HW 2 Solution: More Haskell

CS 440: Programming Languages and Translators
Was due Sat Feb 9

1. (Polymorphic types of polymorphic functions)
   \[ f_{1a} :: (b, a) \rightarrow (a, b) \]
   \[ f_{1a} = \lambda(x, y) \rightarrow (y, x) \]
   \[ f_{1b} :: a \rightarrow [a] \rightarrow [[a]] \]
   \[ f_{1b} = \lambda x \ y \rightarrow [[x], \ y] \]
   \[ f_{1c} :: a \rightarrow a \rightarrow [a] \rightarrow [[a]] \]
   \[ f_{1c} = \lambda x \ y \ z \rightarrow [x : z, \ y : z] \]
   \[ f_{1d} :: (a \rightarrow \text{Bool}) \rightarrow [a] \rightarrow \text{Int} \]
   \[ f_{1d} f = \text{length} \ . \ (\text{filter} \ f) \]

2. (Iterate via standard recursion)
   \[ \text{iterate1} \ n \ f \]
   \[ | \ n \leq 0 = \text{id} \]
   \[ | \ otherwise = f \ . \ (\text{iterate1} \ (n-1) \ f) \]

3. (Iterate via folding a list of functions)
   \[ \text{iterate2} \ n \ f = \text{foldl} \ (\ . \) \ \text{id} \ [f \ | \ i \ <- \ [1..n]] \]

For Problems 4 – 7, we used
   \[ \text{data Tree} \ a = \text{Leaf} \ a \ | \ \text{Branch} \ a \ (\text{Tree} \ a) \ (\text{Tree} \ a) \]
   \[ \text{deriving} \ (\text{Eq}, \ \text{Show}) \]

4. (Our own tree equality)
   \[ \text{treeEq} :: (\text{Eq} \ a) \Rightarrow \text{Tree} \ a \rightarrow \text{Tree} \ a \rightarrow \text{Bool} \]
   \[ \text{treeEq} \ (\text{Leaf} \ x) \ (\text{Leaf} \ y) = x == y \]
   \[ \text{treeEq} \ (\text{Branch} \ x1 \ l1 \ r1) \ (\text{Branch} \ x2 \ l2 \ r2) \]
   \[ = x1 == x2 \ &\& \ \text{treeEq} \ l1 \ l2 \ &\& \ \text{treeEq} \ r1 \ r2 \]
   \[ \text{treeEq} \ _ \ _ = \text{False} \]
5. (Our own tree show)

```haskell
  treeShow :: Show a => Tree a -> [Char]
  treeShow (Leaf x) = "(Leaf " ++ show x ++ ")"
  treeShow (Branch x left right)
    = "(Branch " ++ show x ++ " "
      ++ treeShow left ++ " "
      ++ treeShow right ++ ")"
```

6. (Preorder traversal, standard recursion)

```haskell
  preorder :: Tree a -> [a]
  preorder (Leaf x) = [x]
  preorder (Branch x left right)
    = x : preorder left ++ preorder right
```

7. (Tail-recursive preorder traversal)

```haskell
  preorder' :: Tree a -> [a] -> [a]
  preorder' (Leaf x) xs = x : xs
  preorder' (Branch r left right) xs
    = r : preorder' left (preorder' right xs)

  preorder2 :: Tree a -> [a]
  preorder2 tree = preorder' tree []
```

8. (Pattern matching)

```haskell
  data Pattern a = P a | POr (Pattern a) (Pattern a) | PAnd (Pattern a) (Pattern a) deriving Show

  match pattern [] = (False, [])
  match (P x) (y : ys) = if x == y then (True, ys) else (False, y : ys)
  match (POr pat1 pat2) xs =
    case match pat1 xs of
      (True, leftover) -> (True, leftover) -- stop if pat1 succeeded
      (False, _) -> match pat2 xs -- else try pat2
  match (PAnd pat1 pat2) xs =
    case match pat1 xs of
      (False, _) -> (False, xs)
      (True, leftover) ->
        case match pat2 leftover of
          (False, _) -> (False, xs)
          (True, leftover2) -> (True, leftover2)
```