CptS355 - Assignment 1 (Haskell)
Spring 2021

Assigned: Tuesday February 2, 2021

Due: Wednesday February 10, 2021

Weight: Assignment 1 will count for 6% of your course grade.

Your solutions to the assignment problems are to be your own work. Refer to the course academic integrity statement in the syllabus.

This assignment provides experience in Haskell programming. Please compile and run your code on command line using Haskell GHC compiler. You may download GHC at https://www.haskell.org/platform/.

Turning in your assignment

The problem solution will consist of a sequence of function definitions and unit tests for those functions. You will write all your functions in the attached `HW1.hs` file. You can edit this file and write code using any source code editor (Notepad++, Sublime, Visual Studio Code, etc.). We recommend you to use Visual Studio Code, since it has better support for Haskell.

In addition, you will write unit tests using `HUnit` testing package. Attached file, `HW1SampleTests.hs`, includes 2 to 4 sample unit tests for each problem. You will edit this file and provide additional tests (add at least 2 tests per problem). Please rename your test file as `HW1Tests.hs`.

The instructor will show how to import and run tests on `GHCI` during the lecture. Please refer to the lecture video (February 3) on Canvas.

To submit your assignment, please upload both files (`HW1.hs` and `HW1Tests.hs`) on the Assignment1 (Haskell) DROPBOX on Canvas (under Assignments). You may turn in your assignment up to 3 times. Only the last one submitted will be graded.

The work you turn in is to be **your own personal work**. You may not copy another student's code or work together on writing code. You may not copy code from the web, or anything else that lets you avoid solving the problems for yourself. **At the top of the file in a comment, please include your name and the names of the students with whom you discussed any of the problems in this homework**. This is an individual assignment and the final writing in the submitted file should be *solely yours*.

Important rules

- Unless directed otherwise, you must implement your functions using recursive definitions built up from the basic built-in functions. (You are not allowed to import an external library and use functions from there.)
- The type of your functions should be compatible with the type specified in each problem. You don't need to include the "type signatures" for your functions.
- Make sure that your function names match the function names specified in the assignment specification. Also, make sure that your functions work with the given tests**.** However, the given test inputs don't cover all boundary cases. You should generate other test cases covering the

extremes of the input domain, e.g. maximum, minimum, just inside/outside boundaries, typical values, and error values.

- When auxiliary functions are needed, make them local functions (inside a `let..in` or `where` block). In this homework you will lose points if you don't define the helper functions inside a `let..in` or `where` block.
- Be careful about the indentation. The major rule is "*code which is part of some statement should be indented further in than the beginning of that expression*". Also, "*if a block has multiple statements, all those statements should have the same indentation*". Refer to the following link for more information: https://en.wikibooks.org/wiki/Haskell/Indentation
- The assignment will be marked for good programming style (indentation and appropriate comments), as well as clean compilation and correct execution. Haskell comments are placed inside properly nested sets of opening/closing comment delimiters:
  ```
  {- multi line
  comment-}
  ```
  .
  Line comments are preceded by double dash, e.g., `-- line comment`

Problems

1. `getUniqueRight` and `getUniqueLeft` – 20%

a) **[10pts]** Define a recursive function `getUniqueRight` which takes a list as input and it returns the unique values from the list. Your function should keep the last (i.e., rightmost) copies of the duplicate elements in the result. It should not sort the elements in the input list. You may use the `elem` function in your implementation.
The type of the `getUniqueRight` function should be compatible with the following:

```
getUniqueRight:: Eq a => [a] -> [a]
```

Examples:

```
> getUniqueRight [6,5,1,6,4,2,2,3,7,2,1,1,2,3,4,5,6,7]
[1,2,3,4,5,6,7]

> getUniqueRight "CptS322 - CptS322 - CptS 321"
"-CptS 321"

> getUniqueRight [[1,2],[1],[],[3],[1],[]]
[[1,2],[3],[1],[]]

> getUniqueRight [('a',1),('a',2),('b',1),('a',2),('c',1),('b',1),('a',2)]
[('a',1),('c',1),('b',1),('a',2)]

> getUniqueRight ["Let","it","snow","let","it","rain","let","it","hail"]
["Let","snow","rain","let","it","hail"]
```

b) **[10pts]** Re-write your `getUniqueRight` function so that it keeps the first (i.e., leftmost) copies of the duplicate values in the output instead of the last ones. Name your function `getUniqueLeft`. Your solution should not sort the elements in the input list. You may use the `elem` function in your implementation.
The type of the `getUniqueLeft` function should be compatible with the following:

```
getUniqueLeft:: Eq a => [a] -> [a]
```

