HW 12 **Prof. Caldwell**
Due: 5 October 2011 **COSC 3015**

Exercise 0.1. Read sections 4.2, 4.3, 4.4 and 4.5 of Bird (pp. 95—128)

Exercise 0.2. Write a recursive function to implement `ordered1` which tells whether a list is in strictly increasing order or not.

```
ordered1 :: (Ord a) => [a] -> Bool
```

Write enough test cases to convince the grader that your code is correct.

Exercise 0.3. There is a function `f` such that the following definition behaves as the predicate `ordered1` defined above. Fill in the "???".

```
pairs [] = []
pairs m = zip m (tail m)

ordered2 m = foldr f True (pairs m)
  where f (x,y) z = ???
```

To solve this problem you'll probably have to experiment with the function `pairs` to see what it does.

Write enough test cases to convince the grader that your code is correct.

Exercise 0.4. Implement the function `filter1` which has the same type as, and behaves just like `filter` except that it is defined using `foldr`. Write enough test cases to convince the grader that your code is correct.

Exercise 0.5. Write Haskell code to implement `partition` using the `foldr` function.

```
partition :: (a -> Bool) -> [a] -> ([a],[a])
```

The specification[1] for `partition` is as follows:

```
partition p l = (filter p l, filter (not . p) l)
```

Recall that `(f . g)` denotes function composition, *i.e.* `(f . g)(x) = f(g(x))`.

Write enough test cases to convince the grader that your code is correct. You can use the specification to test the output of your version for correctness.

[1] This means that this equation characterizes the `partition` function, *i.e.* for any implementation of partition, this property must hold.