Middle East Technical University

Department of Computer Engineering

CENG 242

Programming Language Concepts
2023-2024 Spring
Programming Exam 2

Due: 24 March 2024 11:59
Submission: **via ODTUClass**

1 General Specifications

- There are four questions in this Programming Exam, for each question you will implement a given Haskell function.
- Definitions of the data types, signatures of the functions, their explanations and specifications are given in the following section. Read them carefully.
- Make sure that your implementations comply with the function signatures.
- You may define helper function(s) as needed.

2 Data Types

We only have one custom data type in the homework. It is a tree that holds a key-value pair in each of its nodes. Each node can have any number of children, including 0 (in which case it is a leaf). Its definition is as follows:

```
data Tree k v = EmptyTree | Node k v [Tree k v]
```

Here is an example use, and an illustration to match:

```
Node 2 3 [Node (-2) (-4) [], Node (-1) 7 [], Node 2 (-3) []]
```

This is a tree that holds pairs of integers (so both `k` and `v` are `Integer`), with three leaves. It is drawn below:

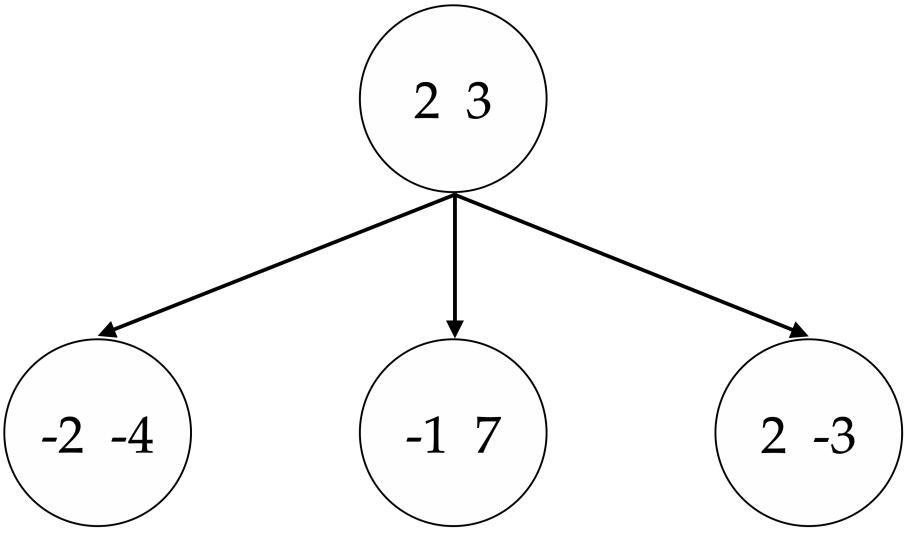

Figure 1: A visual representation of the tree given above

3 Functions

3.1 selectiveMap (25 pts.)

This function takes two functions and a list as arguments, and applies the second function to only those members of the list that satisfy the first function. The elements that don't satisfy the first function are left in the list unchanged.

This function has the signature:

```
selectiveMap :: (a -> Bool) -> (a -> a) -> [a] -> [a]
```

Here is an example function call:

```
ghci> selectiveMap even (\x -> x*x) [1,2,3,4,5,6,7]
[1,4,3,16,5,36,7]
```

3.2 tSelectiveMap (25 pts.)

This function is a selective map operation for our custom tree type. It takes two functions and a tree of key-value pairs as arguments, and applies the second function to the values of the nodes whose key satisfy the first function. It's signature is:

```
tSelectiveMap :: (k -> Bool) -> (v -> v) -> Tree k v -> Tree k v
```

Here is an example function call:

```
ghci> t = Node 2 3 [Node (-2) (-4) [], Node (-1) 7 [], Node 2 (-3) []]
ghci> tSelectiveMap (<0) (\x -> x*x) t
Node 2 3 [Node (-2) (16) [], Node (-1) 49 [], Node 2 (-3) []]
```

3.3 tSelectiveMappingFold (25 pts.)

This function is similar to the previous one, except that it also combines all the values that is mapped into a single value using a given combiner function (similar to how `foldl`, `foldr` etc. work). Here's its type signature:

```
tSelectiveMappingFold :: (k -> Bool) -> (k -> v -> r) -> (r -> r -> r) ->
                         r -> Tree k v -> r
```

Each of its arguments are as follows (in the given order):

- A predicate (a function that returns a boolean) to determine which values to use. It takes the key of a node as argument.
- The mapping function to be applied to the values satisfying the predicate. It takes both the key and the value of the node as arguments.
- The combining function to reduce the resulting values to a single value.
- The right identity element (also called the neutral element) of the combiner function, which you'll use as an accumulator. Having this will make writing this function much simpler.
- The tree to which we are applying the operation

When using the combiner function to combine a node and it's children, you should start with the node itself, then combine the children of that node in the same order that they appear in the list. When combining, accumulator should be on the left (so when you call the combiner, the new element is the second argument and the accumulator is the first argument).

Here is an example function call:

```
ghci> t = Node (-2) 3 [Node (-2) (-4) [], Node (-1) 7 [], Node 2 (-3) []]
ghci> tSelectiveMappingFold (<0) (\x y -> y) (+) 0 t
6
```

3.4 searchTree (25 pts.)

In the last question, we've written an very powerful function. Let's see what we can build with it. Complete the definition of the `searchTree` function by defining each of `a`, `b`, `c`, `d`, so that when given a tree, it returns a function that can search for keys in that tree. Don't forget that each of `a`, `b`, `c`, `d` can either be a value or a function. Do not change the definition of `searchTree`! Here's its definition:

```
searchTree :: (Eq k, Eq v) => v -> Tree k v -> (k -> v)
searchTree def = tSelectiveMappingFold a b c d
    where a =
          b =
          c =
          d =
```

where `def` is the default value that should be returned if the searched key isn't in the tree.

Here are some example function calls:

```
ghci> t = Node (-2) 3 [Node (-2) (-4) [], Node (-1) 7 [], Node 2 (-3) []]
ghci> searchT = searchTree 0 t
ghci> searchT (-1)
7
ghci> searchT 12
0
ghci> searchT 2
-3
ghci> searchT 9
0
```

note how, when given a tree, the function returns another function that we can give a key to get the corresponding value (if it exists). If the key isn't in the tree, we get the default value (0 in this case).

4 Regulations

1. **Implementation and Submission:** The template file named "PE2.hs" is available in the Virtual Programming Lab (VPL) activity called "PE2" on OdtuClass. At this point, you have two options:

   - You can download the template file, complete the implementation and test it with the given sample I/O (and also your own test cases) on your local machine. Then submit the same file through this activity.
   - You can directly use the editor of VPL environment by using the auto-evaluation feature of this activity interactively. Saving the code is equivalent to submit a file.

   If you work on your own machine, make sure that your implementation can be compiled and tested in the VPL environment after you submit it.
   There is no limitation on online trials or submitted files through OdtuClass. The last one you submitted will be graded.

2. **Cheating: We have zero tolerance policy for cheating.** People involved in cheating (any kind of code sharing and codes taken from internet included) will be punished according to the university regulations.

3. **Evaluation:** Your program will be evaluated automatically using "black-box" technique so make sure to obey the specifications. No erroneous input will be used. Therefore, you don't have to consider the invalid expressions.

- **Important Note:** The given sample I/O's are only to ease your debugging process and NOT official. Furthermore, it is not guaranteed that they cover all the cases of

required functions. As a programmer, it is your responsibility to consider such extreme cases for the functions. Your implementations will be evaluated by the official test cases to determine your ***actual*** grade after the deadline.