Unassessed Learning Exercise, Week 3

Functional Programming
School of Computer Science, University of Birmingham
Lecturer: Martin Escardo
Teaching Assistants: Bram Geron, Cory Knapp, Wai Tak Cheung

Submission instructions:

- Produce a single Haskell file with the solution to all programming questions.
- The non-programming questions should be answered in the same file as a comment.

Exercises

1. Morse code

   This exercise is a variation of the lecture codes `bittree.hs` and `prefixfreebittree.hs`. You may want to familiarise yourself with the two files before you start doing this exercise.

   Look at the Wikipedia page for **Morse code** (`https://en.wikipedia.org/wiki/Morse_code#Representation.2C_timing_and_speeds`). According to this page, International Morse code is composed of five elements:

   - short mark, dot or "dit" (·) – one unit long;
   - longer mark, dash or "dah" (-) – three units long;
   - inter-element gap between the dots and dashes within a character – one unit long;
   - short gap (between letters) – three units long;
   - medium gap (between words) – seven units long.

   We represent a Morse unit as either a beep or silence:

```
data MorseUnit = Beep | Silence    deriving (Eq, Show)
```

   Then Morse code can be represented by a list of these units, i.e. the type `[MorseUnit]`. Now we can write constants for a short mark "·" (`dit`), a long mark "-" (`dah`), a gap between letters (`shortGap`) and a gap between words (`mediumGap`):

```
dit, dah, shortGap, mediumGap :: [MorseUnit]
dit       = [Beep, Silence]
dah       = [Beep, Beep, Beep, Silence]
shortGap  = replicate (3-1) Silence
mediumGap = replicate (7-3) Silence
```

   Note that the length of `shortGap` and `mediumGap` are made so that a `shortGap` has the correct length 3 if following a `dit` or `dah` and a `mediumGap` has length 7 if following a `dit` or `dah` followed by a `shortGap`.

   (1) Write a function

```
encode :: String -> [MorseUnit]
```

   that encodes a given string to Morse code. Note that unlike the encoding shown on Wiki, our Morse code will end with a `shortGap`. For example, "E" shall be encoded to [Beep, Silence, Silence, Silence]. You may want to write some helper functions in order to solve the problem.

```
codeWord :: String -> [MorseUnit]
```

   is a function that, given a string with a single word in it, produces the Morse code for that word. You can use `codeSymbol` and `shortGap` to make it so that there is a `shortGap` after each letter (even the last one).

```
codeText :: [String] -> [MorseUnit]
```

is a function that, given a list of strings, encodes each string of the list into Morse code. Put a `mediumGap` after each word (but not the last one), and concatenates the results. Checkout a function named `words` which can split a text into a list of words in Hoogle (`www.haskell.org/hoogle/`)

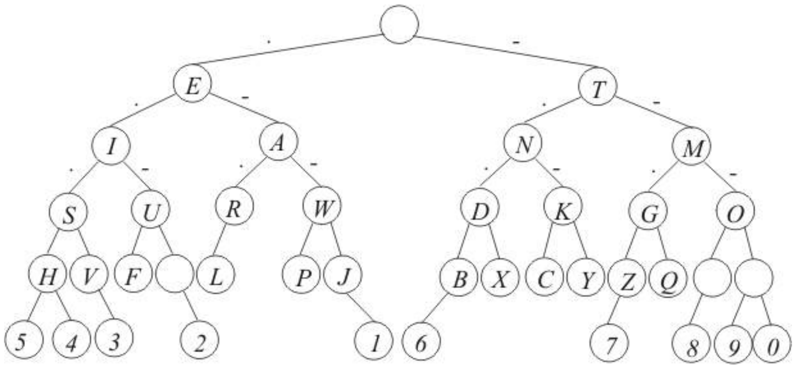
(2) Write a decoder

```
decode :: [MorseUnit] -> String
```

for a given Morse code using the provided morse table.

```
table :: MorseTable
table = [(dit ++ dah, ’A’),
         (dah ++ dit ++ dit ++ dit, ’B’),
         (dah ++ dit ++ dah ++ dit, ’C’),
         ...
```

Besides a table, we can also store the morse code interpretation in a tree where a left branch is a `dit` and a right branch is a `dah` as shown below.



Here is the type of such a tree.

```
data MorseTree = Nil
               | Leaf Char
               | Branch1 Char MorseTree MorseTree
               | Branch0 MorseTree MorseTree
  deriving Show
```

Note that `Branch1` corresponds to those branches with exactly one label and `Branch0` represents those branches without a lable.

(3) Write a function

```
toTree :: MorseTable -> MorseTree
```

that translates a given MorseTable into a MorseTree.

(4) Write a function

```
toTable :: MorseTree -> MorseTable
```

that does the opposite.

Try to write a function to test if

```
elem(toTable (toTree table)) == elem(table).
```