Middle East Technical University Northern Cyprus Campus

CNG242 Programming Language Concepts – Homework 1 - Haskell
Assignment 1: THE VERSION TREE

Date handed out: 7 April 2022, Thursday 14:00
Submission due: 22 April 2022, Friday 23:59

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. Please refer to the syllabus[1] provided for CNG 242 for the measures in place in case of any academic dishonesty[2,3].

3. The instructors or TAs may ask for demo sessions for any of the submissions.

4. You need to submit a **single Haskell file named with your 7-digit student id** only. For example, **1234567.hs**

5. Function names must be the same as the provided questions.

6. You cannot share this worksheet with any third parties. Upon doing so, any detected action will directly be sent to the disciplinary committee.

7. **You cannot import libraries; you cannot use anything that we did not cover in the first five weeks. Everything you need is in lab 2,3,4, and 5. You can also use guards.**

8. You should only submit one solution per question. Your solution might have multiple lines. **Only the functions with the same name will be graded.**

9. 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!**

10. **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 where 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.

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

• Have used different approaches to define data types in Haskell
• Have practised how to use recursive definitions in the functional programming paradigm.
• Have practised how to program using functional programming languages.

In this assignment, we are going to work with tree structures. The case study we will consider is a generic version tree that can be used to record different versions of software. Software configuration management (SCM) is the process of tracking and controlling changes in the software[4]. The version control system can be defined as software that allows you to create versions of elements and be involved in different versions independently from each other. The overall process of introducing versions involves both their creation and the implementation of structures for their storage. Typically, the structures used are either chains or trees.

In this assignment, you will work with a custom TernaryTree[5] data type for version control. In each node, this tree will hold a string and a level value representing the corresponding node. An example tree is shown in figure 1.

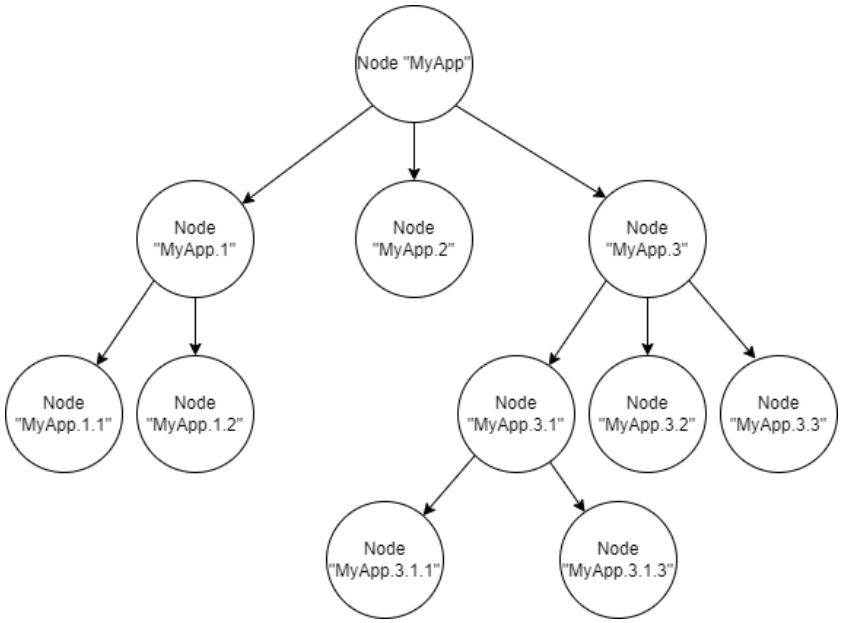

Figure 1: Leveled Ternary Tree

<u>Part One: charToInt</u> (5 pts)

In the first part of this assignment, you will create a function named charToInt, which takes a numerical character from the user and returns the integer version. **Hint**: you can use this function in other parts if necessary.

Input: a character between '0' and '9'.

Output: an Integer converted from the given

Sample run:

```
*Main>  charToInt '1'
1
*Main>  charToInt '2'
2
*Main>  charToInt '3'
3
```

[4] Sommerville, F. Claire A., Reidar Conradi, Jacky Estublier, and Bernhard Westfechtel. Software Configuration Management: ICSE'96 SCM-6 Workshop, Berlin, Germany, March 25-26, 1996, Selected Papers. Vol. 6. Springer Science & Business Media, 1996.

