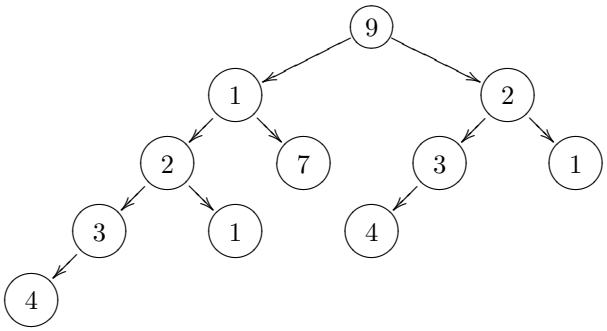
METU NCC,
CNG 242, Spring 2010,
Homework 1

Due 16^{th} March 2010

In this homework you will deal with recursive data structures in Haskell. The following Haskell data definition defines a binary tree:

```
data Tree alpha = Empty | Node (alpha,(Tree alpha),(Tree alpha)) deriving Show
```

Consider the following tree:



The tree above is represented by the following value:

```
Node (9,Node (1,Node (2,Node (3,Node (4,Empty,Empty),Empty),
                        Node (1,Empty,Empty)),Node (7,Empty,Empty)),
        Node (2,Node (3,Node (4,Empty,Empty),Empty),
                Node (1,Empty,Empty)))
```

You are asked to write the following functions:

```
get :: Tree alpha -> [alpha] -> Maybe (Tree alpha)
set :: Tree alpha -> [alpha] -> Tree alpha -> Tree alpha
```

`Maybe alpha` is a built-in type and it has two values, either `Nothing` or `Just value`. It is used to show a value or a failure on a single type.

Your purpose in get and set functions is to access nodes and subtrees of the tree by a list. In order to access nodes a list of values you need to traverse starting from the root is given. For example `[9,1]` addresses the left subtree in the figure, `[9,1,7]` addresses the left then right subtree which is a leaf node with 7. `[9,1,2,1]` gives the left,left,right subtree which is the leaf node with 1.

get gets a tree and a list of values as a address and returns the subtree on that address. Since the address does not have to contain valid values, if it cannot be found `Nothing` is returned. In success, `Just subtree` is returned.

set gets a tree, a list of values, another tree and returns a tree. It replaces the addressed subtree with the third parameter tree in the first parameter tree. The same tree returned if address cannot be found.

Sample run (assume t is the tree in the example, '-' is the prompt):

```
- get t4 [9,1]
Just (Node (1,Node (2,Node (3,Node (4,Empty,Empty),Empty),Node
(1,Empty,Empty)),Node (7,Empty,Empty)))

- get t4 [9,2,3]
Just (Node (3,Node (4,Empty,Empty),Empty))

- get t4 [9,1,2,1]
Just (Node (1,Empty,Empty))

- get t4 [9,1,2,1,3]
Nothing

- get t4 [9,5,1]
Nothing

- set t4 [9,1] (Node (11, Empty, Empty))
```

```
Node (9,Node (11,Empty,Empty),Node (2,Node (3,Node
(4,Empty,Empty),Empty),Node (1,Empty,Empty)))

- set t4 [9] Node (11,Node (10,Empty,Empty),Empty)
Node (11,Node (10,Empty,Empty),Empty)

- set t4 [9,2,3] t4
Node (9,Node (1,Node (2,Node (3,Node (4,Empty,Empty),Empty),Node
(1,Empty,Empty)),Node (7,Empty,Empty)),Node (2,Node (9,Node (1,Node (2,Node
(3,Node (4,Empty,Empty),Empty),Node (1,Empty,Empty)),Node
(7,Empty,Empty)),Node (2,Node (3,Node (4,Empty,Empty),Empty),Node
(1,Empty,Empty))),Node (1,Empty,Empty)))

- set t4 [9,1,2,1,3] (Node (10, Empty, Empty))       -- no change
Node (9,Node (1,Node (2,Node (3,Node (4,Empty,Empty),Empty),Node
(1,Empty,Empty)),Node (7,Empty,Empty)),Node (2,Node (3,Node
(4,Empty,Empty),Empty),Node (1,Empty,Empty)))

- set (Node (1, Empty, Empty)) [2] (Node (10, Empty, Empty))
Node (1, Empty, Empty)

- set (Node (2, Empty, Empty)) [2] (Node (10, Empty, Empty))
Node (10, Empty, Empty)
```

The function is given to display trees in an easy to read format (development purpose only do not include in your submission!):

```
prettyprint t =
   let pretty2 Empty _ _ ind = ind ++ "*"
       pretty2 (Node (x, left, right)) first second ind =
                          let nfirst = first ++ " |    "
                              nsecond = second ++ " |    "
                              firstnull = first ++ "      "
                              secondnull = second ++ "      "
                              innf = first ++ " /---"
                              inns = second ++ " \\---"
                          in (pretty2 right firstnull nfirst innf) ++ "\n"
                              ++ ind ++ (show x) ++ "+\n"
                              ++ (pretty2 left nsecond secondnull inns)
   in putStr (pretty2 t "" "" "" ++"\n")
```

Post your submission on METU Online. You can ask questions on METU Online forums. Note that all homeworks should be result of you individual work. No code sharing is allowed. Cheating policy: cheaters (both parties) take 0 from all 5-6 homeworks. Commissioning/trying to commission someone else to do your homework is subject to disciplinary action.