CptS355 - Assignment 1 (Haskell)
Spring 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 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. list_diff – 15%

a) **[10pts]** Define a recursive function `list_diff` which takes two lists as input and returns a list of the non-common elements from the two input lists. The output list should include the elements from the first list that don't appear in the second list and the elements from the second list that don't appear in the first list. You should not eliminate the duplicates in the output. The order of the elements in the output can be arbitrary.

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

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

Examples:

```
> list_diff [1,6,1,3,4,3,5]  [3,8,5,4,7,4,8]
[1,6,1,8,7,8]

> list_diff "We care about our world."
            "We care most about the humans in our world!"
".mshhmnsin!"

> list_diff  ["alpha","gamma","beta","pi","theta"]
            ["pi","gamma","alpha","pi","beta","theta"]
[]
```

2. replace – 15%

The function `replace` takes a list, a value v1, a replacement value v2, and a number n as input, and it replaces the first n occurrences of v1 in the list with v2. Note that, if n is greater than or equal to the number of occurrences of v1 in the list, then it will replace all occurrences of v1 with v2.

The type of the `replaces` function should be compatible with one of the following (depending on the comparison you apply on the n value, the type of your function will be different):

```
replace :: (Num t, Eq t, Eq a) => [a] -> a -> a -> t -> [a]
replace :: (Ord t, Num t, Eq a) => [a] -> a -> a -> t -> [a]
```

Examples:

```
> replace "CptS 355 is offered at 5:55pm" '5' '2' 3
"CptS 322 is offered at 2:55pm"

> replace [(1,10),(2,20),(3,30),(1,10),(4,40)] (1,10) (0,0) 2
[(0,0),(2,20),(3,30),(0,0),(4,40)]

> replace [1,1,2,1,2,3,1,2,3,4,1,2,3,4,5] 4 40 3
[1,1,2,1,2,3,1,2,3,40,1,2,3,40,5]

> replace [1,1,2,1,2,3,1,2,3,4,1,2,3,4,5] 6 60 1
[1,1,2,1,2,3,1,2,3,4,1,2,3,4,5]
```

3. `max_date` – 10%

Assume we represent a date value as a 3-tuple which includes the month, day, and year. For example, (1, 21, 2022) is the tuple representing the date Jan 21, 2022.

Define a recursive function `max_date` which takes a list of date tuples as input and returns the tuple representing the most recent date (i.e., max date). If there are more than one tuple having the max date, it should return one of them.

Hint: Write a helper function that returns the max of the two given date tuples and use It in your solution.

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

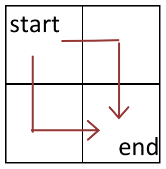
```
max_date :: (Ord a1, Ord a2, Ord a3) => [(a2, a3, a1)] -> (a2, a3, a1)
```

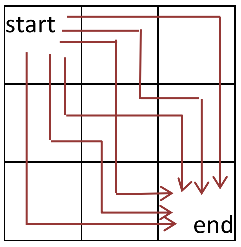
Examples:

```
>max_date [(12,1,2021), (11,30, 2021), (2,1,2022), (1,5,2021), (12,15,2021),
(2,1,2022), (12,1,2021), (1,5,2022)]
(2,1,2022)
>max_date [(11,26,2021), (1,26,2022), (1,27,2021), (1,26,2022), (1,27,2021),
(11, 26, 2021)]
(1,26,2022)
>max_date [(11, 30, 2021)]
(11,30,2021)
```

4. `num_paths` – 10%

Consider a robot in a (m X n) grid who is only capable of moving right or down in the grid (can't move left, up, or diagonal). The robot starts at the top left corner, (1,1), and is supposed to reach to the bottom right corner: `(m,n)`. Write a function `numPaths` that takes the grid length and width (i.e.`,m,n`) as argument and returns the number of different paths the robot can take from the start to the end. Give an answer using recursion.





For example, the 2x2 , there are 2 paths the robot can take from (1,1) to (2,2). For the 3x3 grid, the robot has 6 different paths.

Examples:

```
>num_paths 4 3
10
>num_paths 5 8
330
```

```
> num_paths 9 10
24310
> num_paths 1 1000
1
```

5. `find_courses` and `max_count` – 25%

Assume that we store the list of CptS courses and the programming languages that are used in those classes as a list of tuples. The first element of each tuple is the course major + number and the second element is the list of the programming languages that are used in that course. See below for an example.

```
progLanguages =
     [ ("CptS121" , ["C"]),
     ("CptS122" , ["C++"]),
     ("CptS223" , ["C++"]),
     ("CptS233" , ["Java"]),
     ("CptS321" , ["C#"]),
     ("CptS322" , ["Python", "JavaScript"]),
     ("CptS355" , ["Haskell", "Python", "PostScript", "Java"]),
     ("CptS360" , ["C"]),
     ("CptS370" , ["Java"]),
     ("CptS315" , ["Python"]),
     ("CptS411" , ["C", "C++"]),
     ("CptS451" , ["Python", "C#", "SQL"]),
     ("CptS475" , ["Python", "R"])
     ]
```

(a) Write a Haskell function `find_courses` that takes the list of courses (similar to `progLanguages`) and a programming language name (for example “Java”) as input and returns the list of the courses which use that programming language.

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

```
find_courses :: Eq t1 => [(a, [t1])] -> t1 -> [a]
find_courses :: (Foldable t1, Eq t2) => [(a, t1 t2)] -> t2 -> [a]
```

Note: `Foldable` is the typeclass which admits a folding operation, for example a list. A fold aggregates the elements of a structure using a combining function.

Examples:

```
> find_courses progLanguages "Python"
["CptS322","CptS355","CptS315","CptS451","CptS475"]
> find_courses progLanguages "C++"
["CptS122","CptS223","CptS411"]
> find_courses progLanguages "Go"
[ ]
```

(b) Write a Haskell function `max_count` that takes the list of courses as input (similar to `progLanguages`) and returns the course with max number of programming languages. It returns a two-tuple including the course major + number and the number of programming languages it uses. In the above list, the course associated with the most number of languages is "CptS355".

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

```
max_count :: [(a1, [a2])] -> (a1, Int)
max_count :: Foldable t => [(a1, t a2)] -> (a1, Int)
```

Examples:

```
> max_count progLanguages
("CptS355",4)
```

5. split_at_duplicate – 20%

Write a function `split_at_duplicate` that takes a list as input and splits the input list at the consecutive duplicate elements. The goal is to produce a result in which the elements of the original list have been collected into ordered sub-lists each containing the elements between the repeated duplicate elements in the input list. The function will return a nested list including the sublists obtained by the splits. The duplicate values should be included in the sublists.

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

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

Examples:

```
> split_at_duplicate [1,2,3,1,1,4,5,5,5,6,7,6,6,8,9,9]
[[1,2,3,1],[1,4,5],[5],[5,6,7,6],[6,8,9],[9]]
> split_at_duplicate [10,10,10,10,10,10,10,10]
[[10],[10],[10],[10],[10],[10],[10],[10]]
> split_at_duplicate [1,2,3,4,5,3,4,5,6,7,8]
[[1,2,3,4,5,3,4,5,6,7,8]]
> split_at_duplicate ([]::[Int])
[]
```

(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.

Windows (using cabal installer) :

Run the following commands on the terminal.

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

Mac (using stack installer)

Run the following commands on the terminal.

```
stack install HUnit
```

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

Running Tests

The `HW1SampleTests.zip` file includes 6 `.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 5(b), it is sufficient to provide one additional test case. 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