CptS355 - Assignment 2 (Haskell)
Spring 2022

Assigned: Thursday, February 17, 2022

Weight: Assignment 2 will count for 8% 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 `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.

In addition, you will write unit tests using `HUnit` testing package. You will write your tests in the file **`HW2tests.hs`** – the template of this file is available on the HW2 assignment page. You will edit this file and provide additional tests (add at least 2 tests per problem).

To submit your assignment, please upload both files (`HW2.hs` and `HW2Tests.hs`) on 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 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. `commons`, `commons_tail`, and `commons_all`

(a) `commons` – 5%

The function `commons` takes two lists, `l1` and `l2`, and returns a list including the elements that exists in both lists. The resulting list should not include any duplicates. The elements in the output can have arbitrary order.
You may use the built in `elem` function in your solution.

The type of commons should be compatible with the following :

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

Examples:

```
> commons [2,2,5,6,6,8,9] [1,3,2,2,4,4,5,7,8,10]
[2,5,8]
> commons [5,6,7,8,9] [8,8,10,10,11,12,5]
[5,8]
> commons ["a","b","d"] ["c","e","f","g"]
[]
> commons [1,2,3] []
[]
```

(b) `commons_tail` – 9%

Re-write the `commons` function from part (a) as a tail-recursive function. Name your function `commons_tail`.
The type of `commons_tail` should also be `Eq a => [a] -> [a] -> [a]`
You may use the same test cases provided above to test your function.

(c) `commons_all` – 3%

Using `commons` function defined above and the `foldr (or foldl)` function, define `commons_all` which takes a list of lists and returns a list containing the common elements from all the sublists of the input list. **Provide an answer using `foldr` (or `foldl`); without using explicit recursion.**
The type of `commons_all` should be compatible with the following:

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

Examples:

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

2. find_courses and max_count

Assume the "progLanguages" data we used in HW1.

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

(a) find_languages – 10%

Write a Haskell function `find_languages` that takes the list of courses (similar to `progLanguages`) and a list of course names (for example `["CptS451", "CptS321", "CptS355"]`) as input and returns the common languages used by all courses in the course list.

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.

Hint: You can use the `commons_all` function you defined in problem 1.

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

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

```
Examples:
> find_languages progLanguages ["CptS451", "CptS321", "CptS355"]
["Java"]
> find_languages progLanguages ["CptS451", "CptS321", "CptS355","CptS411"]
[ ]
> find_languages progLanguages ["CptS451", "CptS322", "CptS355","CptS475"]
["Python"]
```

(b) `find_courses` – 12%

Write a Haskell function `find_courses` that takes the list of courses (similar to `progLanguages`) and a list of programming language names (for example `["Python","C","C++"]`) as input and returns a list of tuples where each tuple pairs the programming languages with the list of courses that use that programming language.

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.

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

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

Examples:

```
> find_courses progLanguages ["Python","C","C++"]
[("Python",["CptS322","CptS355","CptS315","CptS451","CptS475"]),("C",
["CptS121","CptS360","CptS411"]),("C++",["CptS122","CptS223","CptS411"])]

> find_courses progLanguages ["Java","Go","R"]
[("Java",["CptS233","CptS321","CptS355","CptS370","CptS451"]),("Go",[]),
("R",["CptS475"])]
```

3. `nested_max, max_maybe, and max_numbers`

(a) `nested_max` – 2%

Function `nested_max` is given a `list` of `lists` and it returns the maximum of all numbers in all sublists of the input list. Your function shouldn't need a recursion but should use functions "`map`" and "`foldr`". You may define additional helper functions which are not recursive.

