Recursion & the Caesar Cipher
Informatics 1 – Introduction to Computation
Functional Programming Tutorial 3

Banks, Heijltjes, Fehrenbach, Melkonian, Sannella, Scott, Vlassi-Pandi, Wadler

Week 4
due 12:00 Tuesday 10 October 2023
tutorials on Thursday 12 and Friday 13 October 2023

You will not receive credit for your coursework unless you attend the corresponding tutorial session. Please email kendal.reid@ed.ac.uk if you cannot join your assigned tutorial.

Good Scholarly Practice: Please remember the good scholarly practice requirements of the University regarding work for credit. You can find guidance at the School page

http://web.inf.ed.ac.uk/infweb/admin/policies/academic-misconduct.

This also has links to the relevant University pages. Please do not publish solutions to these exercises on the internet or elsewhere, to avoid others copying your solutions.

1 The Caesar Cipher

When we talk about cryptography these days, we usually refer to the encryption of digital messages, but encryption actually predates the computer by quite a long period. One of the best examples of early cryptography is the Caesar cipher, named after Julius Caesar because he is believed to have used it, even if he didn't actually invent it. The idea is simple: take the message you want to encrypt and shift all letters by a given amount between 0 and 26, called the *offset.* For example: encrypting the sentence "THIS IS A BIG SECRET" with offset of 5, would result in "YMNX NX F GNL XJHWJY".

In this exercise you will implement a version of the Caesar cipher. You can use the list of library functions in the Appendix of this tutorial sheet, and any other library functions in `Data.Char` and `Data.List`.

Encrypting text

A character-by-character cipher such as a Caesar cipher can be represented by a *key*, a list of pairs. Each pair in the list indicates how one letter should be encoded. For example, a cipher for the letters A–E could be given by the list

```
[('A', 'C'), ('B', 'D'), ('C', 'E'), ('D', 'A'), ('E', 'B')] .
```

Although it's possible to choose any letter as the ciphertext for any other letter, this tutorial deals mainly with the type of cipher where we encipher each letter by shifting it the same number of spots around a circle, for the whole English alphabet.

We wrote a function `makeKey :: Int -> [(Char, Char)]` for you which will generate such a key for a given offset. Example:

```
Tutorial3> makeKey 5
 [('A','F'),('B','G'),('C','H'),('D','I'),('E','J'),('F','K'),
  ('G','L'),('H','M'),('I','N'),('J','O'),('K','P'),('L','Q'),
  ('M','R'),('N','S'),('O','T'),('P','U'),('Q','V'),('R','W'),
  ('S','X'),('T','Y'),('U','Z'),('V','A'),('W','B'),('X','C'),
  ('Y','D'),('Z','E')]
```

The cipher key shows how to encrypt all of the uppercase English letters. There are no duplicates: each letter appears just once amongst the pairs' first components (and just once amongst the second components).

Exercise 1

(a) Write a function

```
lookUp :: Char -> [(Char, Char)] -> Char
```

that finds a pair by matching its *first* component and returns that pair's *second* component. When you try to look up a character that does not occur in the cipher key, your function should leave it unchanged. Examples:

```
Tutorial3> lookUp 'B' [('A','F'), ('B','G'), ('C','H')]
'G'
Tutorial3> lookUp '9' [('A','F'), ('B','G'), ('C','H')]
'9'
```

Use *list comprehension* to implement it.

(b) Write an equivalent function `lookUpRec` using *recursion.*

(c) Write a function `prop_lookUp` which checks that the two functions are equivalent, and then use QuickCheck to test this.

Exercise 2

(a) Using `makeKey` and `lookUp`, write a function

```
encipher :: Int -> Char -> Char
```

that encrypts the given single character using the key with the given offset. For example:

```
Tutorial3> encipher 5 'C'
'H'
Tutorial3> encipher 7 'Q'
'X'
```

Exercise 3

(a) Text encrypted by a cipher is conventionally written in uppercase and without punctuation. Write a function

```
normalise :: String -> String
```

that converts a string to uppercase, removing all characters other than letters. Example:

```
Tutorial3> normalise "July 4th!"
"JULYTH"
```

Use *list comprehension*.

(b) Write an equivalent function `normaliseRec` using *recursion*.

(c) Check that your two functions are equivalent with a test function `prop_normalise`.

Exercise 4

Write a function

```
enciphers :: Int -> String -> String
```

that normalises a string and encrypts it, using your functions `normalise` and `encipher`. Example:

```
Tutorial3> enciphers 5 "July 4th!"
"OZQDYM"
```

Decoding a message

The Caesar cipher is one of the easiest forms of encryption to break. Unlike most encryption schemes commonly in use today, it is susceptible to a simple brute-force attack of trying all the possible keys in succession. The Caesar cipher is a *symmetric key* cipher: the key has enough information within it to use it for encryption as well as decryption.

Exercise 5

(a) Decrypting an encoded message is easiest if we transform the key first. Write functions

```
reverseKey :: [(Char, Char)] -> [(Char, Char)]
```

to reverse a key. This function should swap each pair in the given list. For example:

```
Tutorial3> reverseKey [('A','G'), ('B','H'), ('C','I')]
[('G','A'), ('H','B'), ('I','C')]
```

Use *list comprehension* to write the function.

(b) Write an equivalent function `reverseKeyRec` using *recursion*.

(c) Check that your two functions are equivalent with a test function `prop_reverseKey`.

