CptS355 - Assignment 1 (Haskell)
Fall 2022

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. 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. `merge_sorted` – 10%

a) **[10pts]** Define a recursive function `merge_sorted` which takes two **sorted** lists as input, merges the elements from the two lists, and returns a sorted merged list. You can assume that the input lists are already sorted in ascending order. You are not allowed to use the built-in '`sort`' function of Haskell in your solution.
The type of the `merge_sorted` function should be compatible with the following:

```
merge_sorted :: Ord a => [a] -> [a] -> [a]
```

Examples:

```
> merge_sorted [5,8,9,15] [1,2,8,9,11,13,15]
[1,2,5,8,8,9,9,11,13,15,15]

> merge_sorted "dfgkmn" "abhijk"
"abdfghijkkmn"

> merge_sorted [1,2,3,4,5] [5,6,7,8]
[1,2,3,4,5,5,6,7,8]

> merge_sorted [1,2,3] []
[1,2,3]
```

2. `sum_range` – 15%

The function `sum_range` takes a tuple (representing an index range) and a list and sums all values in the list for the given index range.
For example, `sum_range (3,7) [0,1,2,3,4,5,6,7,8,9]` will sum all values in the given list from index 3 to 7 (inclusive), i.e., `3+4+5+6+7 = 25`
Given a range tuple (e.g., `(a,b)`), you can assume that the lower bound of the range is always less than or equal to the upper bound, i.e., a<=b. And both bounds are lower than the length of the input list, i.e., `a <= b < length of the list.` Note that both a and b are 0-based indexes.

The type of the `sum_range` function should be compatible with one of the following (depending on the comparison you apply on the range boundaries the type of your function may be different):

```
sum_range :: (Ord a, Num p, Num a) => (a, a) -> [p] -> p
sum_range :: (Num p, Num a) => (a, a) -> [p] -> p
```

Examples:

```
> sum_range (3,7) [0,1,2,3,4,5,6,7,8,9]
25

> sum_range (0,5) [0,1,2,3,4,5,6,7,8,9]
15

> sum_range (7,9) [0,1,2,3,4,5,6,7,8,9]
24

> sum_range (7,7) [0,1,2,3,4,5,6,7,8,9]
7
```

3. calc_collatz_seq and longest_collatz_seq

Collatz sequence is an iterative sequence which is defined as follows:
(https://projecteuler.net/problem=14)
Given a positive number value n,

n → n/2 (n is even)
n → 3n + 1 (n is odd)

Using the rule above and starting with 13, we generate the following sequence:
13 → 40 → 20 → 10 → 5 → 16 → 8 → 4 → 2 → 1

(a) calc_collatz_seq – 10%

Write a function `calc_collatz_seq` which takes a number value (e.g., n) and returns the Collatz sequence starting at the number `n`, as a Haskell list.
The type of the `calc_collatz_seq` function should be compatible with the following:

```
calc_collatz_seq :: Integral a => a -> [a]
```

Examples:

```
> calc_collatz_seq 9
[9,28,14,7,22,11,34,17,52,26,13,40,20,10,5,16,8,4,2,1]

> calc_collatz_seq 27
[27,82,41,124,62,31,94,47,142,71,214,107,322,161,484,242,121,364,182,91,274,137,412,2
06,103,310,155,466,233,700,350,175,526,263,790,395,1186,593,1780,890,445,1336,668,334
,167,502,251,754,377,1132,566,283,850,425,1276,638,319,958,479,1438,719,2158,1079,323
8,1619,4858,2429,7288,3644,1822,911,2734,1367,4102,2051,6154,3077,9232,4616,2308,1154
,577,1732,866,433,1300,650,325,976,488,244,122,61,184,92,46,23,70,35,106,53,160,80,40
,20,10,5,16,8,4,2,1]

> calc_collatz_seq 13
[13,40,20,10,5,16,8,4,2,1]

> calc_collatz_seq 11
[11,34,17,52,26,13,40,20,10,5,16,8,4,2,1]
```

(b) **`longest_collatz_seq` – 15%**

Write a function `longest_collatz_seq` which takes a number value (e.g., n) and returns the longest Collatz sequence among all numbers between 1 and n.
For example, given n = 50, among all numbers between 1 and 50, 27 has the longest Collatz sequence with 112 elements. So, `longest_collatz_seq 50` will return the Collatz sequence of 27.
The type of the `longest_collatz_seq` function should be compatible with the following:

```
longest_collatz_seq :: Integral a => a -> [a]
```

Hint: Write a helper function that returns the longer of the two given lists and use it in your solution.

