CptS355 - Assignment 2 (Haskell)
Fall 2023

Assigned: Thursday, September 28, 2023

Weight: Assignment 2 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 Haskell GHC compiler.

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 `HW2.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.

To submit your assignment, please upload `HW2.hs` file to the Assignment2 (Haskell) DROPBOX on Canvas (under Assignments).

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 the basic built-in functions in the Prelude library. (You are not allowed to import an additional library and use functions from there.)
- If a problem asks for a non-recursive solution, then your function should make use of the higher order functions we covered in class (`map, foldr/foldl,` or `filter`.) For those problems, your main functions can't be recursive. If needed, you may define non-recursive helper 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.
- Question 1(b) requires the solution to be tail recursive. Make sure that your function is tail recursive otherwise you won't earn points for this problem.
- You will call `foldr/foldl, map,` or `filter` in several problems. You can use the built-in definitions of these functions.
- When auxiliary/helper functions are needed, make them local functions (inside a `let..in` or `where` blocks). You will be deducted points if you don't define the helper functions inside a `let..in` or `where` block. If you are calling a helper function in more than one function, you can define it in the main scope of your program, rather than redefining it in the let blocks of each calling function.

- 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
- Haskell comments : `-- line comment`

```
{- multi line
   comment-}.
```

Problems

1. group_items tail-recursive – 10%

Examples:

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

> group_items "abcdefghijklmnopqrstuwxyz012"
["a","bc","def","ghij","klmno","pqrstu","wxyz012"]

> group_items []
[]
```

2. get_latest_nested higher-order solution – 10%

Your function shouldn't need a recursion but should use higher order functions (`map`, `foldr/foldl`, or `filter`). Your helper functions should not be recursive as well, but they can use higher order functions.

Examples:

```
> get_latest_nested [[(31,8,2022),(20,7,2023),(25,9,2023)], [(1,10,2023)],
[(30,12,2022),(6,7,2021),(6,7,2023)]]
(1,10,2023)

> get_latest_nested [[(31,8,2021),(20,8,2021),(1,8,2021),(30,12,2020)],
[(6,7,2019),(6,8,2021)], [(30,12,2020),(1,9,2021),(6,7,2019),(6,8,2021)]]
(1,9,2021)
```

3. study_list – higher-order solution – 12%

Your function shouldn't need a recursion but should use higher order functions (`map`, `foldr/foldl`, or `filter`). Your helper functions should not be recursive as well, but they can use higher order functions.

```
log_input = [("CptS355", [("Mon",3), ("Wed",2), ("Sat",2)]) ,
             ("CptS360", [("Mon",3), ("Tue",2), ("Wed",2), ("Fri",10)]),
             ("CptS321", [("Tue",2), ("Wed",2), ("Thu",3)]),
             ("CptS322", [("Tue",1), ("Thu",5), ("Sat",2)]) ]
```

Examples:

```
> study_list log_input "Sat"
["CptS355", "CptS322"]

> study_list log_input "Tue"
["CptS360", "CptS321", "CptS322"]
> study_list log_input "Sun"
[]
```

4.

Define the following Haskell datatype:

```
data IEither  = IString String | IInt Int
                deriving (Show, Read, Eq)
```

(a) `either_sum1` – recursive solution 7%

You may use the following function to convert a string value to integer.

```
getInt x = read x::Int
```

Examples:

```
> either_sum1  [[IString "1",IInt 2,IInt 3], [IString "4",IInt 5], [IInt 6,
IString "7"], [],[IString "8"]]
IInt 36

> either_sum1 [[IString "10" , IInt 10],[],[IString "10"],[]]
IInt 30

> either_sum1  [[]]
IInt 0
```

(b) `either_sum2` – higher order solution 7%

Your function shouldn't need a recursion but should use higher order functions (`map`, `foldr/foldl`, or `filter`). Your helper functions should not be recursive as well, but they can use higher order functions.

Examples:

