CptS355 - Assignment 1 (Haskell)
Spring 2023

Weight: Assignment 1 will count for 7% 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 GHCI. You may download Haskell Platform 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 (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. You will write your tests in the file **`HW1tests.hs`** – the template of this file is available on the HW1 assignment page. You will edit this file and provide additional tests (add at least 2 tests per problem). The instructor will show how to import and run tests on `GHCI` during the lecture.

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.)
- 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. <u>Also, make sure that your functions work with the given tests.</u> 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. (a) `count – 6%`

Write a function `count` which takes a value and a list and returns the number of occurrences of that value in the input list.

Examples:

```
> count '5' "355-451"
3

> count [] [[],[1,2],[3,2],[5,6,7],[8],[]]
2

> count 0 [1,2,3,4,5,6,7,8,9]
0
```

1. (b) `diff – 6%`

Write a function `diff` that takes two lists as input and returns the difference of the first list with respect to the second. The input lists may have duplicate elements. If an element in the first list also appears in the second one, the element – and its duplicate copies – should be excluded in the output.

The elements in the resulting list may have arbitrary order.

Examples:

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

> diff [1,2,2,3,3,3,6,7,4,4,4,4,5,5,5,5,5,6,6] [1,1,2,2,3,3,4,4,5,6,6,6]
[7]

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

1. (c) `bag_diff – 8%`

Write a function `diff` that takes two lists as input and returns the difference of the first list with respect to the second. The input lists may have duplicate elements. If an element appears in both lists and if the number of duplicate copies of the element is bigger in the first list, then this element should appear in the result as many times as the difference of the number of occurrences in the input lists.

The duplicates <u>should not</u> be eliminated in the result. The elements in the resulting list may have arbitrary order. (*Hint:* You can make use of `count` function in your solution.)

Examples:

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

> diff [1,2,2,3,3,3,6,7,4,4,4,4,5,5,5,5,5,6,6] [1,1,2,2,3,3,4,4,5,6,6,6]
[3,4,4,5,5,5,5,7]

> diff [6,2,2,3,5,3,6,7,4,4,5,4,5,5,4,5,3,1,6] [1,2,2,3,3,3,6,7,4,4,4,4,5,5,5,5,5,6,6]
[]

> diff "testing my function" "fit "
"testing myuncon"
```

2. everyN – 10%

The function `everyN` takes a list and a number '**n**' (representing a count) and returns every **n**th value in the input list. For example,

```
          123123123123123123123
   everyN "-hH-aA-sS-kK-eE-lL-lL" 3
```

will return the characters every 3 elements, i.e., "HASKELL".

Examples:

```
> everyN [1,2,3,4,5,6,7,8,9,10,11,12,13] 3
[3,6,9,12]

> everyN "hHaAsSkKeElLlL" 2
"HASKELL"

> everyN "haskell" 1
"haskell"

> everyN [] 5
[]
```

3. make_sparse & compress

A ***sparse vector*** is a vector having a relatively small number of nonzero elements. When a sparse vector is saved, it is typically put in storage without its zero elements. A possible solution for storing sparse vector is ***compressing*** it as a list of tuples where the tuples store the indices and values for non-zero elements. For example,
`[0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1]` can be stored as: `[(3,1),(20,1)]`

Assume the indexes are 0-based.

3.(a) make_sparse – 15%

Write a function `make_sparse` which takes a compressed vector value (represented as a Haskell list of tuples) and returns the equivalent sparse vector (including all `0` values).

Examples:

```
> make_sparse [(3,30),(10,100),(11,110)]
[0,0,0,30,0,0,0,0,0,0,100,110]
```

```
> make_sparse [(1,1),(2,2),(4,4),(6,6),(9,9)]
[0,1,2,0,4,0,6,0,0,9]
```

```
> make_sparse [(20,1)]
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1]
```

```
> make_sparse []
[]
```

3.(b) compress – 15%

Write a function `compress` which takes a sparse vector value (represented as a Haskell list) and returns the equivalent compressed values as a list of tuples.

Examples:

```
> compress [0,0,0,30,0,0,0,0,0,0,100,110]
[(3,30),(10,100),(11,110)]
```

```
> compress [0,1,2,0,4,0,6,0,0,9]
[(1,1),(2,2),(4,4),(6,6),(9,9)]
```

```
> compress [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1]
[(20,1)]
```

```
> compress []
[]
```

4. added_sums – 8 %

Write a function `added_sums` that takes a list of numbers and returns a list including the cumulative partial sums of these numbers.
(*Hint*: Define and use a helper function that takes a list and a number holding the accumulated sum.)

Examples:

```
> added_sums [1,2,3,4,5,6,7,8,9,10]
[1,3,6,10,15,21,28,36,45,55]
```

```
> added_sums [0,-2,3,4,-4,-1,2]
[0,-2,1,5,1,0,2]
```

```
> added_sums []
[]
```

5. find_routes - 8%

Pullman Transit offers many bus routes in Pullman. Assume that they maintain the bus stops for their routes as a list of tuples. The first element of each tuple is the bus route and the second element is the list of stops for that route (see below for an example).