[5] Rosen, Kenneth H., and Kamala Krithivasan. Discrete mathematics and its applications: with combinatorics and graph theory. Tata McGraw-Hill Education, 2012.

Part Two: insertNode (40 pts)

In this part, you will be adding new versions to your version tree. For this purpose, a node will be created from a given string and will be inserted to the existing Tree. After the insertion, the function should return the new tree. If the given position is already occupied, the function should show NodeExists in the terminal. If the given position is not reachable at all, the function should show NotReachable in the terminal. **You cannot use the following functions or any helpers containing the following functions: show, putStrLn**

Input: any ternary tree, string to generate a node

Output: TernaryTree

Sample run (Using the same tree used in Figure 1):

*Main> **leftSide** = Node "MyApp.1" (Node "MyApp.1.1" Empty Empty Empty) (Node "MyApp.1.2" Empty Empty Empty) Empty

*Main> **midSide** = Node "MyApp.2" Empty Empty Empty

*Main> **rightSide =** Node "MyApp.3" (Node "MyApp.3.1" (Node "MyApp.3.1.1" Empty Empty Empty) Empty (Node "MyApp.3.1.3" Empty Empty Empty)) (Node "MyApp.3.2" Empty Empty Empty) (Node "MyApp.3.3" Empty Empty Empty)

*Main> **figureOne** = Node "MyApp" leftSide midSide rightSide

*Main> **figureOne**

Node "MyApp" (Node "MyApp.1" (Node "MyApp.1.1" Empty Empty Empty) (Node "MyApp.1.2" Empty Empty Empty) Empty) (Node "MyApp.2" Empty Empty Empty) (Node "MyApp.3" (Node "MyApp.3.1" (Node "MyApp.3.1.1" Empty Empty Empty) Empty (Node "MyApp.3.1.3" Empty Empty Empty)) (Node "MyApp.3.2" Empty Empty Empty) (Node "MyApp.3.3" Empty Empty Empty))

*Main> insertNode figureOne "MyApp.1"

NodeExists

*Main> insertNode Empty "GoodLuck"

Node " GoodLuck " Empty Empty Empty

*Main> insertNode figureOne "MyApp.1.1"

NodeExists

*Main> insertNode figureOne "MyApp.1.3"

Node "MyApp" (Node "MyApp.1" (Node "MyApp.1.1" Empty Empty Empty) (Node "MyApp.1.2" Empty Empty Empty) (Node "MyApp.1.3" Empty Empty Empty)) (Node "MyApp.2" Empty Empty Empty) (Node "MyApp.3" (Node "MyApp.3.1" (Node "MyApp.3.1.1" Empty Empty Empty) Empty (Node "MyApp.3.1.3" Empty Empty Empty)) (Node "MyApp.3.2" Empty Empty Empty) (Node "MyApp.3.3" Empty Empty Empty))

*Main> insertNode figureOne "MyApp.2.2.2"

NotReachable

*Main> insertNode figureOne "MyApp.3.2.2"

Node "MyApp" (Node "MyApp.1" (Node "MyApp.1.1" Empty Empty Empty) (Node "MyApp.1.2" Empty Empty Empty) Empty) (Node "MyApp.2" Empty Empty Empty) (Node "MyApp.3" (Node "MyApp.3.1" (Node "MyApp.3.1.1" Empty Empty Empty) Empty (Node "MyApp.3.1.3" Empty Empty Empty)) (Node "MyApp.3.2" Empty (Node "MyApp.3.2.2" Empty Empty Empty) Empty) (Node "MyApp.3.3" Empty Empty Empty))

Part Three: totalNodes (8 pts)

In this part, the total number of nodes of a given tree will be calculated.

Input: Leveled ternary tree

Output: The height

Sample run (Assume figureOne from PartTwo):

*Main> totalNodes leftSide

3

*Main> totalNodes midSide

1

*Main> totalNodes rightSide

6

*Main> totalNodes figureOne

11

Part Four: Height of the tree (8 pts)

In the third part, the height of a given tree will be calculated. Assume that we have leftSide, midSide, rightSide, figureOne in Part Two

Input: Leveled ternary tree

Output: The height

Sample run (Assume leftSide, midSide, rightSide, figureOne from PartTwo):

*Main> height figureOne
4
*Main> height leftSide
2
*Main> height midSide
1
*Main> height rightSide
3

Part Five: node counts of each level (8 pts)

In the fourth part, the number of nodes in a specific level will be computed.

