Introduction to Functional Programming Final Examination

Question 1

(a) Write a higher-order function hof that captures this common behaviour

Here is a higher order function hof that captures the common behaviour.

```
hof            :: (a -> b -> a) -> a -> [b] -> a
hof f z []     =  z
hof f z (x:xs) =  hof f (f z x) xs
```

(b) Rewrite each of f1, f2, f3, f4, and f5 above, to be a call to hof with appropriate arguments.

Here is f1,f2,f3,f4 and f5 rewritten to call hof.

```
f1 = hof (*)
f2 = hof (||)
f3 = hof (\x y -> 2*x+y)
f4 = hof (++)
f5 = hof (-)
```

(c)

Yes, it is foldl. It , applied to a binary operator, a starting value (typically the left-identity of the operator), and a list, reduces the list using the binary operator, from left to right.

Question 2

(a) Describe, using an example, one way in which a call to function beval can fail, with a Haskell runtime error that is not due to pattern-matching.

Division by zero will result in a runtime error. An Example would be calling beval in the form of beval _ (B True). We would end up calculating (3 ‘div‘ 0), which would raise an exception at run time. This is not due to pattern matching. The result would be “*** Exception: divide by zero”. This is not equal to 0.

(b) Describe, using examples, two examples of different calls to function beval that fail, with Haskell runtime pattern-matching errors. Note: such pattern errors may occur in other functions called by beval.

```
beval d (B False)      -- Only True is accepted, should accept B False.
beval d (Not p1)       -- Only And is accepted, should also accept Not.
```

(c) Rewrite the beval function above, by adding in any missing code, correcting any code giving wrong answers, and adding proper error handling, with informative error messages, using Monads. You will need to change the type of the function, so your answer should give the revised type of the function.

```
data Prop = B Bool
          | P String
          | Not Prop
          | And Prop Prop
          | Let String Prop Prop
          deriving Show

type Dict = [(String,Bool)]
ins :: String -> Bool -> Dict -> Dict
ins str int ds = (str, int) : ds
lkp :: (Monad m, MonadFail m) => String -> Dict -> m Bool
lkp _ [] = fail "Not in Dictionary"
lkp name ((s, v) : ds)
                  | name == s = return v
                  | otherwise = lkp name ds

beval :: MonadFail m => Dict -> Prop -> m Bool
beval d (And p1 p2) = do
      i <- beval d p1
      j <- beval d p2
      return (i && j)
beval d (Not p1) = do
      i <- beval d p1
      return (not i)
beval d (P s) = lkp s d
beval d (Let v p1 p2) = do
       b <- beval d p1
       beval (ins v b d) p2
beval _ (B True) = return False
beval _ (B False) = return True
```

Question 3

(a)

```
THEORY Simple

IMPORT-HASKELL List

THEOREM theorylength

       len [x] == 1

STRATEGY ReduceLHS

len [x]
 = DEF len.2
1 + len([])
 = DEF len.1
1 + 0
 = SIMP
1

QED theorylength
```

Where List.hs is:

```
module List where
len [] = 0
len (x:xs) = 1 + len xs
rev [] = []
rev (x:xs) = rev xs ++ [x]
```

(b)

```
BASE CASE

P([])
= ”expand P”
length ([]++ys) = length [] + length ys
= ”Defs of ++ and length”
length ys = 0 + length ys
= ”arithmetic”
length ys = length ys
= ”reflexivity of =”
True
```

```
INDUCTIVE STEP

Assume P(xs), i.e. len (rev xs) = len xs

Show P(x:xs):

P(x:xs)
= ”expand P”
len (rev (x:xs) ys) = len (x:xs) + len ys
= ”Defs of rev and len”
len (x:(rev xs ys)) = (1 + len xs) + len ys
= ”Defs of len, + is assoc”
1 + len (rev xs ys) = 1 + (len xs + len ys)
= ”arithmetic”
len (rev xs ys) = len xs + len ys
= ”by ind. hypothesis”
True
```

Where List.hs is:

```
module List where
len [] = 0
len (x:xs) = 1 + len xs
len (xs++ys) = len xs + len ys
rev [] = []
rev (x:xs) = rev xs ++ [x]
```

(c)

```
interleaves :: IO ()
interleaves  = do
      -- read the log file
      file1 <- readFile "input1.txt"
      file2 <- readFile "input2.txt"
      -- get all file names line by line via function "lines"
      let lines1 = lines file1
      let lines2 = lines file2
      writeFile "output12.txt" concat ([lines1][lines2])
```