G6021: Comparative Programming

Exercise Sheet 3

1 Functions on lists

1. Write a function `addOneToAll` that will add one to all the elements of a list. Write this two ways, first using pattern matching, then using list comprehensions.

2. Write a function `timesTwoToAll` that will multiply all the elements of a list by two. Write this function using pattern matching on lists, and compare your answer with that of the previous question.

3. Haskell provides a useful function called map:

   ```
   map :: (a -> b) -> [a] -> [b]
   ```

   This function takes a function `a -> b` and a list `[a]` and returns a list `[b]`.

   Write functions `addOne` and `timesTwo`, and use these as arguments to `map` to generate the same answers as the previous two questions.

4. Write your own version of `map`.

5. Haskell also provides a way of creating functions for one-off use. For example:

   ```
   map (\x -> x*2) [1,2,3]
   ```

   This saves us having to write the `timesTwo` function first. Think of the notation `\x -> x*2` as meaning a function that takes an argument `x` and returns `x*2`.

   Try to understand the following examples, testing them to confirm your intuitions:

   (a) `map (\x -> if x==0 then False else True) [0,1,0,1]`

   (b) `map (\(x,y) -> x*y) [(1,3),(4,5),(1,9)]`

   (c) `map (\(x,y) -> (y,x)) [(1,3),(4,5),(1,9)]`

   We remark that other patterns can also be used in these functions.

6. Using the notation of the previous question, use map to write the following:

   (a) A function that converts a list of Booleans (True, False) to a List of binary numbers. Example: `[True,False,True,True]` should be converted to `[1,0,1,1]`.

   (b) A function that converts a list of lists to a list containing just the head of each list. Example: `[[1,2,3],[9,3,4],[7,4,1]]` should give `[1,9,7]`.

7. The infix function `++` concatenates (or appends) two lists, for example:

   `[1,2,3] ++ [4,5,6] = [1,2,3,4,5,6]`

   Write your own version, called `append`, of this function. How many recursive calls are needed to append two lists?

8. Quick Sort. If a list is empty, then it is already sorted. Otherwise, we can sort a list by:

   - Selecting an element, called the *pivot*, from the list (for example the first element).
   - Collecting and sorting all the elements less than or equal to the pivot.
   - Collecting and sorting all the elements greater than the pivot.
   - Joining the above two lists with the pivot in the middle.

   Write this function using Haskell (you may find the use of list comprehensions useful).

2 Trees

Here is a data type for binary trees:

```
data BinaryTree a = EmptyTree | Node a  (BinaryTree a) (BinaryTree a)
```

Add this definition to your file. If you try to create a tree, for example: `Leaf 6`, you will see that Haskell does not know how to print it. What we need to do is to provide details how we would like expressions of type `Tree` to be printed. One easy way is to ask Haskell to simply derive automatically a function, and this is done by:

```
data BinaryTree a = EmptyTree | Node a  (BinaryTree a) (BinaryTree a)  deriving (Show)
```

Try again some examples:

- `EmptyTree`
- `Node 3 (Node 4 EmptyTree EmptyTree) EmptyTree`

1. Write a function to convert a binary tree to a list. The elements should be generated *in-order*: for a tree `Node v l r`, the list should contain all the elements of `l` in-order, followed by the element `v`, followed by all the elements of `r` in-order.

   Example: `toList (Node 3 (Node 4 EmptyTree EmptyTree) EmptyTree) = [4,3]`

   (You may find it useful to draw the binary tree.)

2. We will say that binary tree is *sorted* if the list generated in-order from the tree is a sorted list. Using only the functions you have defined so far, and function composition, write a function that will test if a Tree of integers is sorted in ascending order.

3. Write a function `insertTree` that takes an integer `x` and a sorted tree, and returns a tree with `x` inserted in the correct place.

3 Additional questions (if you have time)

1. Write two functions `preOrderTree`, `postOrderTree` that generate lists from trees using *pre-order* and *post-order* respectively.

   - For a binary tree `(Node v l r)` *pre-order* means a list with `v`, then all the elements `l` in pre-order, followed by all the elements of `r` in pre-order.
   - For a binary tree `(Node v l r)` *post-order* means a list with all the elements `l` in post-order, followed by all the elements of `r` in post-order, followed by `v`.

2. Write a function to reverse a list:

   Example: `revList [1,3,5] = [5,3,1]`

3. Write a function to reverse a binary tree.