Input: Any ternary tree

Output: The number of nodes in this level

Sample run (Assume figureOne from PartTwo):

*Main> levelcount figureOne 0
1
*Main> levelcount figureOne 1
3
*Main> levelcount figureOne 2
5
*Main> levelcount figureOne 3
2
*Main> levelcount figureOne 4
0
*Main> levelcount Empty 0
0
*Main> levelcount Empty 1
0

Part Six: findNode (25 pts)

In part five, you will be searching for the given String inside the given Tree. If the value is found, the function should return the whole Node with its children, if not it should show NodeNotFound in the terminal. **You cannot use following functions or any helpers containing following functions: show, putStrLn**

Input: any ternary tree, the string to search for

Output: TernaryTree

Sample run (Assume figureOne from PartTwo):

*Main> findNode Empty "MyApp.2"
NodeNotFound
*Main> findNode figureOne "MyApp.3.1.1"
Node "MyApp.3.1.1" Empty Empty Empty
*Main> findNode figureOne "MyApp.3.1.2"
NodeNotFound
*Main> findNode figureOne "MyApp.1"
Node "MyApp.1" (Node "MyApp.1.1" Empty Empty Empty) (Node "MyApp.1.2" Empty Empty Empty) Empty

*Main> findNode figureOne "MyApp"
Node "MyApp" (Node "MyApp.1" (Node "MyApp.1.1" Empty Empty Empty) (Node "MyApp.1.2" Empty Empty Empty) Empty) (Node "MyApp.2" Empty Empty Empty) (Node "MyApp.3" (Node "MyApp.3.1" (Node "MyApp.3.1.1" Empty Empty Empty) Empty (Node "MyApp.3.1.3" Empty Empty Empty)) (Node "MyApp.3.2" Empty Empty Empty) (Node "MyApp.3.3" Empty Empty Empty))
*Main> findNode figureOne "MyApp.2.2"
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 should work with any given ternary tree.
- Do not give extra errors.

Hints:

- Even if you can't do certain parts, you can still do other ones.
- If you separate each question with comments, including helper functions for an uncompleted question may earn you partial points. In that case, please comment and explain your work and idea creating that helper.
- Check appendix for extra input/outputs.
- Try to create and use helper functions for hard questions.
- You may assume all the test cases will be given correctly.
- You may use included parts as your helpers for other parts if necessary.

Assessment Criteria

The assignment will be marked as follows:

Item	Marks (Total 100)
charToInt	5
totalNodes	8
height	8
levelcount	8
findNode	25
insertNode	40
NodeExists, NotReachable, NodeNotFound implementation	6

* Code clarity, repetitiveness, following the rules will be graded too (%40 each grading item).
** Function formats has to be same to receive full points (%20 each grading item).
*** Partially working solutions may receive up to %80 per grading item.
**** Please check the appendix below for further sample runs.

APPENDIX

Part One

Sample Run

```
*Main> charToInt '1'
1
*Main> charToInt '2'
2
*Main> charToInt '3'
3
*Main> charToInt '4'
4
*Main> charToInt '5'
5
*Main> charToInt '6'
6
*Main> charToInt '7'
7
*Main> charToInt '8'
8
*Main> charToInt '9'
9
*Main> charToInt '0'
0
```

Raw Text Input

```
charToInt '1'
charToInt '2'
charToInt '3'
charToInt '4'
charToInt '5'
charToInt '6'
charToInt '7'
charToInt '8'
charToInt '9'
charToInt '0'
```

Supplemental (Assume following trees for rest of the parts)

treeOne = Node "Zamzung SolarSystem" (Node "Zamzung SolarSystem.1" Empty Empty Empty) (Node "Zamzung SolarSystem.2" Empty Empty Empty) (Node "Zamzung SolarSystem.3" Empty Empty Empty)

```
*Main> treeOne
Node "Zamzung SolarSystem" (Node "Zamzung SolarSystem.1" Empty Empty Empty) (Node "Zamzung SolarSystem.2" Empty Empty Empty)
(Node "Zamzung SolarSystem.3" Empty Empty Empty)
```