```
> either_sum2  [[IString "1",IInt 2,IInt 3], [IString "4",IInt 5], [IInt 6,
IString "7"], [],[IString "8"]]
IInt 36

> either_sum2 [[IString "10" , IInt 10],[],[IString "10"],[]]
IInt 30

> either_sum2  [[]]
IInt 0
```

(c) `either_ filter1 – recursive solution` 7%

Examples:

```
> either_filter1 (IInt 8) [[IString "15",IInt 8,IInt 7], [IString "7",IInt 9],
[IInt 8, IString "11"], [],[IString "4"]]
[IString "15", IInt 8, IInt 9, IInt 8, IString "11"]

>either_filter1 (IString "10") [[IString "10" , IInt 9],[],[IString "9", IInt
10],[]]
[IString "10",IInt 10]

>either_filter1 (IString "1") []
[ ]
```

(d) `either_ filter2 – higher order solution` 7%

Your function shouldn't need a recursion but should use higher order functions (`map`, `foldr/foldl`, or `filter`). Your helper functions should not be recursive as well, but they can use higher order functions.

```
> either_filter2 (IInt 8) [[IString "15",IInt 8,IInt 7], [IString "7",IInt 9],
[IInt 8, IString "11"], [],[IString "4"]]
[IString "15", IInt 8, IInt 9, IInt 8, IString "11"]

>either_filter2 (IString "10") [[IString "10" , IInt 9],[],[IString "9", IInt
10],[]]
[IString "10",IInt 10]

>either_filter2 (IString "1") []
[ ]
```

5. isBSTtree, createtree, binarysearch

In Haskell, a polymorphic binary tree type with data both at the leaves and interior nodes might be represented as follows:

```
data Btree a = BLeaf a | BNode a (BTree a) (BTree a)
             deriving (Show, Read, Eq)
```

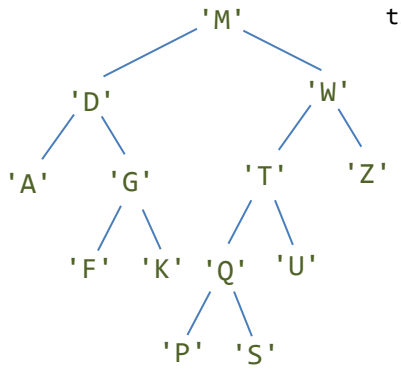
Assume we modify the above datatype and have each interior node (i.e., BNode value) store the level of the sub-tree rooted at that node (with respect to the root). We call this new datatype LTree, defined as follows:

```
data Ltree a = LLeaf (Int,a) | LNode (Int,a) (Ltree a) (Ltree a)
             deriving (Show, Read, Eq)
```

The Int value in the LNode is the level of the subtree.

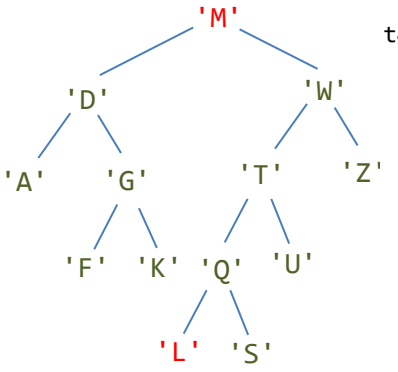
(a) isBSTtree - 12%

Assume there are no duplicate keys in the input tree.




```
t3 =  BNode 'M'
        (BNode 'D' (BLeaf 'A')
                   (BNode 'G' (BLeaf 'F') (BLeaf 'K')))
        (BNode 'W' (BNode 'T' (BNode 'Q' (BLeaf 'P') (BLeaf 'S')) (BLeaf 'U'))
                   (BLeaf 'Z'))
```

isBSTtree t3 returns True.




