```
  1 {- This is the first assignment main code file for COM2108, started on October 4
th and to be
  2     sumbitted on October 28th, written by Maxime Fontana, every function descrip
tion is mentioned above it,
  3     everything related to testing for main functions can be found below those-}
  4 module Ciphers where
  5     import AssignmentHelp
  6     import Data.List -- needed for use of elemIndex
  7     import Data.Maybe -- needed for use of "fromJust"
  8     import Data.Tuple -- needed for use of "swap"
  9     import Data.Char -- needed for use of toLower/toUpper
 10     ------------------------------------------------------------------------
------------------
 11     -- (everything in relation with validateCipher)
 12
 13     -- Function from the slides that helps, output is a list which represents th
e number of times a character have been used in it
 14     isRepeated :: Cipher -> [Int]
 15     isRepeated my_cipher = map f my_cipher
 16                               where f x = length [ y | y <- my_cipher, x == y]
 17     -- We use the help function to return a string of characters, '1' if this is
 valid or '0' if it is not, valid means Upper Case Letters
 18     isaLetter :: Char -> Char
 19     isaLetter c = if alphaPos c >= 0 && alphaPos c <= 26 then '1' else '0'
 20
 21     -- Applying the function above to every character in a cipher, we check if t
hese are are all real Upper Case letters
 22     areAllValidLetters :: Cipher -> String
 23     areAllValidLetters xs = map isaLetter xs
 24
 25     -- Validating a Cipher first making sure there are 26 upper case letters tha
n making sure any is repeated
 26     validateCipher :: Cipher -> Bool
 27     validateCipher the_cipher = (areAllValidLetters the_cipher == "1111111111111
1111111111111" &&
 28           all (==1) (isRepeated the_cipher))
 29      {- I have tested this function by making sure every condition was met, i.e
testing without 26 letters,
 30      with lower case letters and 26 letters and repeated letters -}
 31
 32     ------------------------------------------------------------------------
------------------
 33     -- (everything in relation with encoding/reverseEncoding
 34
 35     -- adds the last element to the head of the list
 36     sliding :: Cipher -> String
 37     sliding xs = (last xs) : xs
 38
 39     -- removes the last element to create this right sliding effect
 40     adjusting :: Cipher -> String
 41     adjusting xs = init xs
 42
 43      {- encode a character, right sliding occurs on ciper as many times as the o
ffset is repeated by incrementation of 1,
 44     at the end returns the character in the cipher that matches the position of
the letter -}
 45     encode :: Cipher -> Int -> Char -> Char
 46     encode my_cipher offset my_char = if offset > 0 then encode (adjusting (slid
ing my_cipher)) (offset-1) (my_char) else my_cipher !! (alphaPos my_char)
 47     {- I have tested encode with ciphers of my own and ciphers provided in the a
ssignment, primarly, I had errors due to "Out of Bounds" cases in which
 48     I wanted to encode either A or Z, but it has been now resolved and have not
found any problem with it, the offset can take large numbers it will not fail
 49     so the wrapping part works fine. I have not included the validateCipher, so
in this this case as well as in the other functions below,
 50     I consider Ciphers are indeed Ciphers, but these can actually be just a stri
ng-}
 51
 52     -- applies "encode" to each element in the cipher, returning the full encode
d string
 53     encodeMessage :: Cipher -> Int -> String -> String
 54     encodeMessage my_cipher offset my_message =  map f my_message
 55                                                   where f x = encode my_cipher o
```

```
ffset x
 56     {- I have tested this function with different lengths of my_message and it w
orks perfectly for every letter we can have. However, it holds the same
 57     "validateCipher" limitation.-}
 58
 59     plain = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
 60     {- right sliding occurs on cipher as many times as the offset is repeated by
 incrementation of 1,
 61     at the end returns the character in the plain alphabet that matches the inde
x of the element in the cipher  -}
 62     reverseEncode :: Cipher -> Int -> Char -> Char
 63     reverseEncode my_cipher offset my_char = if offset > 0 then reverseEncode (a
djusting (sliding my_cipher)) (offset-1) (my_char) else plain !! (fromJust $ elemInd
ex my_char my_cipher)
 64     {- I have tested reverseEncode the same way I tested its twin function "enco
de" and this end up having the same limitation but no further errors have been notic
