CSCC24 2023 Summer – Assignment 1
Due: Wednesday June 7, 11:59PM
This assignment is worth 10% of the course grade.

In this assignment, you will write an induction proof for a program, and you will work in Haskell with algebraic data types and implement interesting recursive algorithms.

As usual, you should also aim for reasonably efficient algorithms and reasonably organized, comprehensible code.

Code correctness (mostly auto-testing) is worth 90% of the marks; code quality is worth 10%.

Question 1: Induction Proof (4 marks)

Summing a list of numbers can be helped by this helper function:

```
h [] acc = acc
h (x:xt) acc = h xt (acc + x)
```

Use induction on the input list to prove:

$$\forall xs \cdot \forall acc \cdot h\ xs\ acc = acc + (\text{sum of } xs)$$

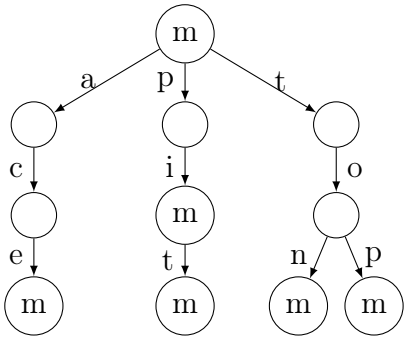
Note that the induction step retains "$\forall acc$", and it will help you be rigorous.

Please create and hand in a text file proof.txt for this question. You can write "forall" if you can't write "∀".

Question 2: Trie

A "trie" is a data structure that stores a finite set of strings or a dictionary (finite map) of string keys. (In this assignment, we store a set.) The name "trie" came from "retrieval". Most confusingly (when you pronounce it), it is a tree structure indeed. Here is an example (*albertTrie* in the starter code): On the left is the set, in the middle is the trie picture, and on the right is how the provided *printTrie* function prints it.

member
(empty string)
ace
pi
pit
top
ton




```
"" member
  a
    c
      e member
  p
    i member
      t member
  t
    o
      n member
      p member
```

In general, a trie node has:

- a marker saying "member" or "not member"
- a collection of character-labelled edges to child nodes (in this assignment, a list of (*char*, *node*) tuples)

A string is a member of a trie iff starting at the root, there is a path of edges corresponding to the string, and it ends at a node marked as "member".

In this assingment, tries are represented by

```
data Trie a = Node Membership [(Char, Trie)]
data Membership = Member | NotMember
```

Since members that share a common prefix (e.g., "ton" and "top" share prefix "to") become nodes that share a common ancestor path, tries are great for supporting autocompletion! Let's do this.

Please hand in Complete.hs for the following.

2(a). Enumeration (8 marks)

Implement in Complete.hs

```
enumerate :: Trie -> [String]
```

to output all members of the input trie.

We assume that in a (*char*, *node*)-list, each character occurs at most once.

The output order does not matter; the tester sorts your answer before comparing with the model answer.

2(b). Completion (5 marks)

This is our autocompletion function! Implement in Complete.hs

```
complete :: Trie -> String -> [String]
```

to output all members of the input trie that has the input string as a prefix. The output order does not matter.

This can be easily done by recursion on the input string and eventually calling `enumerate`. You will also find the standard library function `lookup` handy:

```
lookup 'x' [('a', foo), ('x', bar), ('e', meh)] = Just bar
lookup 'y' [('a', foo), ('x', bar), ('e', meh)] = Nothing
```

End of questions.