treeTwo = Node "Pear kOS" (Node "Pear kOS.1" Empty (Node "Pear kOS.1.2" Empty Empty (Node "Pear kOS.1.2.3" Empty Empty Empty)) Empty) Empty (Node "Pear kOS.3" (Node "Pear kOS.3.1" Empty Empty Empty) Empty Empty)

```
*Main> treeTwo
Node "Pear kOS" (Node "Pear kOS.1" Empty (Node "Pear kOS.1.2" Empty Empty (Node "Pear kOS.1.2.3" Empty Empty Empty)) Empty)
Empty (Node "Pear kOS.3" (Node "Pear kOS.3.1" Empty Empty Empty) Empty Empty)
```

treeThree = Node "Suum" Empty (Node "Suum.2" (Node "Suum.2.1" Empty (Node "Suum.2.1.2" Empty Empty Empty) (Node "Suum.2.1.3" (Node "Suum.2.1.3.1" Empty Empty Empty) Empty Empty)) (Empty) (Node "Suum.2.3" Empty Empty Empty)) (Node "Suum.3" Empty (Node "Suum.3.2" (Node "Suum.3.2.1" Empty Empty (Node "Suum.3.2.1.3" Empty (Node "Suum.3.2.1.3.2" Empty Empty Empty) Empty)) Empty (Node "Suum.3.2.3" Empty Empty Empty)) Empty)

```
*Main> treeThree
Node "Suum" Empty (Node "Suum.2" (Node "Suum.2.1" Empty (Node "Suum.2.1.2" Empty Empty Empty) (Node "Suum.2.1.3" (Node "Suum.2.1.3.1" Empty
 Empty Empty) Empty Empty)) Empty (Node "Suum.2.3" Empty Empty Empty)) (Node "Suum.3" Empty (Node "Suum.3.2" (Node "Suum.3.2.1" Empty Empty
 (Node "Suum.3.2.1.3" Empty (Node "Suum.3.2.1.3.2" Empty Empty Empty) Empty)) Empty (Node "Suum.3.2.3" Empty Empty Empty)) Empty)
```

Part Two

Sample Run

```
*Main> insertNode treeOne "Zamzung SolarSystem.3"
NodeExists
*Main> insertNode treeOne "Zamzung SolarSystem.3.1.2"
NotReachable
*Main> insertNode treeOne "Zamzung SolarSystem.3.1"
Node "Zamzung SolarSystem" (Node "Zamzung SolarSystem.1" Empty
 Empty Empty) (Node "Zamzung SolarSystem.2" Empty Empty Empty)
 (Node "Zamzung SolarSystem.3" (Node "Zamzung SolarSystem.3.1"
 Empty Empty Empty) Empty Empty)
```

```
*Main> insertNode treeTwo "Pear kOS.1"
NodeExists
*Main> insertNode treeTwo "Pear kOS.2"
Node "Pear kOS" (Node "Pear kOS.1" Empty (Node "Pear kOS.1.2"
Empty Empty (Node "Pear kOS.1.2.3" Empty Empty Empty)) Empty)
(Node "Pear kOS.2" Empty Empty Empty) (Node "Pear kOS.3" (Node
 "Pear kOS.3.1" Empty Empty Empty) Empty Empty)
```

```
*Main> insertNode treeThree "Suum.3.2.1.3.2"
NodeExists
*Main> insertNode treeThree "Suum.3.2.1.3.2.3"
Node "Suum" Empty (Node "Suum.2" (Node "Suum.2.1" Empty (Node "Suum.2.1.2" Empt
y Empty Empty) (Node "Suum.2.1.3" (Node "Suum.2.1.3.1" Empty Empty Empty) Empty
 Empty)) Empty (Node "Suum.2.3" Empty Empty Empty)) (Node "Suum.3" Empty (Node
"Suum.3.2" (Node "Suum.3.2.1" Empty Empty (Node "Suum.3.2.1.3" Empty (Node "Suu
m.3.2.1.3.2" Empty Empty (Node "Suum.3.2.1.3.2.3" Empty Empty Empty)) Empty)) E
mpty (Node "Suum.3.2.3" Empty Empty Empty)) Empty)
*Main> insertNode treeThree "Suum.3.2.1.3.1.1"
NotReachable
```

Raw Text Input

```
insertNode treeOne "Zamzung SolarSystem.3"
insertNode treeOne "Zamzung SolarSystem.3.1.2"
insertNode treeOne "Zamzung SolarSystem.3.1"
```

```
insertNode treeTwo "Pear kOS.1"
insertNode treeTwo "Pear kOS.2"
```

```
insertNode treeThree "Suum.3.2.1.3.2"
insertNode treeThree "Suum.3.2.1.3.2.3"
insertNode treeThree "Suum.3.2.1.3.1.1"
```