ed,
 65     the only part changing being the elem index and the created plain alphabet v
ariable, it has not brought any other problem -}
 66
 67     -- applies "reverseEncode" to each element in the cipher, returning the full
 string
 68     reverseEncodeMessage :: Cipher -> Int -> String -> String
 69     reverseEncodeMessage my_cipher offset my_message =  map f my_message
 70                                                           where f x = reverseEnco
de my_cipher offset x
 71      {-I have tested this function the same way I tested encodeMessage, no parti
cular noticed errors-}
 72
 73     ----------------------------------------------------------------------------
-------------------
 74     -- (everything in relation with letterStats)
 75
 76     {- A function that looks life isRepeated but where a character may be taken
into parameters
 77     and only returns the number of times this characters appears in the string -
}
 78     count :: Eq a => Integral b => a -> [a] -> b
 79     count e [] = 0
 80     count e (a:xs) = (count e xs +) $ if a == e then 1 else 0
 81
 82     -- applies count to every element in the list and compute the percentage for
 each letter without taking care if an element is repeated
 83     getFre :: String -> [Int]
 84     getFre xs = map f xs
 85                 where f x = ((count x xs) * 100) `div` length xs
 86
 87     -- a function that takes off all elements that are repeated therefore, in th
e output, every element appears once
 88     uniq :: Eq a => [a] -> [a]
 89     uniq [] = []
 90     uniq (x:xs) = (if x `elem` xs then id else (x:)) $ uniq xs
 91
 92     -- a function that sorts a list of tuples (by zipping getFre's output and an
d the input String) + using uniq to make sure we dont have duplicates
 93     preLetterStats :: String -> [(Int, Char)]
 94     preLetterStats xs = mergesort (>) (uniq (zip (getFre xs) xs))
 95
 96     -- same as preLetterStats but swapping elements within tuples
 97     letterStats :: String -> [(Char,Int)]
 98     letterStats xs = map swap (preLetterStats xs)
 99     {-I have tested this function with differents length of the message, from ma
de up relatively small length to mystery provided in the help file, it turned out
100     the function holds a small limitation which is, the computing part in getFre
 is not rounding up or down, so sometimes it may be possible to end up having
101     statistiques that don't add up to 100% but like 98%/99% due to this problem.
 For instance, with the example provided in the assignment handout with "STDDWSD"
102     the output add up to 98% due to rounding issue : [('D',42),('S',28),('W',14)
,('T',14)] -}
103
104
105     ----------------------------------------------------------------------------
-------------------
```

```
106     -- (last function)
107
108     -- Pulls out the first element of the tuples in the input list
109     getAllFst :: [(Char,Char)] -> String
110     getAllFst xs = map f xs
111                     where f x = snd x
112
113     {- If the character belongs to one of the first element of the tuples,
114      then we return the second element of that index in the reflector as a lower
 case otherwise we make sure we return the uppercase X -}
115     partialDecode :: [(Char,Char)] -> String -> String
116     partialDecode my_reflector my_message = map f my_message
117                                             where f x = if x `elem` (getAllFst
my_reflector) then toLower(fst (my_reflector !! (fromJust $ elemIndex x (getAllFst m
y_reflector)))) else toUpper x
118      {- I have tested this function using two examples, first the example given
in the handout : "DXPWXW" with the reflector [(¿E¿,¿X¿),(¿S¿,¿W¿)], which means it
119      correctly reads the reflector in the right order "E ciphers to X" etc... It
 gives the correct output. And now with reflectorB and mystery this gives the follow
ing
120      output : "eJAvjAbJFBEvjtjADBAJeJDJeFageVCvEVEFKedvdbFDCjKvjkFaMvaFDMCivAAjM
vAJFVkvJAijtvJCeAFavjkeJJkvBeJkFaMvEAJFVFteJACFDkdBvJCvEeMCJAFEJFTAecvaFfAJFVijbBvaF
Jkv
121      JAeagEvjAvJCvivAAjMvkvaMJCjBeJiFEvAJFVtvvVJCeAivAAjMvAvgEvJFEACjEveTbFDfjaJ
JCvfCFkvgkjAAJFMvJJCvBFaDAijEtAAJFV" which I have briefly reviewed by hand and I hav
e not
122      noticed anything goig wrong with the decoding here. -}
123
124
125
126
```

```
  1 Testing functions for student aca18mf
  2 Compiling to test part 1
  3 [1 of 3] Compiling AssignmentHelp   ( AssignmentHelp.hs, AssignmentHelp.o )
  4 [2 of 3] Compiling Ciphers          ( Ciphers.hs, Ciphers.o )
  5 [3 of 3] Compiling Main             ( test_assignment-1.hs, test_assignment-1.o
)