```
t4 =  BNode 'M'
        (BNode 'D' (BLeaf 'A')
                   (BNode 'G' (BLeaf 'F') (BLeaf 'K')))
        (BNode 'W' (BNode 'T' (BNode 'Q' (BLeaf 'L') (BLeaf 'S')) (BLeaf 'U'))
                   (BLeaf 'Z'))
```

isBSTtree t4 returns False.

Examples:

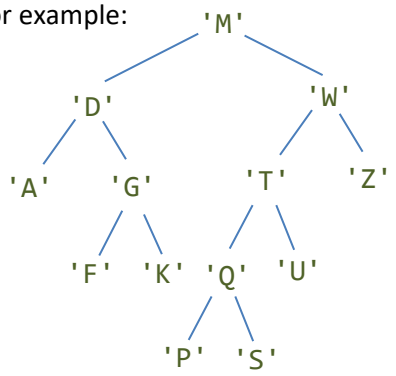
```
t1 = BNode 9 (BNode 6 (BNode 4 (BLeaf 2) (BLeaf 5)) (BLeaf 8)) (BNode 13 (BLeaf 11)
(BLeaf 14))

t2 = BNode 9 (BNode 6 (BNode 4 (BLeaf 2) (BLeaf 5)) (BLeaf 10)) (BNode 13 (BLeaf 11)
(BLeaf 14))
```

```
t3 =  BNode 'M' (BNode 'D' (BLeaf 'A') (BNode 'G' (BLeaf 'F') (BLeaf 'K'))) (BNode
'W' (BNode 'T' (BNode 'Q' (BLeaf 'P') (BLeaf 'S')) (BLeaf 'U')) (BLeaf 'Z'))

t4 =  BNode 'M' (BNode 'D' (BLeaf 'A') (BNode 'G' (BLeaf 'F') (BLeaf 'K'))) (BNode
'W' (BNode 'T' (BNode 'O' (BLeaf 'P') (BLeaf 'S')) (BLeaf 'U')) (BLeaf 'Z'))

t5 = BNode 8 (BLeaf 5) (BNode 20 (BNode 15 (BNode 11 (BLeaf 9) (BLeaf 14)) (BLeaf
17)) (BNode 25 (BLeaf 21) (BLeaf 30)))

t6 = BNode 8 (BLeaf 5) (BNode 20 (BNode 15 (BNode 11 (BLeaf 9) (BLeaf 14)) (BLeaf
17)) (BNode 25 (BLeaf 19) (BLeaf 30)))
```

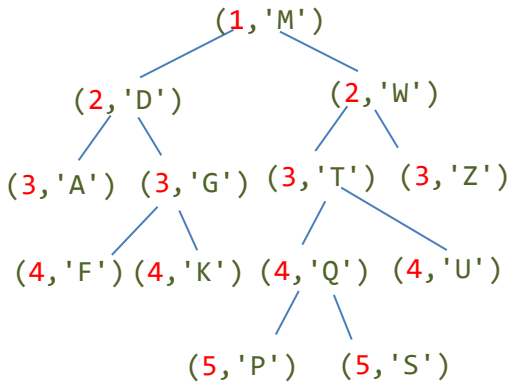
```
> isBSTtree t1
True
> isBSTtree t2
False
> isBSTtree t3
True
> isBSTtree t4
False
> isBSTtree t5
True
> isBSTtree t6
False
```

(b) createtree – 12%

For example:



createtree for the left tree will return the tree of the right. The red values are the levels of the nodes.



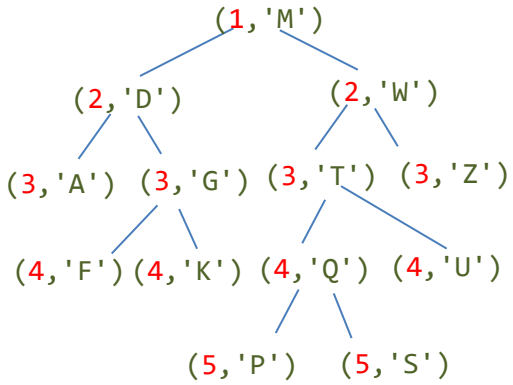
Examples:

