Homework 5
CS 360
Summer 2021-2022

Geoffrey Mainland Patrick Brinich

Due: 11:59:59pm EDT
Friday, August 5, 2022

Assignment Link

Please accept the assigment on GitHub Classroom Here: `https://classroom.github.com/a/MnnM8gce`. In this assignment, you will implement the Luhn algorithm for validating credit card numbers and solve some additional small problems in Haskell.

You must implement the functions as specified. You may write other helper functions and define test data in your file, but you may not change the functions' names or the number or order of arguments.

You should complete the homework by modifying the file `HaskellIntro.hs` that we provide as part of your repository.Note: The first time you build your code, it will take some time, as all prerequisites will need to be automatically downloaded and installed. Subsequent builds will be much faster.

This assignment is worth 100 points. There are 105 possible points.

Homework Instructions:

You must complete this assignment on your own.

Problem 1: Implmenting the Luhn Algorithm (40 points total)

The Luhn algorithm is used to check the validity of credit card numbers. You can read about it on Wikipedia here. For this problem, you will implement the Luhn algorithm in Haskell. The algorithm encompasses the following steps:

1. Double the value of every second digit beginning *from the right*. That is, the last digit is unchanged; the second-to-last digit is doubled; the third-

to-last digit is unchanged; and so on. For example, `[1,3,8,6]` becomes `[2,3,16,6]`.

2. Add the digits of the doubled values and the undoubled digits from the original number. For example, `[2,3,16,6]` becomes $2+3+1+6+6 = 18$.
3. Calculate the remainder when the sum is divided by 10. For the above example, the remainder would be 8. If the result equals 0, then the number is valid.

Problem 1.1 (8 points)

We first need to be able to break up a number into its last digit and the rest of the number. Write these functions:

```
lastDigit :: Integer -> Integer
dropLastDigit :: Integer -> Integer
```

If you're stumped, look through some of the arithmetic operators mentioned in the lecture. You may not use lists or strings. This should remind you of the binary problem from the first homework...

Example output:

```
GGHCi, version 8.10.7: http://www.haskell.org/ghc/  :? for help
Prelude> :load HaskellIntro.hs
[1 of 2] Compiling Set              ( Set.hs, interpreted )
[2 of 2] Compiling HaskellIntro     ( HaskellIntro.hs, interpreted )
Ok, two modules loaded.
*HaskellIntro> lastDigit 123
3
*HaskellIntro> lastDigit 0
0
*HaskellIntro> dropLastDigit 123
12
*HaskellIntro> dropLastDigit 5
0
*HaskellIntro>
```

Problem 1.2 (8 points)

Now, we can break apart a number into its digits. Define the function:

```
toDigits :: Integer -> [Integer]
```

which should convert positive Integers to a list of digits. For 0 or negative inputs, `toDigits` should return the empty list.
Examples:

```
toDigits 1234 == [1,2,3,4]
toDigits 0 == []
toDigits (-17) == []
```

Problem 1.3 (8 points)

Once we have the digits in the proper order, we need to double every other one. Define a function

```
doubleEveryOther :: [Integer] -> [Integer]
```

Remember that `doubleEveryOther` should double every other number beginning from the right, that is, the second-to-last, fourth-to-last, ... numbers are doubled. Note that it's much easier to perform this operation on a list of digits that's in reverse order. You will likely need helper functions to make this work. Examples:

```
doubleEveryOther [8,7,6,5] == [16,7,12,5]
doubleEveryOther [1,2,3] == [1,4,3]
```

Problem 1.4 (8 points)

The output of `doubleEveryOther` has a mix of one-digit and two-digit numbers. Define the function

```
sumDigits :: [Integer] -> Integer
```

to calculate the sum of all digits. Example:

```
sumDigits [16,7,12,5] = 1 + 6 + 7 + 1 + 2 + 5 = 22
```

Problem 1.5 (8 points)

Define the function

```
validate :: Integer -> Bool
```

that indicates whether an Integer could be a valid credit card number. This will use all functions defined in the previous exercises. Examples:

```
validate 4012888888881881 = True
validate 4012888888881882 = False
```

Problem 2: Hofstadter Sequences (40 points total)

In this problem you will use a higher-order function to compute two Hofstadter Sequences, the Hofstadter G sequence and the Hofstadter H sequence.

We will represent sequences not as lists, bus as functions that take an index into the sequence and return the value of the sequence at that index. Sequences will be indexed from 0, so $G(0)$ will be the first item in the Hofstadter G sequence.

Problem 2.1 (20 points)

Write a higher-order function

```
pow f n
```

that computes the function f^n, where $f^n(x) = (f \circ \cdots \circ f)(x)$. That is, f^n is the function that composes the function f with itself n times. For example, given the following definition:

```
square x = x*x
```

`pow square 2` would be the function `square . square`, where . is the function composition operator demonstrated in class. For any number $n, n^0 = 1$. Analogously, for any function f, f^0 is the identity function.