  6 Linking assignment_test ...
  7 Testing part 1
  8 (10) Testing validateCipher with various inputs...
  9 First six are valid ciphers
 10 True
 11 True
 12 True
 13 True
 14 True
 15 True
 16 Next four should be False
 17 "ABC" is not a valid cipher (does not contain the whole alphabet)
 18 False
 19 'A':rotor5 is not a valid cipher (contains an extra 'A')
 20 False
 21 "aBCDEFGHIJKLMNOPQRSTUVWXYZ" is not a valid cipher (not all uppercase)
 22 False
 23 "AAAAAAAAAAAAAAAAAAAAAAAAAA" is not a valid cipher (right number of characters,
but all duplicates)
 24 False
 25 Score on validateCipher: 10
 26 ------------------------------------------------------------
 27 Compiling to test part 2
 28 [1 of 3] Compiling AssignmentHelp   ( AssignmentHelp.hs, AssignmentHelp.o )
 29 [2 of 3] Compiling Ciphers          ( Ciphers.hs, Ciphers.o )
 30 [3 of 3] Compiling Main             ( test_assignment-2.hs, test_assignment-2.o
)
 31 Linking assignment_test ...
 32 Testing part 2
 33 (5) Testing encode with various inputs...
 34 The most simple case of all (no encoding):
 35  encode "ABCDEFGHIJKLMNOPQRSTUVWXYZ" 0 'M'
 36 'M'
 37 rot13: encode "ABCDEFGHIJKLMNOPQRSTUVWXYZ" 13 'A' (should give 'N')
 38 'N'
 39 Caesar cipher: encode "ABCDEFGHIJKLMNOPQRSTUVWXYZ" 3 'A' (should give 'D' or 'X'
)
 40 'X'
 41 encode rotor5 0 'A' (should give 'V')
 42 'V'
 43 encode rotor5 10 'A' (should give 'S' or 'A')
 44 'A'
 45 Score on encode: 5
 46 ------------------------------------------------------------
 47 Compiling to test part 3
 48 [1 of 3] Compiling AssignmentHelp   ( AssignmentHelp.hs, AssignmentHelp.o )
 49 [2 of 3] Compiling Ciphers          ( Ciphers.hs, Ciphers.o )
 50 [3 of 3] Compiling Main             ( test_assignment-3.hs, test_assignment-3.o
)
 51 Linking assignment_test ...
 52 Testing part 3
 53 (5) Testing encodeMessage with various inputs...