Examples:

```
> getUniqueLeft [6,5,1,6,4,2,2,3,7,2,1,1,2,3,4,5,6,7]
[6,5,1,4,2,3,7]

> getUniqueLeft "CptS322 - CptS322 - CptS 321"
"CptS32 -1"

> getUniqueLeft [[1,2],[1],[],[3],[1],[]]
[[1,2],[1],[],[3]]

> getUniqueLeft [('a',1),('a',2),('b',1),('a',2),('c',1),('b',1),('a',2)]
[('a',1),('a',2),('b',1),('c',1)]

> getUniqueLeft ["Let","it","snow","let","it","rain","let","it","hail"]
["Let","it","snow","let","rain","hail"]
```

2. cansInLog and numCans and getMonths – 40%

Assume your cat gained a lot of weight since the beginning of the pandemic. Your veterinarian recommended you keep track of the number of cat food cans you feed to your pet every month. In addition, they recommended you balance their diet and avoid giving the same types of flavors. So, you start recording how many cans you feed to your cat in a Haskell list. This list logs the number of cans your cat consumed during that month for each flavor.

a) (10 pts) First consider the food you give to your cat during a single month, for example:
`feedinglog =[("Oceanfish",7),("Tuna",1),("Whitefish",3),("Chicken",4),("Beef",2)]`
`feedinglog` is a list of *(flavor, number of cans)* pairs.

Write a function `cansInLog` that takes a month's feeding log as input and returns the total number of cans consumed. The type for `countInLog` should be compatible with the following:
`cansInLog :: Num p => [(a, p)] -> p`

Examples:
```
> cansInLog [("Oceanfish",7),("Tuna",1),("Whitefish",3),("Chicken",4),("Beef",2)]
17
> cansInLog [("Oceanfish",3),("Tuna",2),("Whitefish",2),("Salmon",2),("Chicken",4),
("Beef",2),("Turkey",1),("Sardines",3)]
19
```

b) (15 pts) Now we combine the feeding logs for all months into one single list. An example list is given below:

```
myCatsLog=[((7,2020),[("Oceanfish",7),("Tuna",1),("Whitefish",3),("Chicken",4),("Beef",2)]),
 ((8,2020),[("Oceanfish",6),("Tuna",2),("Whitefish",1),("Salmon",3),("Chicken",6)]),
 ((9,2020),[("Tuna",3),("Whitefish",3),("Salmon",2),("Chicken",5),("Beef",2),("Turkey",1),("Sardines",1)]),
 ((10,2020),[("Whitefish",5),("Sardines",3),("Chicken",7),("Beef",3)]),
 ((11,2020),[("Oceanfish",3),("Tuna",2),("Whitefish",2),("Salmon",2),("Chicken",4),("Beef",2),("Turkey",1)]),
 ((12,2020),[("Tuna",2),("Whitefish",2),("Salmon",2),("Chicken",4),("Beef",2),("Turkey",4),("Sardines",1)]),
 ((1,2021),[("Chicken",7),("Beef",3),("Turkey",4),("Whitefish",1),("Sardines",2)])  ]
```

The list includes tuples where the first value in each tuple is the timestamp representing a month and the second value is the feeding log for that month. The timestamp itself is a tuple including the month and year. For example `(8,2020)` represents `August 2020.`

Write a function, numCans, that takes your cat's food log (similar to the above) and a year value (for example `2020`) and returns the total number of cans that your cat consumed in that year.
(Hint: You can make use of `cansInLog` function you defined in part-a.)
The type of numCans should be compatible with the following:
`numCans :: (Num p, Eq t) => [((a1, t), [(a2, p)])] -> t -> p`

Examples:
```
> numCans myCatsLog 2020
103
> numCans myCatsLog 2021
17
```

b) (15 pts) You would like to make sure that your cat's diet includes a balanced portion of fish, poultry and meat. So, you should check what months you fed your cat more than a certain number of cans of a certain flavor.
Write a function called `getMonths` that takes a copy of your cat's food log (similar to the above), a number "n", and a cat food flavor (e.g., "chicken, "oceanfish", etc.) and returns the list of months whose feeding logs include more than "n" cans of the given flavor.

