HW 17 **Prof. Caldwell**
Due: 7 November 2006 **COSC 3015**

Exercise 0.1. Read pages 185–202 of chapter 6.
Consider the inorder traversal.
inorder:

i.) visit the left subtree, then

ii.) visit the root, then

iii.) visit the right subtree.

This traversal is essentially stack-based, although no stack appears in the code, the order of recursive calls simulates the a stack. Breadth first traversals are queue-based. Instead of pushing the unexplored parts of the tree onto a stack, we push them onto a queue.
Given a tree `t` to be traversed pseudo-code for the algorithm is given as follows:
breadth first:

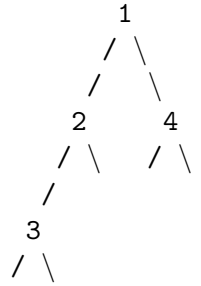
```
breadth first (t: Btree) =
  q = push t (mkQueue());
  ans = [];
  while (not empty q) do
      if (top q == Nil) then
          q:= pop q
      elseif (top q == Node x t1 t2) then
          q := push t2 (push t1 (pop q));
          ans := ans ++ [x]
      else
          skip
  end
  return ans
```

So, for example if the type `Btree` is defined as:

```
data Btree a = Nil | Node a (Btree a) (Btree a) deriving (Eq,Ord,Show)
```

Then, given a tree of the following form:

```
Node 1 (Node 2 (Node 3 Nil Nil) Nil) (Node 4 Nil Nil)
```



The breadth first traversal results in the list `[1,2,4,3]`.

The following code implements a queue.

```
type Queue a = ([a],[a])

mkQueue () = ([],[])

push x (m,n) = (x:m,n)

top ([],[]) = error "head: empty queue."
top (m,h:t) = h
top (m,[]) = top ([],reverse m)

pop ([],[]) = error "pop: empty queue."
pop (m, h:t) = (m,t)
pop (m,[]) = pop ([], reverse m)

emptyq ([],[]) = True
emptyq (m,n) = False
```

Exercise 0.2. Write Haskell code to do a breadth first traversal of a `BTree`.