 54 encodeMessage "ABCDEFGHIJKLMNOPQRSTUVWXYZ" 0 "ASHORTTESTMESSAGE" (should be "ASH
ORTTESTMESSAGE")
 55 "ASHORTTESTMESSAGE"
 56 encodeMessage "ABCDEFGHIJKLMNOPQRSTUVWXYZ" 13 "ASHORTTESTMESSAGE" (should be "NF
UBEGGRFGZRFFNTR")
 57 "NFUBEGGRFGZRFFNTR"
 58 encodeMessage "ABCDEFGHIJKLMNOPQRSTUVWXYZ" 3 "ASHORTTESTMESSAGE" (should be "DVK
RUWWHVWPHVVDJH" or "XPELOQQBPQJBPPXDB")
 59 "XPELOQQBPQJBPPXDB"
 60 encodeMessage rotor4 0 "ASHORTTESTMESSAGE" (should be "ETAXFGGPTGRPTTEJP")
 61 "ETAXFGGPTGRPTTEJP"
 62 encodeMessage rotor3 10 "ASHORTTESTMESSAGE" (should be "XFWQDHHYFHUYFFXIY" or "I
RSJPTTKRTFKRRIUK")
 63 "IRSJPTTKRTFKRRIUK"
 64 Score on encodeMessage: 5
```

```
 65 ------------------------------------------------------------------
 66 Compiling to test part 4
 67 [1 of 3] Compiling AssignmentHelp   ( AssignmentHelp.hs, AssignmentHelp.o )
 68 [2 of 3] Compiling Ciphers          ( Ciphers.hs, Ciphers.o )
 69 [3 of 3] Compiling Main             ( test_assignment-4.hs, test_assignment-4.o
)
 70 Linking assignment_test ...
 71 Testing part 4
 72 (5) Testing reverseEncode with various inputs...
 73 reverseEncode "ABCDEFGHIJKLMNOPQRSTUVWXYZ" 0 'M' -- 'M'
 74 'M'
 75 reverseEncode "ABCDEFGHIJKLMNOPQRSTUVWXYZ" 13 'B' -- 'O' (rot13)
 76 'O'
 77 reverseEncode "ABCDEFGHIJKLMNOPQRSTUVWXYZ" 3 'D' -- 'A' or 'G' if shifted wrong
way (Caesar cipher (offset 3))
 78 'G'
 79 reverseEncode rotor5 0 'V' -- 'A'
 80 'A'
 81 reverseEncode rotor5 10 'S' -- 'A' (or 'U' if shifted wrong way)
 82 'U'
 83 Score on reverseEncode: 5
 84 ------------------------------------------------------------------
 85 Compiling to test part 5
 86 [1 of 3] Compiling AssignmentHelp   ( AssignmentHelp.hs, AssignmentHelp.o )
 87 [2 of 3] Compiling Ciphers          ( Ciphers.hs, Ciphers.o )
 88 [3 of 3] Compiling Main             ( test_assignment-5.hs, test_assignment-5.o
)
 89 Linking assignment_test ...
 90 Testing part 5
 91 (5) Testing reverseEncodeMessage to reverse the inputs provided from encodeMessa
ge.
 92     Partial marks awarded if reverseEncodeMessage works to decode a message from
 encodeMessage,
 93     even if reverseEncodeMessage is not completely correct
 94 reverseEncodeMessage "ABCDEFGHIJKLMNOPQRSTUVWXYZ" 0 "ASHORTTESTMESSAGE" -- (no e
ncoding)
 95 "ASHORTTESTMESSAGE"
 96 reverseEncodeMessage "ABCDEFGHIJKLMNOPQRSTUVWXYZ" 13 "NFUBEGGRFGZRFFNTR" -- (rot
13)
 97 "ASHORTTESTMESSAGE"
 98 reverseEncodeMessage "ABCDEFGHIJKLMNOPQRSTUVWXYZ" 3 "DVKRUWWHVWPHVVDJH"  (or "XP
ELOQQBPQJBPPXDB")
 99 "GYNUXZZKYZSKYYGMK"
100 "ASHORTTESTMESSAGE"
101 reverseEncodeMessage rotor4 0 "ETAXFGGPTGRPTTEJP"
102 "ASHORTTESTMESSAGE"
103 reverseEncodeMessage rotor3 10 "XFWQDHHYFHUYFFXIY" (or "IRSJPTTKRTFKRRIUK")
104 "UMBILNNYMNGYMMUAY"
105 "ASHORTTESTMESSAGE"
106 Score on reverseEncodeMessage: 5
107 ------------------------------------------------------------------
108 Compiling to test part 6
109 [1 of 3] Compiling AssignmentHelp   ( AssignmentHelp.hs, AssignmentHelp.o )
110 [2 of 3] Compiling Ciphers          ( Ciphers.hs, Ciphers.o )
111 [3 of 3] Compiling Main             ( test_assignment-6.hs, test_assignment-6.o
)
112 Linking assignment_test ...
113 Testing part 6
114 (20) Testing letterStats with various inputs... Rounding errors have been ignore
d.