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

```
nested_max :: [[Int]] -> Int
```

If the list is empty, you should return (`minBound::Int`) value

Examples:

```
> nested_max [[1,2,3],[4,5],[6,7,8,9],[]]
9
> nested_max [[10,10],[10,10,10],[10]]
10
> nested_max [[-1,-2,-3],[-4,0,-5],[-6,0,-7,0,-8,-9],[]]
0
```

(b) `max_maybe` – 8%

Function `max_maybe` is given a `list` of `Maybe lists` and it returns the maximum of all `Maybe` values in all sublists of the input list. Your function shouldn't need a recursion but should use functions "`map`" and "`foldr`". You may define additional helper functions which are not recursive. The type of the `max_maybe` function should be compatible with the following:

```
max_maybe :: Ord a => [[Maybe a]] -> Maybe a
```

(Note: To implement `max_maybe`*, change your* `nested_max` *function and your helper function in order to handle* `Maybe` *values instead of Int's. Assume* `Nothing` *is smaller than any Just value.)*

Examples:

```
> max_maybe [[(Just 1),(Just 2),(Just 10),(Just 3)],[(Just 4),(Just 5)],[(Just 6),(Just 10),Nothing ],[],[Nothing ]]
Just 10
> max_maybe [[(Just "A"),Nothing],[(Just "X"), (Just "B"), (Just "Z"),Nothing,Nothing]
Just "Z"
> max_maybe [[Nothing ]]
Nothing
> max_maybe []
Nothing
```

(c) **max_numbers - 8%**

Define the following Haskell datatype:

```
data Numbers  = StrNumber String | IntNumber Int
              deriving (Show, Read, Eq)
```

Define a Haskell function `max_numbers` that takes a nested list of **Numbers** values and it returns the maximum value (as an `Int`) among all **Numbers** in all sublists of the input list. The parameter of the **StrNumber** values should be converted to integer and included in the comparison. You may use the following function to convert a string value to integer.

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

Your `max_numbers` function shouldn't need a recursion but should use functions "`map`" and "`foldr`". You may define additional helper functions <u>which are not recursive.</u> The type of the `max_numbers` function should be compatible with the following:

```
max_numbers :: [[Numbers]] -> Int
```

If the list is empty, you should return (`minBound::Int`) value.

Examples:

```
> max_numbers [[StrNumber "9",IntNumber 2,IntNumber 8],[StrNumber "4",IntNumber 5],[IntNumber 6,StrNumber "7"],[],[StrNumber "8"]]
9
> max_numbers [[StrNumber "10" , IntNumber 12],[],[StrNumber "10"],[]]
12
> max_numbers [[]]
-9223372036854775808     (minBound value)
```

4. tree_scan, tree_search, merge_trees

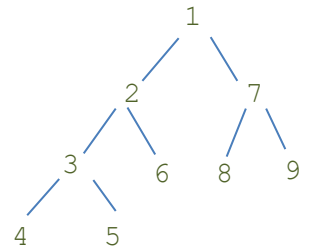
In Haskell, a polymorphic binary tree type with data both at the leaves and interior nodes might be represented as follows:

```
data Tree a = LEAF a | NODE a (Tree a) (Tree a)
              deriving (Show, Read, Eq)
```

(a) tree_scan - 5%

Write a function `tree_scan` that takes a tree of type `(Tree a)` and returns a list of the `a` values stored in the leaves and the nodes. The order of the elements in the output list should be based on the "InOrder" traversal of the tree.

The type of the tree_scan function should be: `tree_scan :: Tree a -> [a]`



InOrder traversal of the tree:
`[4,3,5,2,6,1,8,7,9]`

Examples:

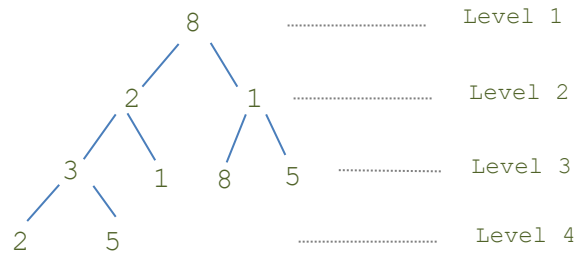
```
t1 =  NODE
         "Computer"
         (NODE "of" (LEAF "School")(NODE
                                    "Engineering"
                                    (LEAF "Electrical")
                                    (LEAF "and")))
          (LEAF "Science")
tree_scan t1
["School","of","Electrical","Engineering","and","Computer","Science"]

t2 = NODE 1 (NODE 2 (NODE 3 (LEAF 4) (LEAF 5)) (LEAF 6)) (NODE 7 (LEAF 8) (LEAF 9))
tree_scan t2
[4,3,5,2,6,1,8,7,9]

tree_scan (LEAF 4)
[4]
```

(b) tree_search – 12%

Write a function `tree_search` that takes a tree of type (`Tree a`) and a value and returns the level of the tree where the value is found. If the value doesn't exist in the tree, it returns -1. The tree nodes should be visited with "InOrder" traversal and the level of the first matching node should be returned.

