COM2108: Functional Programming – Autumn 2019
Assignment 1: Substitution Ciphers

This assignment is 30% of the assessment for COM2108

1 Cipher Systems

In cipher systems the aim is to encode a message (a String) in such a way that it will be meaningless to anyone who does not have the key which specifies how the transformation from the original plain text was done.

In a substitution cipher, each character is replaced by another, using a fixed mapping, e.g.

```
Plain     ABCDEFGHIJKLMNOPQRSTUVWXYZ
Cipher    EKMFLGDQVZNTOWYHXUSPAIBRCJ
```

So that the message `"SPANGLES"` would be transmitted as `"SHEWDTLS"`. The key is the mapping.

All our messages will contain upper case letters only. Punctuation is done by words – `"STOP"`, `"COMMA"` etc. Numbers are spelt out.

1.1 Your Tasks

In your solutions, you should use the following Haskell data structure *where appropriate*:

```
type Cipher = String -- a substitution cypher for A,B..Z
```

This type has been predefined in the help file. You should not redefine it, but download and use the help file as described below. You should not make any changes to the help file, and you should not submit the help file. **Your submission will be tested using the original help file.**

Please note: The *correctness* of your solutions to the listed tasks will be determined by automated testing. In order for this testing to succeed, every function must be implemented exactly as specified. If, for example, you write a function with the wrong number of arguments, or expect the arguments to be given in a different order, this testing **will fail** and you will receive **no credit** for the correctness element of your work.

1. Define and test a Haskell function `validateCipher` to validate a cipher – which must contain each letter once and once only. The input is a cipher, and the return value should be a boolean value.

A common variation is to add an **offset** of n characters : the cipher moves n places to the right and wraps around, so if $n = 2$ we have

```
Plain     ABCDEFGHIJKLMNOPQRSTUVWXYZ
Cipher    CJEKMFLGDQVZNTOWYHXUSPAIBR
```

2. Define and test a Haskell Function `encode` which takes a cipher, an offset and a character, and returns the corresponding encoded character.

3. Define and test a Haskell function `encodeMessage` which given a cipher, an offset, and a message, uses `encode` to encode and return the complete encoded message.

Substitution ciphers can be used in reverse. Returning to our original

```
Plain     ABCDEFGHIJKLMNOPQRSTUVWXYZ
Cipher    EKMFLGDQVZNTOWYHXUSPAIBRCJ
```

Reverse encoding is given the cipher and returns the plain – this is how you'd decode the message if you had the key. So `"SHEWDTLS"` would be reverse encoded to `"SPANGLES"`

4. Write and test `reverseEncode` (which takes as input input a cipher, an offset and an encoded character, and returns the plain character) and `reverseEncodeMessage` (taking as input a cipher, an offset and an encoded message to return the plain text message) corresponding to *encode* and *encodeMessage* above.

A substitution cipher can be broken using knowledge of letter frequencies. In English the most common letter is E so the most frequently occurring letter in the coded message is most likely (though not certain) to decode to E. The next most frequent is T, then A and so on – the full list is given in the help file on Blackboard – see below.

5. Write and test a function `letterStats`, which given a message, returns the percentage of each letter occurring in this message as a list of (Char, Int) tuples, e.g. `letterStats "STDDWSD"` should return

   `[(’D’,43),(’S’,29),(’W’,14),(’T’,14)]`

It will be useful if letters with zero scores are removed and the remainder are returned in decreasing order of frequency in the message. You can use `mergeSort`, which is in the help file (see below).

Results from *letterStats* can be used to make guesses for letter encodings, which can then be substituted back into the coded message.

6. Write and test `partialDecode`, which is given a list of guesses for letters in the message and returns the message with these guesses filled in, in lower case. E.g. if the message is `"DXPWXW"` and the guesses are that E ciphers to X and S ciphers to W then partialDecode should return `"DePses"`. **The guesses are supplied by hand**, as a list of (Char, Char) items where the first Char is the plain letter and the second Char is the encoded letter guess, so for the given example, the input would be `[(’E’,’X’),(’S’,’W’)]`. The output should be a string with the *unchanged* letters remaining in uppercase, but the *guessed* substitutions in lowercase.

7. To test `partialDecode`, decipher the message `mystery` given in the help file. Supply your answer as a comment below your definition of `partialDecode`.

2 The Help File

The file AssignmentHelp.hs in the COM2108 Blackboard area contains the predefined type mentioned above, as well as some functions might find useful (for instance to find the alphabetic position of a letter and to convert upper to lower case) and data you will need (for instance the mystery message). You can import it into your code by copying it to the same directory as your own module and then starting like so

```
Module Ciphers where
 import AssignmentHelp
```

3 Marking Scheme

Task	**Credit(%)**
Correctness of:	
validateCipher	10
encode	5
encodeMessage	5
reverseEncode	5
reverseEncodeMessage	5
letterStats	20
partialDecode	10
(subtotal)	**60**
Coding style	**20**
Documentation	**20**
Total	**100**

Table 1: Mark breakdown for assignment 1

As explained above, correctness of your code will be assessed using automated testing. This testing **will fail** if you do not write your functions exactly as specified. This means exactly the same names, exactly the same arguments (in exactly the same order) and exactly the same return type for the functions.

4 Submission

Work should be submitted via the appropriate link on Blackboard. Any work received after the deadline will have standard lateness penalties applied (see `https://sites.google.com/sheffield.ac.uk/comughandbook/general-information/assessment/late-submission`).

You should submit a single Haskell (`.hs`) file, with your name in an initial comment. As previously stated, you do not need to submit the help file; we will provide it for the testing (and so you should ensure that you make **no changes** to the help file).

In the comments for your code, you should clearly state any known limitations, or be clear about modifications that you made to address any identified limitations – the limitations are things that you should have identified when testing your work (or if you planned your work perfectly, that you identified before you started coding). If know *know* your code has limitations that you have been unable to address, you will achieve a better score if you clearly discuss those limitations in your comments than if you ignore them and your code fails test cases.

5 Individual Work

You are reminded that this is intended to be assessment of *your own, individual work*, in order to assess whether you have achieved with the learning outcomes for this module. If there is any suspicion that that you have utilised unfair means, the department will take appropriate action, which could lead to disciplinary procedures.

You should have had a discussion about this subject with your personal tutor in first year. If you need refreshing on the rules, please see the Computer Science Undergraduate Handbook entry on the topic at

`https://sites.google.com/sheffield.ac.uk/comughandbook/general-information/assessment/unfair-means`

and/or discuss the matter with your personal tutor.