```
> createtree t1
LNode (1,9) (LNode (2,6) (LNode (3,4) (LLeaf (4,2)) (LLeaf (4,5))) (LLeaf (3,8))) (LNode
(2,13) (LLeaf (3,11)) (LLeaf (3,14)))

> createtree t3
LNode (1,'M') (LNode (2,'D') (LLeaf (3,'A')) (LNode (3,'G') (LLeaf (4,'F')) (LLeaf (4,'K'))))
(LNode (2,'W') (LNode (3,'T') (LNode (4,'Q') (LLeaf (5,'P')) (LLeaf (5,'S'))) (LLeaf (4,'U')))
(LLeaf (3,'Z')))
```

```
> createtree t5
LNode (1,8) (LLeaf (2,5)) (LNode (2,20) (LNode (3,15) (LNode (4,11) (LLeaf (5,9)) (LLeaf
(5,14))) (LLeaf (4,17))) (LNode (3,25) (LLeaf (4,21)) (LLeaf (4,30))))
```

(c) binarysearch – 12%

Assume there are no duplicate keys in the input tree.
For example:



`binarysearch` for the left tree with target value 'U' will return:
`(Just (4,'U'))`

`binarysearch` for the left tree with target value 'X' will return:
`Nothing`

```
level_t1 = LNode (1,9) (LNode (2,6) (LNode (3,4) (LLeaf (4,2)) (LLeaf (4,5))) (LLeaf
(3,8))) (LNode (2,13) (LLeaf (3,11)) (LLeaf (3,14)))

level_t3 = LNode (1,'M') (LNode (2,'D') (LLeaf (3,'A')) (LNode (3,'G') (LLeaf
(4,'F')) (LLeaf (4,'K')))) (LNode (2,'W') (LNode (3,'T') (LNode (4,'Q') (LLeaf
(5,'P')) (LLeaf (5,'S'))) (LLeaf (4,'U'))) (LLeaf (3,'Z')))

level_t5 = LNode (1,8) (LLeaf (2,5)) (LNode (2,20) (LNode (3,15) (LNode (4,11) (LLeaf
(5,9)) (LLeaf (5,14))) (LLeaf (4,17))) (LNode (3,25) (LLeaf (4,21)) (LLeaf (4,30))))

> binarysearch level_t1 2
Just (4,2)

> binarysearch level_t1 4
Just (3,4)

> binarysearch level_t3 'U'
Just (4,'U')

> binarysearch level_t3 'X'
Nothing

> binarysearch level_t5 14
Just (5,14)

> binarysearch level_t5 99
Nothing
```

Assignment rules – 4%

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

- The module name of your `HW2.hs` files should be `HW2`. 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 `HW2.hs` file, e.g. , tree examples provided in the assignment prompt , the test files and the '`log_input`' list for Problem-3.
- Make sure to define your helper functions inside a let..in or where block.
- Make sure that your solutions meet the specified requirements:
    - Your solution for 1 should be tail-recursive.
    - Your solutions for 2, 3, 4(b), 4(d) shouldn't need a recursion but should use higher order function(s) **`map`, `foldr`/`foldl`, or `filter`.**
    - Any helper function you used for 2, 3, 4(b), 4(d) should not use recursion.

Testing your functions

The `HW2SampleTests.zip` file includes 4 `.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 `HW2` module (`HW2.hs` file) which will include your implementations of the HW problems.

You will write your solutions to `HW2.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_HW2tests.hs
P1_HW2tests> run
```

Repeat the above for other HW problems by changing the test file name, i.e. , `P2_HW2tests.hs`, `P3_HW2tests.hs`, etc.

You don't need to submit any tests for this assignment. However, you should still test your solutions using additional input. Make sure to test your code for boundary cases.