```
routes =
   [("Lentil",["Chinook", "Orchard", "Valley", "Emerald", "Providence", "Stadium",
               "Main", "Arbor", "Sunnyside", "Fountain", "Crestview", "Wheatland",
               "Walmart", "Bishop", "Derby", "Dilke"]),
    ("Wheat",["Chinook", "Orchard", "Valley", "Maple","Aspen", "TerreView", "Clay",
              "Dismores", "Martin", "Bishop", "Walmart", "PorchLight", "Campus"]),
   ("Silver",["TransferStation", "PorchLight", "Stadium", "Bishop","Walmart",
              "Outlet", "RockeyWay","Main"]),
   ("Blue",  ["TransferStation", "State", "Larry", "TerreView","Grand", "TacoBell",
              "Chinook", "Library"]),
   ("Gray",  ["TransferStation", "Wawawai", "Main", "Sunnyside","Crestview",
              "CityHall", "Stadium", "Colorado"]),
   ("Coffee",["TransferStation", "Grand", "Main", "Visitor","Stadium", "Spark",
              "CUB"])]
```

Assume that you are creating an application for Pullman Transit. You would like to write an Haskell function `find_routes` that takes the list of bus routes and a stop name, and returns the list of the bus routes which stop at the given bus stop.
(Hint: You can make use of `elem` function in your solution.)

Examples:

```
> find_routes "Walmart" routes_test
["Lentil","Wheat","Silver"]

> find_routes "Rosauers" routes_test
[]

> find_routes "Main" routes_test
["Lentil","Silver","Gray","Coffee"]
```

6. group_sum – 15 %

`group_sum` function takes two arguments where the first argument is a list (`lst`) and the second argument is an integer (`n`). The goal is to produce a result in which the elements of the original list have been collected into ordered sub-lists each containing maximum number of consecutive elements from `lst` summing up to or less than `n*2^k` (where is `k` the group number starting at `0`, i.e., `k=0,1,2,3,4…`). The leftover elements (if there are any) are included in the last sub-list with a sum less than `n*2^k`. When elements are added to the groups, if the next element in the input list (`lst`) is greater than `n*2^k`, the group will be empty list `[]`.

For example,

```
group_sum [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17] 10
```

will evaluate to,

```
[[1,2,3,4],[5,6,7],[8,9,10,11],[12,13,14,15,16],[17]]
```

```
   n=10          n=10          n=10           n=10           n=10
   k=0           k=1           k=2            k=3            k=4
 elements      elements      elements       elements       elements
  should        should        should         should         should
add up to     add up to     add up to      add up to      add up to
10*2^0=10     10*2^1=20     10*2^2=40      10*2^3=80      10*2^4=160
```

Examples:

```
> group_sum [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17] 10
[[1,2,3,4],[5,6,7],[8,9,10,11],[12,13,14,15,16],[17]]

> group_sum [12,10,1,3,4,7,11,22,2,5,40,100,4] 10
[[],[12],[10,1,3,4,7,11],[22,2,5,40],[100,4]]

> group_sum [5,-2,-3,4,-5,6,7,8,9,10,11,12,-13,14,15,16,20] 4
[[],[5,-2,-3,4,-5,6],[7,8],[9,10,11],[12,-13,14,15,16,20]

> group_sum [] 3
[]
```

Assignment rules – 3%

Make sure that your assignment submission complies with the following. :

- The module name of your `HW1.hs` files should be `HW1`. Please don't change the module name in your submission file.
- The function names in your solutions should match the names in the assignment prompt. Running the given tests will help you identify typos in function names.
- Make sure to remove all test data from the `HW1.hs` file, e.g. , the '`routes`' list for Problem-5.
- Make sure to define your helper functions inside a let..in or where block.

Testing your functions – 6%

You are expected to add **at least 2 more test cases** for each problem. **Write your tests for all problems in `HW1_tests.hs` file** – the template of this file is provided to you in the HW1 assignment page. Make sure that your test inputs cover all boundary cases. Also, your test inputs should not be same or very similar to the given sample tests.

Note : For problem 5, it is sufficient to provide one additional test case; make sure to use different ***list input*** *in your test. For all other problems, please give two tests.*

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.

Running Tests

The `HW1SampleTests.zip` file includes 5 `.hs` files where each one includes the HUnit tests for a different HW problem. The tests compare the actual output to the expected (correct) output and raise an exception if they don't match. The test files import the `HW1` module (`HW1.hs` file) which will include your implementations of the HW problems.
You will write your solutions to `HW1.hs` file. To test your solution for each HW problem run the following commands on the command line (i.e., terminal):

```
$ ghci
$ :l P1_HW1tests.hs
P1_HW1tests> run
```

Repeat the above for other HW problems by changing the test file name, i.e. , `P2_HW1tests.hs, P3_HW1tests.hs, etc.`

To run your own test file run the following command on the GHCI prompt:

```
$ :l HW1tests.hs
HW1tests> run
```

If you don't add new test cases or if your tests are very similar to the given tests, you will be **deduced 6% 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