Assignment 8 solutions

```haskell
import qualified Data.Set as S
import qualified Data.Map as M
import Control.Monad (void)
import Data.Time.Calendar (Day, fromGregorian)
```

Folds

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

g :: Int -> Int -> Int
g x y = x + y*y

foldList g 1 [2..4] =
foldList g 1 (2:3:4:[]) =
g 2 (foldList g 1 (3:4:[])) =
g 2 (g 3 (foldList g 1 (4:[]))) =
g 2 (g 3 (g 4 (foldList g 1 []))) =
g 2 (g 3 (g 4 1)) =
g 2 (g 3 (4 + 1*1)) =
g 2 (g 3 5) =
g 2 (3 + 5*5) =
g 2 28 =
2 + 28*28 =
786

snoc :: Char -> String -> String
snoc c s = s ++ [c]

foldList snoc "z" "arbe" =
foldList snoc "z" ('a':'r':'b':'e':[]) =
snoc 'a' (foldList snoc "z" ('r':'b':'e':[])) =
snoc 'a' (snoc 'r' (foldList snoc "z" ('b':'e':[]))) =
snoc 'a' (snoc 'r' (snoc 'b' (foldList snoc "z" ('e':[])))) =
snoc 'a' (snoc 'r' (snoc 'b' (snoc 'e' (foldList snoc "z" [])))) =
snoc 'a' (snoc 'r' (snoc 'b' (snoc 'e' "z"))) =
snoc 'a' (snoc 'r' (snoc 'b' ("z" ++ ['e']))) =
snoc 'a' (snoc 'r' (snoc 'b' "ze")) =
```

```
snoc 'a' (snoc 'r' ("ze" ++ ['b'])) =
snoc 'a' (snoc 'r' "zeb") =
snoc 'a' ("zeb" ++ ['r']) =
snoc 'a' "zebr" =
("zebr" ++ ['a']) =
"zebra"
```

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

Maps and joins

```haskell
type UserId = Int
data Name = Name { firstName, lastName :: String }
  deriving Eq

instance Show Name where
  show n = firstName n ++ " " ++ lastName n

people :: M.Map UserId Name
people = M.fromList
  [ (13, Name "Alice"   "Arcila")
  , (21, Name "Ben"     "Berman")
  , (40, Name "Carol"   "Cornwall")
  , (58, Name "Panda"   "Patel")
  , (71, Name "Randall" "Rivera")
  ]

birthdays :: M.Map UserId Day
birthdays = M.fromList
  [ (21, fromGregorian 1984 10 15)
  , (40, fromGregorian 1990  3 24)
  , (71, fromGregorian 1979 12  4)
  ]

type TwitterId = String

followers :: M.Map UserId TwitterId
followers = M.fromList
  [ (11, "@anndavis")
  , (13, "@aliccea")
  , (21, "@bb")
```

```
  , (31, "@liveness")
  , (58, "@trailblazer")
  ]

printMap :: (Show k, Show a) => M.Map k a -> IO ()
printMap = void . M.traverseWithKey (\k a ->
  putStrLn $ show k ++ ": " ++ show a)

innerJoin :: Ord k => M.Map k a -> M.Map k b -> M.Map k (a,b)
innerJoin m1 m2 = M.intersectionWith (,) m1 m2

leftJoin :: Ord k => M.Map k a -> M.Map k b -> M.Map k (a, Maybe b)
leftJoin m1 m2 = M.mapWithKey f m1
  where f k a = (a, M.lookup k m2)

rightJoin :: Ord k => M.Map k a -> M.Map k b -> M.Map k (Maybe a, b)
rightJoin m1 m2 = M.mapWithKey f m2
  where f k a = (M.lookup k m1, a)

fullJoin :: Ord k => M.Map k a -> M.Map k b -> M.Map k (Maybe a, Maybe b)
fullJoin m1 m2 = S.foldr f M.empty $ S.union (M.keysSet m1) (M.keysSet m2)
  where f k m = M.insert k (M.lookup k m1, M.lookup k m2) m

main = return ()
```