Problem 2.2 (10 points)

Write functions g and h that compute elements of the Hofstadter G and Hofstadter H sequences, respectively. They are defined as follows

$$\begin{aligned} G(0) &= 0 \\ G(n) &= n - G(G(n-1)) \qquad\qquad n > 0 \end{aligned}$$

$$\begin{aligned} H(0) &= 0 \\ H(n) &= n - H(H(H(n-1))) \qquad\qquad n > 0 \end{aligned}$$

You must define g and h such that all recursive calls to g and h occur indirectly via `pow`, i.e., you may not recursively call g or h directly or through a helper function that is not `pow`.
The first few numbers in the G sequence are 0,1,1,2,3,3,4,4,5,6,6,7,8,8,9,9,10,11,11,12,12.
The first few numbers in the H sequence are 0,1,1,2,3,4,4,5,5,6,7,7,8,9,10,10,11,12,13,13,14

Problem 2.3 (10 points)

The G and H sequences can be generalized to a family of sequences, D_i:

$$\begin{aligned} D_i(0) &= 0 \\ D_i(n) &= n - D_i^i(n-1) \qquad\qquad n > 0 \end{aligned}$$

Observe that $G = D_2$ and $H = D_3$. Write a function `d i` that computes elements of the D_i sequence. Note that D_i^i is just $\underbrace{D_i \circ \ldots \circ D_i}_{i \text{ instances of } D_i}$ You must define `d i` using `pow`.
The first few numbers in the D_4 sequence are 0,1,1,2,3,4,5,5,6,6,7,8,8,9,10,11,11,12,13,14,15.
Hint: The function d can be partially applied to give a function that compute the elements of a particular sequence D_i. For example, the Haskell functions `d 2` and `g` should be equivalent at every non-negative argument.

Problem 3: Power set of a set (20 points)

Implement the power set function, $\mathcal{P}(\cdot)$, from lecture. Recall that the power set $\mathcal{P}(S)$ of a set S is the set of all subsets of S, including the empty set and S itself. Also recall the inductive proof that a set with n elements has a power set with 2^n elements.

The type signature for `powerSet` should look vaguely like this:

```
powerSet :: ... => ... -> ...
```

Before you write the function, you should figure out what its type signature should be. If you want to start by specializing the function to only work with sets of a particular type, like Int, that's OK, but your final type signature should be polymorphic.

One of the goals of this problem is to treat `Set` as an abstract data type; we should be able to change the representation of sets, keeping the external interface the same, and the code you wrote to compute the power set would run unchanged. This means that you cannot rely in any way on the fact that sets happen to be implemented as sorted lists with no duplicates. How then can you test for an empty set? Fortunately, there is an `isEmpty` predicate exported by the `Set` module. How can you split a set into an element plus the rest of the set? Again, we fortunately have a `split` function exported by the `Set` module.

If you are stuck trying to pattern match on the structure of a set, stop! You can't do that, since you don't get to see the representation! Instead, use guards, or, if you prefer, just use Haskell's if. Please note the following constraints:

1. You may not change the code in Set.hs.
2. Solutions that themselves use `fromList` or `toList` will receive zero credit. You must use the `Set` abstraction without relying on the fact that "under the covers" it is implemented using lists. However, you may use `fromList` and `toList` to test your code!
3. You may use any function that is exported by the Set module except `fromList` or `toList`, even if it uses `fromList` or `toList` internally. Think of the Set module as a black box whose implementation you are not allowed to see.
4. You may not write your own functions that convert between Sets and lists in order to bypass the restriction on using `fromList` and `toList`.

In essence use sets, not lists to solve this problem.

Problem 4: Homework Statistics (5 Points Extra Credit)

How long did spend on each problem? Please tell us in your `README.md`.

You may enter time using any format described here. Please enter times using the requested format, as that will make it easy for us to automatically collect.

We are happy to read additional comments you have, but do not put any comments on the same line as the time report. Please do not otherwise edit, move, or reformat the lines reserved for time statistics reports. Here is the reporting format you should use:

```
Problem 1:  15m
Here's a description of my experience solving this problem.
```

If you did not attempt one or more problems, you can still receive full marks by telling us you spent 0 minutes on these problems. You will not receive points for this problem if you leave any of the time reports blank.

Course Software on TUX

1. Append the following line to your `.bash_profile` file (in your home directory) on TUX to use the correct versions of DrRacket and GHC:

   ```
   source ~mainland/courses/software/cs360.sh
   ```

2. Log out of your tux account and then log back in.

3. Use the `which -a` command to verify everything is set up:

   - `which -a racket` should output something like `/home/mainland/courses/software/bin/racket`
   - `which -a ghc` should output something like `/home/mainland/courses/software/bin/ghc`

Development outside of TUX

You're free to develop your solutions in any environment, but please make sure your solutions run on TUX.

Docker containers for VS Code

Every homework repository is set up to work with development containers if you use Visual Studio Code. If you install Docker, then the "Remote - Containers" extension can automatically launch an appropriate Docker container and re-open your repository in the container, giving you a Linux environment in which to work no matter what computer you are using. See the Development Containers in Education post on the Visual Studio Code blog for more details. To use development containers:

1. Install Docker Desktop:
   `https://www.docker.com/products/docker-desktop`

2. Install Visual Studio code.

3. Install the "Remote - Containers" extension in VS Code.

4. Clone your repository as usual.

5. Open your cloned repository in Visual Studio Code. It should ask you if you want to re-open it in a container—click "Reopen in Container." At any time, you can also open the command palette in VS Code and run the command "Remote-Containers: Open Workspace in Container" to re-open your workspace in a container.