HW 8 **Prof. Caldwell**
Due: 21 September 2010 **COSC 3015**

Recall the function `last` which was defined by recursion on the structure of the list.

```
last [] = error "last: nil."
last (x:xs) = if null xs then x else last xs
```

You need to implement the following functions by recursion on the structure of at least one of the list arguments:

Problem 0.1. `ismem :: (Eq a) => a -> [a] -> Bool`

A call to the function `ismem` of the form `(ismem x ys)` returns `True` if `x` is in the list `ys` and returns `False` otherwise.

Problem 0.2. `allaremem :: (Eq a) => [a] -> [a] -> Bool`

A call to `allaremem` of the form `(allaremem xs ys)` returns `True` if all the elements in the list `xs` occur in the list `ys` and returns `False` otherwise. In analogy with subsets, you might think of this as a sublist function, it returns true if the first list is a sublist of the second (disregarding order.) You may find your implementation of `ismem` useful in defining `allaremem`.

Problem 0.3. `find :: (Eq a) => a -> [(a, b)] -> b`

A call of the form `(find x xys)` returns `y` (where `y` has type `b`) if there is a pair of the form `(x,y)` in the list `xys` and raises an exception otherwise. Here's a big hint on `find`, the left side of the definition of find could appear as follows:

```
find x [] =  ...
find x ((y,z): yzs) = ...
```

Setting up the left side of the definition like this allows for pattern matching on the structure of the pairs in the list argument.