Exercise 1

NTNU

TDT4165 fall 2018

1 Theory solutions

- When implementing **treeMaximum** in src/Tree.hs, do you have to return a Maybe? Why or why not?
  You do not have to return a Maybe, because the tree we defined cannot be empty.

- After making **Tree a** an instance of **Foldable**, we can sum it by calling **sum <ourTree>**. This also applies to **maximum** and several other functions. How do we get these functions "for free"?
  This is because these functions (all with Foldable t in their type signature) is defined in terms of folds and will work as long as a type is Foldable.

- Can we make Complex an instance of Ord? The minimal complete definition of Ord is **compare** — **(<=)**, meaning that you have to implement either compare or (<=). Use Hoogle to inspect these functions in more detail, if needed. Provide the reasoning behind your answer.
  You can define Complex Ord by saying that **a + bi < c + di** is true if a < c, for example. But this would not really make sense. You can also define it by the magnitude of the vector, but 1 and i has the same magnitude although they are not equal, which is a problem. Both "yes" and "no" are valid answers to this question, provided that they have reasoned about it and can demonstrate this.

- Load your file and write **negate (Complex 3 4)**, if you implemented (-) in 2.7. If you implemented negate, type **(Complex 3 4) - (Complex 2 7)**. What happens? Why does this happen?
  The other function is automagically implemented by this definition:

```
1    x - y              = x + negate y
2    negate x           = 0 - x
```

- Try to make a Pos instance of the type synonym Position. What kind of error message do you get? From the error message, do you think it's possible to achieve this?

You get a message that you need to use TypeSynonymInstances and FlexibleInstances if you want to permit making a Pos Position instance. These are language extensions, where TypeSynonymInstances allows you to make instances of type synonyms (denoted by **type**, another example: **type String = [Char]**) and FlexibleInstances allows you to make instances where distinct types can appear more than once (e.g. Pos [Char] or Pos (Double, Double)).

- You probably used record syntax when creating your types, and you might have run into issues if you tried to name fields identically. What kind of error message do you get? Why do you get this error message? Hint: Remove one of the entries, load the file and ask for the type of a field. What's the scope of the entry?
  Fields in a record type are functions. The scope of the identifiers is the entire module and all modules that import them, given that they are exported where they are defined. If we allowed several fields with the same name, we would get name collisions.