MIDDLE EAST TECHNICAL UNIVERSITY, NORTHERN CYPRUS CAMPUS
CNG242 Programming Language Concepts – Spring 2021 – Assignment 1

Date handed out: 12 April 2021, Monday 18:00
Submission due: 25 April 2021, Sunday 23:55

Please Read This Page Carefully

Submission Rules

1. **You need to write a comment on the first line of your file**, stating that you read the rules specified here and the submission is your own work. **Submissions without this statement will not be graded.**
   Example, ""-- I read and accept the submission rules and **the extra rules.** This is my own work that is done by myself only""
2. As explained in syllabus[1] provided for CNG 242, attempting any academic dishonesty[2], breaking professionalism and ethics[3] rules may result in the following:
   - You might be asked to perform an oral test to confirm your submission.
   - You may receive a "zero" grade for this submission
   - **You may receive "zero" from all future submissions.**
   - You may receive a failing letter grade for the course and this case might be forwarded to the discipline committee.
3. You cannot use someone' else's code.
4. You **cannot hire** someone to write the code for you.
5. **You cannot share your code with someone else.**
6. You need to be ready to demonstrate your own work, answer related questions, and have short coding sessions if necessary.
7. You need to submit a **single Haskell file named with your student id** only. For example, **1234567.hs**
8. Function names must be the same as the provided questions.
9. You cannot share this worksheet with any third parties. Upon doing so, any detected action will directly be sent to the disciplinary committee.
10. **You cannot import libraries; you cannot use anything that we did not cover in the first five weeks. Everything you need is in lab worksheet 1,2,3,4,5.**
11. You should only submit one solution per question. Your solution might have multiple lines. **Only the functions with the same name will be graded.**
12. You can only get full marks, if your file fully compiles, runs, and generates correct output for all possible cases. **Non-general static solutions will not be graded!**
13. **If you are added to another section as a guest student**, you will still have to submit your submission under the main section you are registered. You can check your registered section by trying to see your grades in ODTUClass. Only submit your file to the section that you can see your grades.

[1] Page 3&4 (Course rules, #1,2,3)
[2] Taking unfair advantage in assessment is considered a serious offence by the university, which will take action against any student who contravenes the regulation through negligence or deliberate intent.
[3] For a comprehensive cheating definition, please refer to: https://ncc.metu.edu.tr/res/academic-code-of-ethics . When a breach of the code of ethics occurs (cheating, plagiarism, deception, etc.), the student will be added to the BLACKLIST.

Assignment I: Positive Tree

Learning Outcomes: On successful completion of this assignment, a student will:

- practice how to program using functional programming languages.
- use different approaches to define data types in Haskell.
- practice how to use recursive definitions in the functional programming paradigm.

This assignment uses the rooted tree terminology from the textbook popularly used for teaching CNG 223 - Discrete Computational Structures [1]. The rooted tree to be considered is a full binary tree. This means every node has either zero or two children. The tree is also a polymorphic one allowing us to store integer or float values.

A very specific form of input is provided for the construction of the tree. The input is provided in the form of values for the leaves of the tree (vertices with no children) and the type of operation to be performed (either multiplication or addition). The operation provided is, in turn, used for the computation of values for internal vertices (vertices that have children). The input to specify the values of leaves and their position, as well as the positions of internal vertices, are provided as a list.

Example

Figure 1a below shows a full binary *tree with some integer values assigned to its leaves*. The internal vertices are not assigned with any value at the beginning. In case addition is the operator chosen to apply on the tree, the internal vertices are calculated by adding the values of children. The resulting tree is illustrated in Fig 1b.