115 letterStats "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
116   should give [('A',4),('B',4),('C',4),('D',4),('E',4),('F',4),('G',4),('H',4),(
'I',4),('J',4),('K',4),('L',4),('M',4),('N',4),('O',4),('P',4),('Q',4),('R',4),('S',
4),('T',4),('U',4),('V',4),('W',4),('X',4),('Y',4),('Z',4)] (order is not important)
117   2 marks
118 [('Z',3),('Y',3),('X',3),('W',3),('V',3),('U',3),('T',3),('S',3),('R',3),('Q',3)
,('P',3),('O',3),('N',3),('M',3),('L',3),('K',3),('J',3),('I',3),('H',3),('G',3),('F
',3),('E',3),('D',3),('C',3),('B',3),('A',3)]
119 letterStats (concat (map (\x -> take 26 (repeat x)) "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
))
120   should give same as previous
121   2 marks
122 [('Z',3),('Y',3),('X',3),('W',3),('V',3),('U',3),('T',3),('S',3),('R',3),('Q',3)
```

```
,('P',3),('O',3),('N',3),('M',3),('L',3),('K',3),('J',3),('I',3),('H',3),('G',3),('F
',3),('E',3),('D',3),('C',3),('B',3),('A',3)]
123 letterStats "A"
124   should give [('A',100)]
125   4 marks (-2 if don't remove 0-values)
126 [('A',100)]
127 letterStats "AB"
128   should give [('A',50),('B',50)]
129   3 marks (-1 if don't remove 0-values)
130 [('B',50),('A',50)]
131 letterStats "ABB"
132   should give i[('B',67),('A',33)]
133    4 marks (-2 if not sorted, -1 if zero-values not removed
134 [('B',66),('A',33)]
135 letterStats "STDDWSD"
136   should give [('D',43),('S',29),('W',14),('T',14)]
137            OR [('D',43),('S',29),('T',14),('W',14)]
138   5 marks (-2 if don't sort or remove 0-values)
139 [('D',42),('S',28),('W',14),('T',14)]
140 Score on letterStats: 20
141 ------------------------------------------------------------
142 Compiling to test part 7
143 Testing part 7
144 (10) Testing partialDecode with various inputs... (2 marks for each test)
145 First the sample from the specification partialDecode "DXPWXW" [('E', 'X'), ('S'
,'W')]
146 "DePses"
147 Now some guesses for the mystery message:
148 [('E','W'), ('T', 'J'), ('A', 'A'), ('O', 'F')]
149 should give: QtaeXaRtoBEeXZXaDBatQtDtQoYLQVCeEVEoKQHeHRoDCXKeXNoYMeYoDMCPeaaXMea
toVNetaPXZetCQaoYeXNQttNeBQtNoYMeEatoVoZQtaCoDNHBetCeEQMCtaoEtoTaQUeYoSatoVPXRBeYotN
etaQYLEeXaetCePeaaXMeNeYMtCXBQtPoEeatoVZeeVtCQaPeaaXMeaeLEetoEaCXEeQTRoDSXYttCeSCoNe
LNXaatoMettCeBoYDaPXEZaatoV
150 "QtaeXaRtoBEeXZXaDBatQtDtQoYLQVCeEVEoKQHeHRoDCXKeXNoYMeYoDMCPeaaXMeatoVNetaPXZet
CQaoYeXNQttNeBQtNoYMeEatoVoZQtaCoDNHBetCeEQMCtaoEtoTaQUeYoSatoVPXRBeYotNetaQYLEeXaet
