```
-- PRG2 Blatt3
-- Marvin Glaser
-- 44242114
-- Gruppe 3 (Montag 12:00 bis 14:00 Uhr)
-- Martin Parnet


import Data.Char

-------------------------------------------------------------------------------
-- Aufgabe 1a)

-- calls help function for end-recursive computation of result
change_to_numbers :: [Char] -> Int
change_to_numbers string = sum (change_help string [])

-- guards check for possible characters and return stack when list
-- of characters is empty
change_help :: [Char] -> [Int] -> [Int]
change_help string stack
          | string == [] = stack
          | (head string) == '0' = change_help (tail string) (stack ++ [0])
          | (head string) == '1' = change_help (tail string) (stack ++ [1])
          | (head string) == '2' = change_help (tail string) (stack ++ [2])
          | (head string) == '3' = change_help (tail string) (stack ++ [3])
          | (head string) == '4' = change_help (tail string) (stack ++ [4])
          | (head string) == '5' = change_help (tail string) (stack ++ [5])
          | (head string) == '6' = change_help (tail string) (stack ++ [6])
          | (head string) == '7' = change_help (tail string) (stack ++ [7])
          | (head string) == '8' = change_help (tail string) (stack ++ [8])
          | (head string) == '9' = change_help (tail string) (stack ++ [9])
          | (head string) == 'a' = change_help (tail string) (stack ++ [1])
          | otherwise = change_help (tail string) (stack ++ [0])

{-
Testfaelle:
*Main> change_to_numbers "138aza"
14
*Main> change_to_numbers "aaaaaa"
6
*Main> change_to_numbers "bbbbbb"
0
*Main> change_to_numbers "1234567890"
45
-}

-------------------------------------------------------------------------------
-- Aufgabe 1b)

-- calls helper function to determine if single strings need to be all or no caps
change_captation string = change_help2 string [] where
        change_help2 string stack
                | string == [] = stack
                | (head (reverse (head string))) == '!' = change_help2 (tail string)
 (stack ++ (all_caps (head string) []))
                | otherwise = change_help2 (tail string) (stack ++ (all_lower (head
string) []))

-- generates new string in stack with all characters in uppercase
all_caps string stack
        | string == [] = [stack]
        | isLower (head string) = all_caps (tail string) (stack ++ [toUpper (head st
ring)])
        | otherwise = all_caps (tail string) (stack ++ [(head string)])

-- generates new string in stack with all characters in lowercase
all_lower string stack
        | string == [] = [stack]
        | isUpper (head string) = all_lower (tail string) (stack ++ [toLower (head s
```

```
tring)])
        | otherwise = all_lower (tail string) (stack ++ [(head string)])

{-
Testfaelle:
*Main> change_captation ["asdf", "asdf"]
["asdf","asdf"]
*Main> change_captation ["asdf!", "asdf!"]
["ASDF!","ASDF!"]
*Main> change_captation ["asdf!", "asd1234", "asdf!"]
["ASDF!","asd1234","ASDF!"]
*Main> change_captation []
[]
-}

--------------------------------------------------------------------------------
-- Aufgabe 1c)

-- calls helper function to ignore lists that are smaller than 4 elements and then
-- call second help function to generate list of last three numbers of list elements
merge_lists list = merge_help list [] where
        merge_help list stack
                | list == [] = stack
                | ((length (head list)) >= 4) = merge_help (tail list) (stack ++ (sw
itch_add (head list) []))
                | otherwise = merge_help (tail list) stack

-- combine last three elements of list to stack and return
switch_add list stack = [head (reverse list)] ++ [head (tail (reverse list))] ++ [he
ad (tail (tail (reverse list)))]

{-
Testfaelle:
*Main> merge_lists [[1,2], [1..4], [6..8], [8..12]]
[4,3,2,12,11,10]
*Main> merge_lists [[1], [2], [3], [4]]
[]
*Main> merge_lists [[1..10000]]
[10000,9999,9998]
-}

--------------------------------------------------------------------------------
-- Aufgabe 2a)

-- matches for list without elements, for last element and for everything else
switch [] = []
switch (x : []) = x : []
switch (x : (xs : xxs)) = (xs : x : (switch xxs))

{-
Testfaelle:
*Main> switch [1..9]
[2,1,4,3,6,5,8,7,9]
*Main> switch []
[]
*Main> switch [1, 1, 1, 1, 1, 1, 1]
[1,1,1,1,1,1,1]
-}

--------------------------------------------------------------------------------
-- Aufgabe 2b)

-- checks outer layer of list for end and calls adder2
adder [] = 0
adder ((x : xs) : []) = adder2 (x : xs)
adder ((x : xs) : xxs) = adder2 (x : xs) + adder xxs

-- checks first inner layer of list for end and calls adder3
adder2 [] = 0
```

```
adder2 ((x : xs) : []) = adder3 (x : xs)
adder2 ((x : xs) : xxs) = adder3 (x : xs) + adder2 xxs

-- checks list of integers and returns first integer or 0
adder3 [] = 0
adder3 (x : xs) = x

{-
*Main> adder [[[1,2], [4,5], [7,8,9]], [[13,15], [19]]]
44
*Main> adder [[[1], [4], [7]], [[15], [19]]]
46
-}

--------------------------------------------------------------------------------
-- Aufgabe 2c)

-- calls for helper function that calls that generates and adds stacks of character
-- lists depending on the index and returns stack at end of string
lister [] = []
lister list = lister_help list [] 1 where
    lister_help (x : xs) output count = if xs == []
        then (output ++ (add_char x [] count 0))
        else lister_help xs (output ++ (add_char x [] count 0)) (count + 1)

-- generates string of same character depending on number of index
add_char char return counter1 counter2 =
    if counter1 == counter2
        then return
        else add_char char (char : return) counter1 (counter2 + 1)

{-
*Main> lister ['a', 'b', 'c']
"abbccc"
*Main> lister []
[]
*Main> lister ['a', 'a', 'a']
"aaaaaa"
*Main> lister ['1', '2', '3']
"122333"
-}
```