COMP 3958: Lab 4

Submit a zip file named `lab4.zip` containing your 2 source files: `bst.ml` and `kvt.ml`. As with the previous lab, you are restricted to the functions in the standard OCaml library. As before, you files must compile. Otherwise, you may fail to get credit. Maximum score: 14

1. The `List.iter` function with signature

   ```
   val iter : ('a -> unit) -> 'a list -> unit
   ```

   basically applies a function to each successive element of a list.

   We would like to do something similar for a binary search tree (BST). However, there are three ways to traverse a BST: preorder, inorder, and postorder traversal. Refer to the following for details:

   ```
   https://en.wikipedia.org/wiki/Binary_search_tree#Traversal
   ```

   Implement three functions `bst_preorder`, `bst_inorder` and `bst_postorder`. Each applies a function to the elements of a BST in the specified order. All three functions have signature:

   ```
   ('a -> unit) -> 'a bstree -> unit
   ```

   (Note: you may need to use type annotations when defining the functions.)

   You'll need to include the definition of `bstree`, the functions `bst_insert` and `bst_of_list` from class in order to use your functions. (Use the code without the comparison function.) For example,

   ```
   bst_postorder (Printf.printf "%d ") @@ bst_of_list [3; 2; 7; 6; 8]
   ```

   would output `2 6 8 7 3`

   Name your file `bst.ml`.

2. A binary search tree is usually used to store key-value pairs and we typically search for a particular key to find the corresponding value.

   Modify the binary search tree code from class to use 2 type parameters – one for the key and the other for the value. We'll call the new tree `kvtree` (for key-value tree). Its type is `('k, 'v) kvtree`.

   The signatures of the new functions are:

   ```
   val kvt_insert : ('k, 'v) kvtree -> cmp:('k -> 'k -> int) -> key:'k
                    -> value:'v -> ('k, 'v) kvtree
   val kvt_find : ('k, 'v) kvtree -> cmp:('k -> 'k -> int) -> 'k
                  -> 'v option
   val kvt_delete : ('k, 'v) kvtree -> cmp:(k' -> 'k -> int) -> 'k ->
                    -> ('k, 'v) kvtree
   val kvt_of_list : ('k * 'v') list -> cmp:('k -> 'k -> int)
                     -> ('k, 'v) kvtree
   ```

   Note that

   - each of the above functions has a labelled parameter (`cmp`) that specifies a comparison function used to compare keys. Its purpose is similar to the `cmp` parameter in `ListLabels.sort`. Note that `kvt_insert` has two additional labelled parameters.
   - for `kvt_insert`, if the key is already in the tree, the corresponding value is updated to the new value;
   - the `kvt_find` function replaces the `bst_mem` function from class; the new version needs to return the corresponding value if there is one; note that its return type is `'v option`.

   Name your file `kvt.ml`.