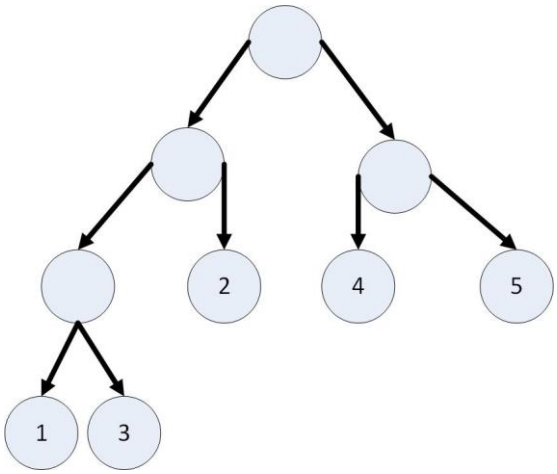

Figure1a: The full binary tree example before calculating the internal vertice

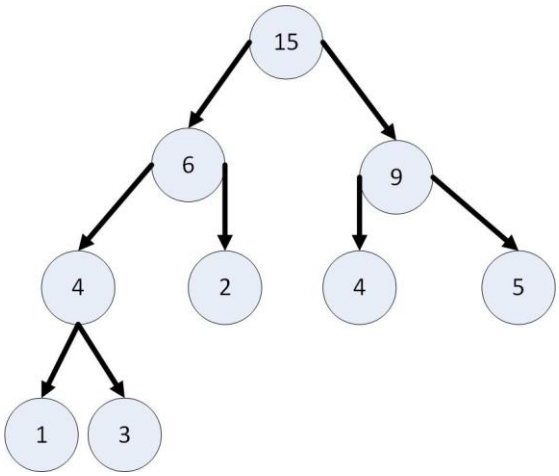

Figure1b: The full binary tree example after calculation of the internal vertice

Part One: Construct the tree (10 pts)

In the first part of this assignment, your program will use a list and a character to construct the full binary tree. The tree considered can hold only positive values; therefore, we will use zero as a sentinel value to specify the internal vertices or the missing leaf nodes. For the leaves, we will have positive values.

Input: Lists of values and a character to specify the operator.
Output: The full binary positive tree
Sample run (given for figure 1a):

```
*Main> positivetree [[0],[0,0],[0,2,4,5],[1,3,0,0,0,0,0,0]]  '+'
```

Each member of the list given (first parameter of the function) is a list showing vertices' values in that particular level. Therefore the order used to specify the values of vertices can be further explained as follows.

Node 15 (Node 6 (Node 4 (Leaf 1) (Leaf 3)) (Leaf 2)) (Node 9 (Leaf 4) (Leaf 5))

Part Two: Height of the tree (2 pts)

In the second part, the height of the tree will be calculated.
Input: The positive tree
Output: The height
Sample run:

```
*Main> postree = positivetree [[0],[0,0],[0,2,4,5],[1,3,0,0,0,0,0,0]]  '+'
generateheight postree
4
```

Part Three: the weight of each level (3 pts)

The level of a vertex v in a rooted tree is the length of the unique path from the root to this vertex. In the third part, the addition of all vertices in a specific level will be computed.

Input: Any tree
Output: The total value of all vertices in this level
Sample run:

```
*Main> postree = positivetree [[0],[0,0],[0,2,4,5],[1,3,0,0,0,0,0,0]]  '+'
*Main> levelweight postree 3
15
*Main> levelweight postree 4
4
*Main> levelweight postree 5
0
*Main> testtree = Node 5 (Node 4 (Leaf 12) (Leaf 7)) (Leaf 5)
*Main> levelweight testtree 1
5
*Main> levelweight testtree 2
9
*Main> levelweight testtree 3
19
*Main> levelweight testtree 4
0
```

Part Four: manual traversal (5 pts)

In part four, a value will be provided as the starting node, and a list of directions will be provided to traverse starting from this node (L for left child and R for right child). The value at the vertex reached will be the output of the function. In case the list of directions is not empty, but we reach a leaf, a message "StoppedEarly" will be printed to show the situation. If a given node is not found, the function should print "NodeNotFound".

Input: The positive tree the value to search for and a list of directions
Output: The value at the final destination
Sample run:

```
*Main> postree = positivetree [[0],[0,0],[0,2,4,5],[1,3,0,0,0,0,0,0]]  '+'
*Main> manualtraversal postree 6 "LR"
Found 3
*Main> manualtraversal postree 6 "LRR"
StoppedEarly
*Main> manualtraversal postree 7 "LRR"
NodeNotFound
```

Extra Rules:

- Strictly obey the specifications, input output formats. Do not print extra things.
- Solutions without a Tree data type will not be accepted.
- All functions, except the construct tree function should work with any given tree.
- In your file, separate your answers with comments. Everything related to each question should be located in their own commented section.
- Do not give extra errors.

Tips:

- Even if you can't construct the tree, you can still build other functions.
- Check appendix for extra input/outputs.

