**HW 10** **Prof. Caldwell**
**Due:** 5 October 2006 **COSC 3015**

# 1 Recursion on Lists

**Exercise 1.1.** Read sections 4.1, 4.2, 4.3, and 4.4 of Bird (pp. 91–120).

In class we presented the following functions defined by recursion on the structure of a list.

```
len :: [a] -> Integer
len [] = 0
len (h:t) = 1 + len t

append :: [a] -> [a] -> [a]
append [] l = l
append (h:t) l = h:(append t l)

rev :: [a] -> [a]
rev [] = []
rev (h:t) =  (rev t ) ++  [h]

rev' :: [a]  -> [a]
rev' l = trev l []
   where
     trev [] m = m
     trev (h:t) m = trev t (h:m)
```

Note that `len` is built into the Haskell prelude as `length`, `append` is built in as the infix operator `(++)` and `rev'` is built in under the name `reverse`.

*For the following exercises, write at least a dozen test cases for each function and send in the script file containing your definitions together with the interaction with Hugs showing the results of your test runs.*

**Exercise 1.2.** Write a function `mem` having type

```
mem ::  Eq a => a -> [a] -> Bool
```

such that `(mem x m)` evaluates to `True` if `x` is in the list `m` and evaluates to `False` otherwise.

**Exercise 1.3.** Write a function `pmem` having type

```
pmem ::  (a -> Bool) -> [a] -> Bool
```

such that `(pmem p m)` evaluates to `True` if the function `p ::  a -> Bool` evaluates to `True` on any element in the list and returns `False` otherwise.

Note that you should be able to implement `(mem x m)` as `(pmem (\y -> y == x) m)`.

**Exercise 1.4.** Write a function `remove` having type

```
remove ::  Eq a => a -> [a] -> [a]
```

that removes the first `x` in `m` (if it is there) and returns the resulting list.

For example, our function should have the following behavior.

```
remove 5 [1,2,3] = [1,2,3]
remove 5 [] = []
remove 5 [5] = []
remove 5 [5,1,3,5] = [1,3,5]
remove 5 [5,5] = [5]
```

**Exercise 1.5.** Write a function `remove_all` having type

```
remove_all ::  Eq a => a -> [a] -> [a]
```

that removes every `x` in the list `m` (if it is there) and returns the resulting list. For example, our function should have the following behavior.

```
remove_all 5 [1,2,3] = [1,2,3]
remove_all 5 [] = []
remove_all 5 [5] = []
remove_all 5 [5,1,3,5] = [1,3]
remove_all 5 [5,5] = []
```