universität innsbruck

Functional Programming | SS 2020 | LVA 703025

Exercise Sheet 12, 10 points | Deadline: **Tuesday, June 9, 2020, 10am**

- Please write all the Haskell code into a single .hs-file and upload it in OLAT.
- You can use the template .hs-files that are provided on the proseminar page[1].
- Your .hs-files should be compilable with ghci.
- Don't forget to mark your completed exercises in OLAT.
- Feel free to import functions from the Haskell standard library.

Exercise 12.1 *Binary trees* 4 p.

Consider the following datatype representing a binary tree:

```
data Tree a = Empty | Node (Tree a) a (Tree a)
```

1. Write a function `mapTree` that takes a function `f` and applies it to every element in the tree and also preserves the tree structure. Example:

   ```
   mapTree (+1) (Node (Node Empty 4 Empty) 6 Empty) = (Node (Node Empty 5 Empty) 7 Empty)
   ```

   (1 point)

2. Write a function `heapP` which checks if a binary tree fulfills the heap property[2]. `heapP` takes the order for the heap property as a parameter. Calling `heapP (<=) t` checks if the key stored in each node in `t` is smaller or equal then all keys in the node's children. Examples:

   ```
   heapP (<=) (Node (Node Empty 2 Empty) 1 (Node Empty 2 (Node Empty 3 Empty))) = True
   heapP (<) (Node (Node Empty 3 Empty) 1 (Node Empty 2 (Node Empty 2 Empty))) = False
   heapP (>=) (Node (Node Empty 4 Empty) 6 (Node Empty 3 Empty)) = True
   ```

   (1 point)

3. Treaps[3] are binary trees with a tuple as key at each node. Looking only at the first values of the tuples, a treap is a binary search tree. Looking only at the second values of the tuples, a treap is a min heap (i.e. fulfills `heapP (<=)`). Write a function `treapP` that checks if a binary tree is a treap. Examples:

[1] http://cl-informatik.uibk.ac.at/teaching/ss20/fp/index.php#exercises

[2] https://en.wikipedia.org/wiki/Binary_heap

[3] https://en.wikipedia.org/wiki/Treap

```
treapP (Node (Node Empty (3,2) Empty)
             (5,1)
             (Node Empty (8,6) (Node Empty (10,10) Empty)))
       = True

-- Visual representation:
--      Treap:                BST if first values:         min heap if second values:
--      (5,1)                         5                              1
--       / \                         / \                            / \
--      /   \                       /   \                          /   \
--     /     \                     /     \                        /     \
--   (3,2)  (8,6)                 3       8                      2       6
--             \                           \                              \
--           (10,10)                        10                             10
```

(2 points)

Exercise 12.2 *Longest valid parentheses* 2 p.

In a string containing only the following brackets `()[]` find the length of the longest valid (well-formed) parentheses substring.

```
lenWfBrackets ")()" = 2
lenWfBrackets ")[()])" = 4
lenWfBrackets "(]" = 0
```

Exercise 12.3 *Summing integers* 2 p.

Write a function `summ :: IO ()` which repeatedly asks for a number. If the user enters `s`, the function should sum up all previously entered numbers. On inputs that are not numbers or `s`, the function should just exit (that means not fail with an exception).

```
Number or sum up (s):
43
Number or sum up (s):
5
Number or sum up (s):
-8
Number or sum up (s):
s
Sum: 40
```

Exercise 12.4 *Summing integers* 2 p.

Given a list of integers and a target integer, the function `combinationSum` should return all combinations from the input list that sum up to the target integer. The resulting list should not contain duplicate combinations.

```
combinationSum [10,1,2,7,6,1,5] 8 = [[1,7], [1,2,5], [2,6], [1,1,6]] -- order of output doesn't matter
combinationSum [2,5,2,1,2] 5 = [[1,2,2], [5]] -- [2,2,1] and [1,2,2] are the same combination
```