The type of `getMonths` should be compatible with the following:

```
getMonths :: (Ord t1, Eq t2) =>
             [((a, b), [(t2, t1)])] -> t1 -> t2 -> [(a, b)]
```

Examples:

```
> getMonths myCatsLog 4 "Oceanfish"
[(7,2020),(8,2020)]
> getMonths myCatsLog 5 "Chicken"
[(8,2020),(10,2020),(1,2021)]
```

3. deepCount – 15%

Write a function `deepCount` that takes a value "v" and a nested list (i.e., list of lists) and returns the number of occurrences of "v" in the nested list.

The type of `deepCount` should be compatible with the following:

```
deepCount :: (Num p, Eq t) => t -> [[t]] -> p
```

Examples:

```
> deepCount 5 [[1,2,3],[5,5],[4,5,6],[7,1,2,3,4,5],[5,6]]
5
> deepCount "a" [["a","b","c"],["b","c"],["b","e","g"],["a","h","c","d"],["d"],[],
["h","a"]]
3
> deepCount 10 [[1,2,3],[1,2],[4,5,6],[7,1,2,3,4,5],[1],[], [5,6]]
0
> deepCount 1 []
0
```

4. clusterConsequtive - 25%

Write a function `clusterConsecutive` that takes a list of numbers as input and it splits the input list into groups of numbers (sublists) where each group includes increasingly consecutive numbers. The split points will be when the adjacent elements are not consecutive or when the numbers decrease.

The type of `clusterConsecutive` should be compatible with the following:

```
clusterConsecutive :: (Eq a, Num a) => [a] -> [[a]]
```

Examples:

```
> clusterConsecutive [1,2,3,5,6,7,8,9,2,3,11,12]
[[1,2,3],[5,6,7,8,9],[2,3],[11,12]]
> clusterConsecutive [1,3,5,7,8,10,13]
```

```
[[1],[3],[5],[7,8],[10],[13]]
> clusterConsecutive [1]
[[1]]
> clusterConsecutive [2,1]
[[2],[1]]
> clusterConsecutive []
[]
```

(5%) Testing your functions

Install HUnit

We will be using the HUnit unit testing package in CptS355. See http://hackage.haskell.org/package/HUnit for additional documentation.

To install HUnit **using cabal installer, r**un the following commands on the terminal (command line).

```
$ cabal update
$ cabal v1-install HUnit
```

Check the attached HUnit_HowtoInstall.pdf document for other options to install HUnit.

Running Tests

The file HW1SampleTests.hs provides 2 to 4 sample test cases comparing the actual output with the expected (correct) output for each problem. This file imports the HW1 module (HW1.hs file) which will include your implementations of the given problems.

You are expected to add **at least 2 more test cases** for each problem. Make sure that your test inputs cover all boundary cases.

In HUnit, you can define a new test case using the TestCase function and the list TestList includes the list of all test that will be run in the test suite. So, make sure to add your new test cases to the TestList list. All tests in TestList will be run through the "runTestTT tests" command. The instructor will further explain this during the lecture.

If you don't add new test cases you will be deduced at least 5% in this homework.

Haskell resources:

- **Learning Haskell**, by Gabriele Keller and Manuel M T Chakravarty (http://learn.hfm.io/)
- **Real World Haskell**, by Bryan O'Sullivan, Don Stewart, and John Goerzen (http://book.realworldhaskell.org/)
- **Haskell Wiki:** https://wiki.haskell.org/Haskell
- **HUnit**: http://hackage.haskell.org/package/HUnit