HW 9 **Prof. Caldwell**
Due: 5 October 2011 **COSC 3015**

1 Trees, maps, and folds

In class we built some code for a tree type:

```
module MyTree where

data Tree a = Leaf | Node a (Tree a) (Tree a) deriving Show

-- mktree turns a list into a realtively balanced tree
mktree :: [a] -> Tree a
mktree [] = Leaf
mktree (x:xs) = Node x (mktree left) (mktree right)
  where (left,right) = splitAt (length xs `div` 2) xs

-- flatten turns a tree into a list (inverse of mktree)
flatten :: Tree a -> [a]
flatten Leaf = []
flatten (Node x left right) = x : (flatten left ++ flatten right)

-- mapT maps a function over a tree
mapT :: (a -> b) -> Tree a -> Tree b
mapT f Leaf = Leaf
mapT f (Node x left right) = Node (f x) (mapT f left) (mapT f right)

-- foldT collapses a tree
foldT :: ( a -> b -> b -> b) -> b -> Tree a -> b
foldT f id Leaf = id
foldT f id (Node x left right) = f x (foldT f id left) (foldT f id right)
```

Exercise 1.1. Implement `mapT` using `foldT`.

Exercise 1.2. Can you implement `foldT` using `mapT` - why or why not?

Exercise 1.3. Here's a type of trees that can have none, one or two childen.

```
data OneTwoTree a =
   Leaf |
   Node1 a (OneTwoTree a) |
   Node2 a (OneTwoTree a) (OneTwoTree a) deriving Show
```

Create a module called `OneTwoTree` and add code to implement map and fold functions for this new type. Note that since there are three constructors for the type `OneTwoTree` there will be three arguments to the fold. I called my versions `mapOTT` and `foldOTT` and they have the following types:

```
mapOTT :: (a -> b) -> OneTwoTree a -> OneTwoTree b
foldOTT :: b -> (a -> b -> b) -> (a -> b -> b -> b) -> OneTwoTree a -> b
```

As always - run some tests on your code to convince the grader that your code works.