COMP 3958: Lab 5

Submit a zip file containing the `kvtree` directory. It should contain all source files for this lab. Do not submit the `_build` directory. If your program does not build, you may receive no credit for it. Be sure to comment any part of your code that may not be clear. Note that you may be asked to explain your work. Maximum score: 15

In class, we have seen how to implement a functor that takes a module containing a comparison function and returns a module for binary search trees. In this exercise, you are asked to do the same for key-value trees. (See lab 3.) Implement a module named `Kvtree` that has a functor named `Make` to create modules of key-value trees. `Make` takes a module that has type `OrderedType` (the type of the keys):

```
module type OrderedType = sig
  type t
  val compare : t -> t -> int
end
```

Make the key-value tree type abtract.

- You'll need to implement the 6 functions specified in lab 3
  - Their signatures need to be suitably modified. (For one thing, they will not have the `cmp` argument.)
  - They should be renamed to `empty`, `is_empty`, `insert`, `find_opt`, `delete` and `of_list` (i.e., without the `kvtree_` prefix).
- Provide 4 additional functions:
  - A `size` function that returns the number of key-value "pairs" in a tree.
  - A `find` function similar to the `find_opt` function but that raises an exception named `Not_found` (defined in `Kvtree`) if the specified key is not found. (Note that this exception is different from the `Not_found` exception in `Stdlib`.)
  - A `to_list` function that takes a tree and returns the list of key-value pairs in the tree in ascending order of keys (as specified by the `compare` function).
  - A `to_string` function that takes a "conversion" function and a tree and returns a string that represents the tree in the format shown below.

As an example, after doing

```
module M = Kvtree.Make(Int);;
let t = M.of_list [(3, "three"); (2, "two"); (7, "seven"); (6, "six"); (8, "eight")];;
```

- `M.to_list t` returns `[(2, "two"); (3, "three"); (6, "six"); (7, "seven"); (8, "eight")]`
- `M.to_string (fun (k, v) -> Printf.sprintf "%d, %s" k v) t` returns:

  ```
  "^(3, three, ^(2, two, #, #), ^(7, seven, ^(6, six, #, #), ^(8, eight, #, #)))"
  ```

  Note that `to_string` takes a function to "convert" a key-value pair to a string. The returned string basically shows the key & value in a node followed by the left subtree and then by the right subtree.

Note that since the tree type is abstract, `to_string` is the only function that allows us to visualize the tree. It is critical that it be correct as it will be used to check the tree after, for example, insert & delete operations. If `to_string` is incorrect, almost all tests will fail. A minimal test file is provided. The tests in the file also show the expected order of the arguments in each function.

Create a directory named `kvtree` and put your implementations in that directory. There should be a file named `kvtree.ml` and its corresponding interface file `kvtree.mli`. You may use additional files if necessary. Make sure your system can be built using the command: `ocamlbuild kvtree.cmo`.