Examples:

```
> longest_collatz_seq 10
[9,28,14,7,22,11,34,17,52,26,13,40,20,10,5,16,8,4,2,1]
> longest_collatz_seq 50
[27,82,41,124,62,31,94,47,142,71,214,107,322,161,484,242,121,364,182,91,274,137,412,2
06,103,310,155,466,233,700,350,175,526,263,790,395,1186,593,1780,890,445,1336,668,334
,167,502,251,754,377,1132,566,283,850,425,1276,638,319,958,479,1438,719,2158,1079,323
8,1619,4858,2429,7288,3644,1822,911,2734,1367,4102,2051,6154,3077,9232,4616,2308,1154
,577,1732,866,433,1300,650,325,976,488,244,122,61,184,92,46,23,70,35,106,53,160,80,40
,20,10,5,16,8,4,2,1]
> longest_collatz_seq 75
[73,220,110,55,166,83,250,125,376,188,94,47,142,71,214,107,322,161,484,242,121,364,18
2,91,274,137,412,206,103,310,155,466,233,700,350,175,526,263,790,395,1186,593,1780,89
0,445,1336,668,334,167,502,251,754,377,1132,566,283,850,425,1276,638,319,958,479,1438
,719,2158,1079,3238,1619,4858,2429,7288,3644,1822,911,2734,1367,4102,2051,6154,3077,9
232,4616,2308,1154,577,1732,866,433,1300,650,325,976,488,244,122,61,184,92,46,23,70,3
5,106,53,160,80,40,20,10,5,16,8,4,2,1]
```

4. game_scores and wins_by_year

The following dictionary stores WSU's college football game scores for the past 4 years as a list of tuples. The first element of each tuple is the year and the second element is the list of games in that year. Each game is a tuple including the opponent team's name and the score tuple i.e., (WSU's score , opponent team's score).

```
wsu_games = [
    (2019, [("NMSU",(58,7)), ("UNCO",(59,17)), ("HOU",(31,24)), ("UCLA",(63,67)),
            ("UTAH",(13,38)), ("ASU",(34,38)), ("COLO",(41,10)), ("ORE",(35,37)),
            ("CAL",(20,33)), ("STAN",(49,22)), ("ORST",(54,53)), ("WASH",(13,31)),
            ("AFA",(21,31))]),
    (2020, [("ORST",(38,28)), ("ORE",(29,43)), ("USC",(13,38)), ("UTAH",(28,45))]),
    (2021, [("USU",(23,26)), ("PORT ST.",(44,24)), ("USC",(14,45)), ("UTAH",(13,24)),
            ("CAL",(21,6)), ("ORST",(31,24)), ("STAN",(34,31)), ("BYU",(19,21)),
            ("ASU",(34,21)), ("ORE",(24,38)), ("ARIZ",(44,18)), ("WASH",(40,13)),
            ("CMU",(21,24))] )
            ]
```

(a) game_scores – 15%

Write a Haskell function `game_scores` that takes the game list (similar to `wsu_games` above) and an opponent team name (e.g., "USC") as input and returns the list of the game scores that WSU played against the given opponent team.

The type of the `game_scores` function should be compatible with the following:

```
game_scores :: Eq t => [(a1, [(t, a2)])] -> t -> [a2]
```

Examples:

```
> game_scores wsu_games "USC"
[(13,38),(14,45)]

> game_scores wsu_games "ORST"
[(54,53),(38,28),(31,24)]

> game_scores wsu_games "YALE"
[ ]
```

(b) wins_by_year – 10%

Assume you would like to find the number of games WSU won each year. Write a function "`wins_by_year`" that takes the WSU game data as input, and it returns a list of tuples where each tuple includes the year and the number wins (of WSU team) in that year.

The type of the `wins_by_year` function should be compatible with the following:

```
wins_by_year :: (Num b, Ord a1) => [(a2, [(a3, (a1, a1))])] -> [(a2, b)]
```

Example:

```
> wins_by_year wsu_games
[(2019,6),(2020,1),(2021,7)]
```

5. `compress_str` - 16%

Write a function `compress_str` that takes a string and compresses the input string according to the following rules:

- If a character (for example 'c') repeats two or more times consecutively (say n times), then the consecutive appearances of the character is replaced by 'cn'. For example, if the substring is a sequence of 'a' ("aaaa"), it will be represented as "a4".
- If a character appears once and doesn't repeat consecutively, then it will remain unchanged. For example, if the substring is "a", then it will be represented as "a".

You can assume that the string input consists of lowercase English characters only.

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

```
compress_str :: [Char] -> [Char]
```

Examples:

```
> compress_str "abcaaabbb"
"abca3b3"
> compress_str "abcd"
"abcd"
> compress_str "aaabaaaaccaaaaba"
" a3ba4c2a4ba"
> compress_str ""
""
```

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 '`wsu_games`' list for Problem-4.
- Make sure to define your helper functions inside a let..in or where block.

Testing your functions – 6%

Install `HUnit`

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

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 window (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.`

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 4(b), it is sufficient to provide one additional test case; make sure to use different 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.

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 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