Assessment Criteria

The assignment will be marked as follows:

Item	Marks (Total 20)
Construction of full binary positive tree (Q1)	6
Tree data constructor (Q1)	2
Polymorphic implementation of positive tree (Q1)	2
Height of the tree (Q2)	2
Weight of each level (Q3)	3
Finding nodes (Q4)	2
Traversing (Q4)	1
StoppedEarly implementation (Q4)	1
NodeNotFound implementation (Q4)	1

* Code clarity, repetitiveness, following the rules will be graded too.
** Function formats has to be same in order to receive full points.

Appendix

```
*Main> ----------Q1 Sample------------
*Main> positivetree [[1]] '+'
Leaf 1
*Main> positivetree [[1],[1,2]] '+'
Leaf 1
*Main> positivetree [[0],[1,2]] '+'
Node 3 (Leaf 1) (Leaf 2)
*Main> positivetree [[0],[0,2],[1,3,0,0]] '+'
Node 6 (Node 4 (Leaf 1) (Leaf 3)) (Leaf 2)
*Main> positivetree [[0],[0,2],[1,3,4,5]] '+'
Node 6 (Node 4 (Leaf 1) (Leaf 3)) (Leaf 2)
*Main> positivetree [[0],[0,2],[1,3,0,0]] '*'
Node 6 (Node 3 (Leaf 1) (Leaf 3)) (Leaf 2)
*Main> positivetree [[0],[0,4],[2,3,0,0]] '*'
Node 24 (Node 6 (Leaf 2) (Leaf 3)) (Leaf 4)
*Main> positivetree [[0],[0,4],[2,0,0,0],[0,0,3,1,0,0,0,0]] '*'
Node 24 (Node 6 (Leaf 2) (Node 3 (Leaf 3) (Leaf 1))) (Leaf 4)
*Main> positivetree [[0],[4,0],[0,0,2,0],[0,0,0,0,0,0,3,5]] '*'
Node 120 (Leaf 4) (Node 30 (Leaf 2) (Node 15 (Leaf 3) (Leaf 5)))
*Main> positivetree [[0],[4,0],[0,0,2,0],[0,0,0,0,0,0,5]] '+'
NotFullBinary
*Main> positivetree [[0],[4,0],[0,0,2,0],[0,0,0,0,0,5,0]] '+'
NotFullBinary
*Main> positivetree [[0],[4,0],[0,0,0,2],[0,0,0,0,3,1,0,0]] '+'
Node 10 (Leaf 4) (Node 6 (Node 4 (Leaf 3) (Leaf 1)) (Leaf 2))
*Main> positivetree [[0],[4,0],[0,0,2,0],[0,0,0,0,0,0,0,0]] '*'
Node 0 (Leaf 4) (Node 0 (Leaf 2) (Node 0 (Leaf 0) (Leaf 0)))
*Main> positivetree [[0],[4,0],[0,0,2,0],[0,0,0,0,0,0,1,3]] '*'
Node 24 (Leaf 4) (Node 6 (Leaf 2) (Node 3 (Leaf 1) (Leaf 3)))
*Main> --------------------------------
*Main> ----------Q2 Sample------------
*Main> generateheight (Leaf 1)
1
*Main> generateheight (Node 3 (Leaf 1) (Leaf 2))
2
*Main> generateheight (Node 1 (Node 2 (Leaf 3) (Leaf 4)) (Leaf 5))
3
*Main> generateheight (Node 24 (Leaf 4) (Node 6 (Leaf 2) (Node 3 (Leaf 1) (Leaf 3))))
4
*Main> --------------------------------
*Main> ----------Q3 Sample------------
*Main> levelweight (Node 15 (Leaf 5) (Node 8 (Leaf 1) (Node 4 (Leaf 1) (Leaf 3)))) 1
15
*Main> levelweight (Node 15 (Leaf 5) (Node 8 (Leaf 1) (Node 4 (Leaf 1) (Leaf 3)))) 2
13
*Main> levelweight (Node 15 (Leaf 5) (Node 8 (Leaf 1) (Node 4 (Leaf 1) (Leaf 3)))) 3
5
