Comp0002 Haskell
Lab exercise sheet 2

Recursive functions

1. Write some functions from last week using recursive function definitions rather than list comprehension. You may find it useful to refer to LabSheet1 to remind yourself of the specification of these functions. You can import the library functions `toUpper` and `toLower` for changing case by using the command `import Data.Char` at the head of your file. Call the file `LabSheet2.hs`

   - a function `inRange :: Int -> Int -> [Int] -> [Int]` to return all numbers in the input list within the range given by the first two arguments (inclusive)
   - a function `countPositives` to count the positive numbers in a list (the ones strictly greater than 0)
   - a function `capitalised :: String -> String` which, given a word, capitalises it. That means that the first character should be made uppercase and any other letters should be made lowercase
   - Using the function `capitalised` from the previous question, write a function `title :: [String] -> [String]` which, given a list of words, capitalises them as a title should be capitalised. The proper capitalisation of a title (for our purposes) is as follows: The first word should be capitalised. Any other word should be capitalised if it is at least four letters long.

2. Define a recursive function `isort :: Ord a => [a] -> [a]`, that implements insertion sort, which can be specified by the following two rules:

   - The empty list is already sorted;
   - Non-empty lists can be sorted by sorting the tail and inserting the head into the result.

3. Define a recursive function `merge :: Ord a => [a] -> [a] -> [a]` that merges two sorted lists to give a single sorted list. For example: `merge [2,5,6] [1,3,4]` produces `[1,2,3,4,5,6]`.

4. Define a recursive function `msort :: Ord a => [a] -> [a]` that implements merge sort, which can be specified by the following two rules:

   - Lists of length ≤ 1 are already sorted;
   - Other lists can be sorted by sorting the two halves and merging the resulting lists.

5. **A simple cipher**. One of the earliest encryption methods was to cyclicly shift the letters of the alphabet by a fixed amount, an offset between 0 and 26. In what follows you write functions to do this.

   - We can cycle a list by a fixed amount by taking the given number of items off the front and adding them to the back of the list. Write a function called `rotor`so `rotor 6 "ABCDEFGHIJKLMN"` produces `"GHIJKLMNABCDEF"`. Give the type as well as the definition for `rotor`. Make sure it returns an error message if the offset number is less than 0 or bigger than or equal to the length of the list.
   - Assume keys are given in terms of upper case letters only. Use the function `rotor` to write a function called `makeKey` that produces a list of 26 pairs of upper case letters which give the cypher key. `makeKey` will have type `makeKey :: Int -> [(Char,Char)]` so it takes an offset amount and returns a list of pairs where the first member of a pair is a member of the alphabet and the second is the member of the alphabet that the offset pairs the first one with.
   - Write a function `lookUp :: Char -> [(Char,Char)] -> Char` that finds a pair in a key by its first character and returns its second character. If the character is not in the key it should return it unchanged.
   - Write a function `encipher :: Int -> Char -> Char` that encrypts a single upper case character using a given offset. For example `encipher 5 'C'` produces `'H'`.
   - Text encrypted by a cipher is conventionally written in uppercase and without punctuation. Write a function `normalise :: String -> String` that converts a string to upper case, removing spaces and all characters other than letters and digits.
   - Write a function `encipherStr :: Int -> String -> String` that normalises a string and then encrypts it. For example `encipherStr 18 "We move at 1500 hours"` becomes `"OWEGNWSL1500ZGMJK"`
   - Can you write a function that decrypts any encrypted text that has been encrypted by this method, using a brute force attack?