CePeaaXMeNeYMtCXBQtPoEeatoVZeeVtCQaPeaaXMeaeLEetoEaCXEeQTRoDSXYttCeSCoNeLNXaatoMettC
eBoYDaPXEZaatoV"
151 [('E','W'), ('T', 'J'), ('S', 'A'), ('O', 'F'), ('P', 'V')]
152 should give: iQtseXsRtoBEeXZXsDBstQtDtQoYLQpCeEpEoKQHeHRoDCXKeXNoYMeYoDMCPessXMe
stopNetsPXZetCQsoYeXNQttNeBQtNoYMeEstopoZQtsCoDNHBetCeEQMCtsoEtoTsQUeYoSstopPXRBeYot
NetsQYLEeXsetCePessXMeNeYMtCXBQtPoEestopZeeptCQsPessXMeseLEetoEsCXEeQTRoDSXYttCeSCoN
eLNXsstoMettCeBoYDsPXEZsstop
153 "QtseXsRtoBEeXZXsDBstQtDtQoYLQpCeEpEoKQHeHRoDCXKeXNoYMeYoDMCPessXMestopNetsPXZet
CQsoYeXNQttNeBQtNoYMeEstopoZQtsCoDNHBetCeEQMCtsoEtoTsQUeYoSstopPXRBeYotNetsQYLEeXset
CePessXMeNeYMtCXBQtPoEestopZeeptCQsPessXMeseLEetoEsCXEeQTRoDSXYttCeSCoNeLNXsstoMettC
eBoYDsPXEZsstop"
154 [('E','W'), ('T', 'J'), ('S', 'A'), ('O', 'F'), ('P', 'V'), ('A', 'X'), ('Y', 'R
'), ('I', 'Q'), ('B', 'B'), ('K', 'Z'), ('R', 'E'), ('M', 'P'), ('N', 'Y'), ('C', 'L
'), ('H', 'C'), ('L', 'N'), ('G', 'M')]
155 should give: itseasytobreakasDbstitDtioncipherproKiHeHyoDhaKealongenoDghmessages
topletsmakethisonealittlebitlongerstopokitshoDlHbetherightsortoTsiUenoSstopmaybenotl
etsincreasethemessagelengthabitmorestopkeepthismessagesecretorshareiTyoDSanttheShole
classtogetthebonDsmarksstop
156 "itseasytobreakasDbstitDtioncipherproKiHeHyoDhaKealongenoDghmessagestopletsmaket
hisonealittlebitlongerstopokitshoDlHbetherightsortoTsiUenoSstopmaybenotletsincreaset
hemessagelengthabitmorestopkeepthismessagesecretorshareiTyoDSanttheSholeclasstogetth
ebonDsmarksstop"
157 [('E','W'), ('T', 'J'), ('S', 'A'), ('O', 'F'), ('P', 'V'), ('A', 'X'), ('Y', 'R
'), ('I', 'Q'), ('B', 'B'), ('K', 'Z'), ('R', 'E'), ('M', 'P'), ('N', 'Y'), ('C', 'L
'), ('H', 'C'), ('L', 'N'), ('G', 'M'), ('U', 'D'), ('D', 'H'), ('F', 'T'), ('V', 'K
'), ('Z', 'U'), ('W', 'S')]
158 should give: itseasytobreakasubstitutioncipherprovidedyouhavealongenoughmessages
topletsmakethisonealittlebitlongerstopokitshouldbetherightsortofsizenowstopmaybenotl
etsincreasethemessagelengthabitmorestopkeepthismessagesecretorshareifyouwantthewhole
classtogetthebonusmarksstop
159 "itseasytobreakasubstitutioncipherprovidedyouhavealongenoughmessagestopletsmaket
hisonealittlebitlongerstopokitshouldbetherightsortofsizenowstopmaybenotletsincreaset
hemessagelengthabitmorestopkeepthismessagesecretorshareifyouwantthewholeclasstogetth
ebonusmarksstop"
160 Score on partialDecode: 10
161 ------------------------------------------------------------
162
163 Total score for aca18mf is 60
```