*Main> levelweight (Node 15 (Leaf 5) (Node 8 (Leaf 1) (Node 4 (Leaf 1) (Leaf 3)))) 4
4
*Main> levelweight (Node 15 (Leaf 5) (Node 8 (Leaf 1) (Node 4 (Leaf 1) (Leaf 3)))) 5
0
```

```
*Main> --------------------------------
*Main> -----------Q4 Sample-----------
*Main> manualtraversal (Node 15 (Leaf 5) (Node 8 (Leaf 1) (Node 4 (Leaf 1) (Leaf 3)))) 2 "LR"
NodeNotFound
*Main> manualtraversal (Node 15 (Leaf 5) (Node 8 (Leaf 1) (Node 4 (Leaf 1) (Leaf 3)))) 8 "LR"
StoppedEarly
*Main> manualtraversal (Node 15 (Leaf 5) (Node 8 (Leaf 1) (Node 4 (Leaf 1) (Leaf 3)))) 8 "R"
Found 4
*Main> manualtraversal (Node 15 (Leaf 5) (Node 8 (Leaf 1) (Node 4 (Leaf 1) (Leaf 3)))) 8 "RL"
Found 1
*Main> manualtraversal (Node 15 (Leaf 5) (Node 8 (Leaf 1) (Node 4 (Leaf 1) (Leaf 3)))) 15 "LL"
StoppedEarly
*Main> manualtraversal (Node 15 (Leaf 5) (Node 8 (Leaf 1) (Node 4 (Leaf 1) (Leaf 3)))) 15 "LR"
StoppedEarly
*Main> manualtraversal (Node 15 (Leaf 5) (Node 8 (Leaf 1) (Node 4 (Leaf 1) (Leaf 3)))) 15 "L"
Found 5
*Main> manualtraversal (Node 15 (Leaf 5) (Node 8 (Leaf 1) (Node 4 (Leaf 1) (Leaf 3)))) 3 "L"
NodeNotFound
*Main> manualtraversal (Node 15 (Leaf 5) (Node 8 (Leaf 1) (Node 4 (Leaf 1) (Leaf 3)))) 1 "L"
NodeNotFound
*Main> manualtraversal (Node 15 (Leaf 5) (Node 8 (Leaf 1) (Node 4 (Leaf 1) (Leaf 3)))) 4 ""
Found 4
*Main> --------------------------------
*Main> ------Polymorphic Examples------
*Main> ---Notice the different types---
*Main> positivetree [[1]] '+'
Leaf 1
*Main> positivetree [[1.5]] '+'
Leaf 1.5
*Main> positivetree [[1],[1.2,2]] '+'
Leaf 1.0
*Main> positivetree [[0],[1,2.5]] '+'
Node 3.5 (Leaf 1.0) (Leaf 2.5)
*Main> positivetree [[0],[0,2.5],[1,3,0,0]] '+'
Node 6.5 (Node 4.0 (Leaf 1.0) (Leaf 3.0)) (Leaf 2.5)
*Main> positivetree [[0],[0,2],[1,3,4,5]] '+'
Node 6 (Node 4 (Leaf 1) (Leaf 3)) (Leaf 2)
*Main> positivetree [[0],[0,2],[1.2,3.5,0,0]] '*'
Node 8.4 (Node 4.2 (Leaf 1.2) (Leaf 3.5)) (Leaf 2.0)
*Main> positivetree [[0],[0,4],[2,3,0,0]] '*'
Node 24 (Node 6 (Leaf 2) (Leaf 3)) (Leaf 4)
*Main> positivetree [[0],[0,4],[2.3,0,0,0],[0,0,3.4,1.3,0,0,0,0]] '*'
Node 40.663999999999994 (Node 10.165999999999999 (Leaf 2.3) (Node 4.42 (Leaf 3.4) (Leaf 1.3))) (Leaf 4.0)
*Main> positivetree [[0],[4,0],[0,0,2,0],[0,0,0,0,0,0,3,5]] '*'
Node 120 (Leaf 4) (Node 30 (Leaf 2) (Node 15 (Leaf 3) (Leaf 5)))
*Main> positivetree [[0],[4,0],[0,0,2,0],[0,0,0,0,0,0,9]] '+'
NotFullBinary
*Main> positivetree [[0],[4,0],[0,0,7,0],[0,0,0,0,0,9.5,0]] '+'
NotFullBinary
```

References

1. Rosen, Kenneth H. Discrete Mathematics and Its Applications. 7th ed, McGraw-Hill, 2012.