The type of the tree_search function should be compatible with the following:

```
tree_search :: (Ord p, Num p, Eq a) => Tree a -> a -> p
```



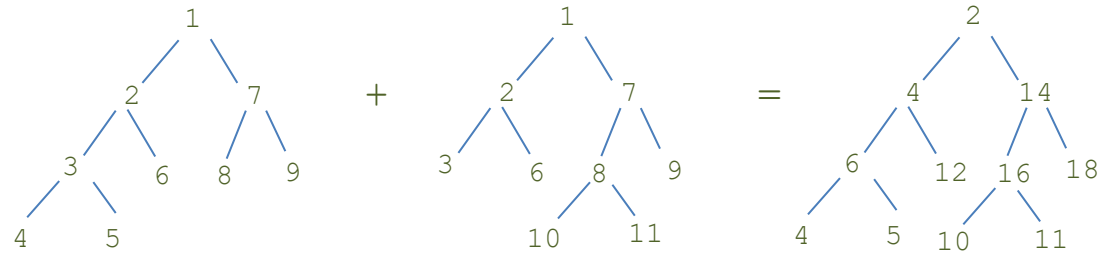

```
t3 = NODE 8 (NODE 2 (NODE 3 (LEAF 2) (LEAF 5)) (LEAF 1)) (NODE 1 (LEAF 8) (LEAF 5))

tree_search t3 1
3
tree_search t3 5
4
tree_search t3 8
1
tree_search t3 4
-1
```

(c) `merge_trees` – 14%

Write a function `merge_trees` that takes two (`Tree int`) values and returns an (`Tree int`) where the corresponding nodes from the two trees are added. The trees might have different depth. You should copy particular branches/nodes of the trees if the other tree doesn't have that branch/node. See the example below.

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

```
merge_trees :: Num a => Tree a -> Tree a -> Tree a
```



We can create the above `Tree(s)` as follows:

left :

```
left = NODE 1 (NODE 2 (NODE 3 (LEAF 4) (LEAF 5)) (LEAF 6)) (NODE 7 (LEAF 8) (LEAF 9))
```

right:

```
right = NODE 1 (NODE 2 (LEAF 3) (LEAF 6)) (NODE 7 (NODE 8 (LEAF 10) (LEAF 11)) (LEAF 9))
```

And `merge_trees left right` will return the rightmost `(Tree Int)` which is equivalent to the following:

```
NODE 2
(NODE 4 (NODE 6 (LEAF 4) (LEAF 5)) (LEAF 12))
(NODE 14 (NODE 16 (LEAF 10) (LEAF 11)) (LEAF 18))
```

5. Tree examples – 3%

Create two trees of type Tree. The height of both trees should be at least 4. Test your functions `tree_scan, tree_search, merge_trees` with those trees. The trees you define should be different than those that are given.

Include your example trees in `HW2Tests.hs` file under the comment "INCLUDE YOUR TREE EXAMPLES HERE". Do not include your sample tests in HW2.hs.

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 'progLanguages' list for Problem-2.
- Make sure to include your own tree values in `HW2Tests.hs` file.
- Make sure to define your helper functions inside a let..in or where block.

Testing your functions – 5%

The `HW2SampleTests.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 `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 are expected to add **at least 2 more test cases** for each problem. **Write your tests for all problems in `HW2_tests.hs` file** – the template of this file is provided to you in the HW2 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(abc), you can use the trees you created in problem-5. See above.

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**.

Important note about negative integer arguments:

In Haskell, the -x, where x is a number, is a special form and it is a prefix (and unary) operator negating an integer value. When you pass a negative number as argument function, you may need to enclose the negative number in parenthesis to make sure that unary (-) is applied to the integer value before it is passed to the function.

For example: `foo -5 5 [-10,-5,0,5,10]` will give a type error, but

`foo (-5) 5 [-10,-5,0,5,10]` will work