Exercise 6

Write the functions

```
decipher :: Int -> Char -> Char
decipherStr :: Int -> String -> String
```

that decipher a character and a string, respectively, by using the key with the given offset. Remove any character that is not an upper case letter. For example:

```
Tutorial3> decipherStr 5 "OZQDYMx4"
"JULYTH"
```

2 Optional Material

Breaking the encryption

One kind of brute-force attack on an encrypted string is to decrypt it using each possible key and then search for common English letter sequences in the resulting text. If such sequences are discovered then the key is a candidate for the actual key used to encrypt the plaintext. For example, the words "the" and "and" occur very frequently in English text: in the *Adventures of Sherlock Holmes*, "the" and "and" account for about one in every 12 words, and there is no sequence of more than 150 words without either "the" or "and". Hence, if we try a key on a sufficiently long sequence of text and the result does not contain any occurrences of "the" or "and" then the key can be discarded as a candidate.

We use the Haskell function `isInfixOf` to determine whether a given string appears in another string.

```
Tutorial3> isInfixOf "amp" "Example"
True
Tutorial3> isInfixOf "xml" "Example"
False
```

Also look at the functions `isPrefixOf` and `isSuffixOf`.

Exercise 7

Write a function

```
candidates :: String -> [(Int, String)]
```

that decrypts the input string with each of the 26 possible keys and, when the decrypted text contains "THE" or "AND", includes the decryption key and the text in the output list.

```
Tutorial3> candidates "DGGADBCOOCZYMJHZYVMTOJOCZHVS"
[(5,"YBBVYWXJJXUTHECUTQHOJEJXUCQN"),
(14,"PSSMPNOAAOLKYVTLKHYFAVAOLTHE"),
(21,"ILLFIGHTTHEDROMEDARYTOTHEMAX")]
```

Use a *list comprehension.*

Strengthened Ceasar

As you have seen in the previous section, the Caesar Cipher is not a very safe encryption method. In this section, security will be upgraded slightly.

To help, we have provided a function `splitEachFive :: String -> [String]` that splits a string into substrings of length five, and fills out the last part with copies of the character `'X'` to make it as long as the others. (Look at the code, which uses the functions `take, drop :: Int -> [a] -> [a]`.)

```
Tutorial3> splitEachFive "Secret Message"
["Secre", "t Mes", "sageX"]
Tutorial3> splitEachFive ""
["XXXXX"]
```

You will also need to use the library function `transpose` from `Data.List` which switches the rows and columns of a list of lists all of the same length:

```
Tutorial3> transpose ["123","abc","ABC"]
["1aA","2bB", "3cC"]
```

Notice that if the rows in a list of lists are of the same length, transposing it twice returns the original one.

Exercise 8

Write a function `encrypt :: Int -> String -> String` that encrypts a string by first applying the Caesar Cipher, then splitting it into pieces of length five, transposing, and putting the pieces together as a single string.

Exercise 9

Write a function to decrypt messages encrypted in the way above.

Hint: The last action of the previous function is to put the transposed list of strings back together. You will need a helper function to undo this (it is not `splitEachFive`).

3 Really optional and unassessed, just for fun

Exercise 10

Another game, also called HaskellQuest, was produced by Eve Bogomil in 2022/2023 as her fourth-year project. It provides a fun way of learning about recursion and some other Haskell features. You can access the game through the link https://github.com/somethingololo. It works on Mac OS, Windows and Linux.

4 Appendix: Utility function reference

Note: for most of these functions you will need to import `Data.Char` or `Data.List`.
(This has already been done for you in `Tutorial3.hs`.)

`ord :: Char -> Int`
Return the numerical code corresponding to a character
Examples: `ord 'A' == 65` `ord '1' == 49`

`chr :: Int -> Char`
Return the character corresponding to a numerical code
Examples: `chr 65 == 'A'` `chr 49 == '1'`

`mod :: Int -> Int -> Int`
Return the remainder after the first argument is divided by the second
Examples: `mod 10 3 == 1` `mod 25 5 == 0`

`isAlpha :: Char -> Bool`
Return `True` if the argument is an alphabetic character
Examples: `isAlpha '3' == False` `isAlpha 'x' == True`

`isDigit :: Char -> Bool`
Return `True` if the argument is a numeric character
Examples: `isDigit '3' == True` `isDigit 'x' == False`

`isUpper :: Char -> Bool`
Return `True` if the argument is an uppercase letter
Examples: `isUpper 'x' == False` `isUpper 'X' == True`

`isLower :: Char -> Bool`
Return `True` if the argument is a lowercase letter
Examples: `isLower '3' == False` `isLower 'x' == True`

`toUpper :: Char -> Char`
If the argument is an alphabetic character, convert it to upper case
Examples: `toUpper 'x' == 'X'` `toUpper '3' == '3'`

`isPrefixOf :: String -> String -> Bool`
Return `True` if the first list argument is a prefix of the second
Examples: `isPrefixOf "has" "haskell" == True` `isPrefixOf "has" "handle" == False`

`isSuffixOf :: String -> String -> Bool`
Return `True` if the first list argument is a suffix of the second
Examples: `isSuffixOf "ell" "haskell" == True` `isSuffixOf "ell" "handle" == False`

`isInfixOf :: String -> String -> Bool`
Return `True` if the first list argument is contained in the second
Examples: `isInfixOf "ask" "haskell" == True` `isInfixOf "ask" "handle" == False`

`error :: String -> a`
Signal an error
Examples: `error "Cannot compute square root of a negative number"`