Part Three

Sample Run

```
*Main> totalNodes treeOne
4
*Main> totalNodes treeTwo
6
*Main> totalNodes treeThree
13
```

Raw Text Input

totalNodes treeOne

totalNodes treeTwo

totalNodes treeThree

Part Four

Sample Run

```
*Main> height treeOne
2
*Main> height treeTwo
4
*Main> height treeThree
6
```

Raw Text Input

height treeOne

height treeTwo

height treeThree

Part Five

Sample Run

```
*Main> levelcount treeOne 0
1
*Main> levelcount treeOne 1
3
*Main> levelcount treeOne 2
0
*Main> levelcount treeOne 3
0
```

Raw Text Input

levelcount treeOne 0
levelcount treeOne 1
levelcount treeOne 2
levelcount treeOne 3

Sample Run

```
*Main> levelcount treeTwo 0
1
*Main> levelcount treeTwo 1
2
*Main> levelcount treeTwo 2
2
*Main> levelcount treeTwo 3
1
*Main> levelcount treeTwo 4
0
```

Raw Text Input

levelcount treeTwo 0
levelcount treeTwo 1
levelcount treeTwo 2
levelcount treeTwo 3
levelcount treeTwo 4

Sample Run

```
*Main> levelcount treeThree 0
1
*Main> levelcount treeThree 1
2
*Main> levelcount treeThree 2
3
*Main> levelcount treeThree 3
4
*Main> levelcount treeThree 4
2
*Main> levelcount treeThree 5
1
*Main> levelcount treeThree 6
0
```

Raw Text Input

levelcount treeThree 0
levelcount treeThree 1
levelcount treeThree 2
levelcount treeThree 3
levelcount treeThree 4
levelcount treeThree 5
levelcount treeThree 6

Part Six

Sample Run

```
*Main> findNode treeOne "Zamzung SolarSystem"
Node "Zamzung SolarSystem" (Node "Zamzung SolarSystem.1" Empty
 Empty Empty) (Node "Zamzung SolarSystem.2" Empty Empty Empty)
 (Node "Zamzung SolarSystem.3" Empty Empty Empty)
*Main> findNode treeOne "Zamzung SolarSystem.1"
Node "Zamzung SolarSystem.1" Empty Empty Empty
*Main> findNode treeOne "Zamzung SolarSystem.1.2"
NodeNotFound
```

Raw Text Input

```
findNode treeOne "Zamzung SolarSystem"
findNode treeOne "Zamzung SolarSystem.1"
findNode treeOne "Zamzung SolarSystem.1.2"
```

```
*Main> findNode treeTwo "Pear kOS.1"
Node "Pear kOS.1" Empty (Node "Pear kOS.1.2" Empty Empty (Node
 "Pear kOS.1.2.3" Empty Empty Empty)) Empty
*Main> findNode treeTwo "Pear kOS.1.2.3"
Node "Pear kOS.1.2.3" Empty Empty Empty
*Main> findNode treeTwo "Pear kOS.1.2.3.2"
NodeNotFound
```

```
findNode treeTwo "Pear kOS.1"
findNode treeTwo "Pear kOS.1.2.3"
findNode treeTwo "Pear kOS.1.2.3.2"
```

```
*Main> findNode treeThree "Suum.3.2.1.3.2"
Node "Suum.3.2.1.3.2" Empty Empty Empty
*Main> findNode treeThree "Suum.3.2.1.3"
Node "Suum.3.2.1.3" Empty (Node "Suum.3.2.1.3.2" Empty Empty Empty) Empty
*Main> findNode treeThree "Suum.3.2"
Node "Suum.3.2" (Node "Suum.3.2.1" Empty Empty (Node "Suum.3.2.1.3" Empty
(Node "Suum.3.2.1.3.2" Empty Empty Empty) Empty)) Empty (Node "Suum.3.2.3"
 Empty Empty Empty)
*Main> findNode treeThree "Suum.3.1"
NodeNotFound
```

```
findNode treeThree "Suum.3.2.1.3.2"
findNode treeThree "Suum.3.2.1.3"
findNode treeThree "Suum.3.2"
findNode treeThree "Suum.3.1"
```