Input and Ouput Exam Solutions

1 2021 Q3

1.1 Part (b)

```
main :: IO ()
main = do
 emptyOutputFile -- reset SHOUT.log since we want a new file called SHOUT.log
 file1Contents <- readFile "files.txt"
 let listFiles = lines file1Contents
 if listFiles == []
     then return ()
     else do
         shoutIntoFile listFiles
 where emptyOutputFile = writeFile "SHOUT.log" ""

shoutIntoFile :: [FilePath] -> IO ()
shoutIntoFile [] = return ()
shoutIntoFile (xs:xss) = do
 fileContent <- readFile xs
 appendFile "SHOUT.log" $ map (toUpper) fileContent
 shoutIntoFile xss
```

2 2021 Q3

2.1 Part (c)

```
main :: IO ()
main = do
 interleaves "input1.txt" "input2.txt"

interleaves :: FilePath -> FilePath -> IO ()
interleaves file1 file2 = do
 file1Contents <- readFile file1
 file2Contents <- readFile file2
 let linesOfFile1 = lines file1Contents
 let linesOfFile2 = lines file2Contents
 writeFile "output12.txt" . unlines $ interleaves' [] linesOfFile1 linesOfFile2

interleaves' :: [String] -> [String] -> [String] -> [String]
interleaves' zss xss [] = zss ++ xss
interleaves' zss [] yss = zss ++ yss
interleaves' zss (xs:xss) (ys:yss) = (interleaves' $! accumulator) xss yss
 where accumulator = zss ++ (xs : [ys])
```

- A efficient-ish implementation using intercalate, strict evaluation.

```
import Data.List (intercalate)

main :: IO ()
main = do
  file1Contents <- readFile "input1.txt"
  file2Contents <- readFile "input2.txt"
  let linesOfFile1 = lines file1Contents
  let linesOfFile2 = lines file2Contents
  writeFile "output12.txt" $ (interleaveLines linesOfFile1 linesOfFile2 ++ "\n")

interleaveLines :: [String] -> [String] -> String
interleaveLines [] ys = intercalate "\n" ys
interleaveLines xs [] = intercalate "\n" xs
interleaveLines (x:xs) (y:ys) = intercalate "\n" $! x : y : [interleaveLines xs ys]
```

- Most efficient solution and shortest code wise, using stritctiness (seq), mapM

```
main = do
  [f1,f2] <- mapM readFile ["input1.txt", "input2.txt"]
  writeFile "output12.txt" $ unlines $ interleave (lines f1) (lines f2)
interleave (x:xs) (y:ys) = x `seq` y `seq` (x : y : interleave xs ys)
interleave _ _ = []
```

3 2019 Q3

3.1 Part (c)

```
main :: IO ()
main = do
  putStr "Input a file with the form <root.ext>: "
  file <- getLine
  fileContents <- readFile file
  writeFile (outputFile file) $ map (toLower) fileContents
    where outputFile file = takeWhile (/='.') file ++ ".log"
```

4 2018 Q3

4.1 Part (c)

```
toDOS :: FilePath -> FilePath
toDOS file = map (toUpper) (take 8 fileName) ++ map (toUpper) (take 4 fileExtension)
 where fileName      = takeWhile (/= '.') file
       fileExtension = dropWhile (/= '.') file -- includes dot therefore take 4 == .DAT
```

4.2 Part (d)

```
main :: IO ()
main = do
 putStr "Input a fileName with an extension: "
 file <- getLine
 fileContents <- readFile . toDOS $ file
 writeFile "LOWER.OUT" $ map (toLower) fileContents
```

5 2015 Q4

5.1 Part (d)

```
hash :: String -> Int
hash str = (sum (map ord str)) `mod` 255

main :: IO ()
main = do
 putStr "Input the name of your file without an extension: "
 fileName <- getLine
 fileContents <- readFile . inputFile $ fileName
 writeFile (outputFile fileName) (show . hash $ fileContents)
  where inputFile fileName  = fileName ++ ".in"
        outputFile fileName = fileName ++ ".chk"
```

6 2014 Q4

6.1 Part (d)

```
main = do
 putStr "Input a file in of the form <Root>.<Extensions>: "
 file <- getLine
 let dosFile = toDOS file
 dosFileContents <- readFile dosFile
 writeFile ((take 8 dosFile) ++ ".OUT") $ map (toLower) dosFileContents

toDOS :: FilePath -> FilePath
toDOS file = map (toUpper) dosNam ++ map (toUpper) dosExt
 where dosNam = take 8 $ takeWhile (/= '.') file
       dosExt = take 4 $ dropWhile (/= '.') file -- take 4 since . must be included
```

7 2013 Q4

7.1 Part (d)

```
main = do
 putStr "Input a filename without the extension: "
 file <- getLine
 fileContents <- readFile (file++".in")
 writeFile (file++".out") $ map (toUpper) fileContents
```