [1,2,3]) (3, [1])` should evaluate to `(10, [1,2,3])`. The computation should be carried out without the use of Haskell's built-in Integer arithmetic. It is not permissible to implement the multiplication of a and b as $\sum_1^a b$.

How to turn in

Use the UNM Learn facility as follows: Navigate to <https://learn.unm.edu/> and log in. Then click on CS-357L-000 (Spring 2018). Now click on Assignments in the left side navigation menu. After that, click on the appropriate homework assignment link. Now attach your .hs file(s) and click submit. You are allowed to submit as many times as you like but only the latest submission will be graded.