Question 2 [25]: Tree Freak

A value `FL` of type `(string * int) list` is called a *frequency list* if and only if:

- Every string appearing in the first component of any pair in `FL` appears in exactly one pair in `FL`.
- Every integer appearing in the second component of any pair in `FL` is strictly positive.

For example,

```
[("dan", 9), ("ian", 4)]
```

is a frequency list but

```
[("dan", 9), ("ian", 4), ("zach", 0), ("ian", ~20)]
```

is not.

(a) (7 points) Write the function

```
insert : (string * int) -> (string * int) list -> (string * int) list
```

`insert` takes a frequency list `FL` and a new pair `(s_new,c_new)`. Conceptually, `insert` either updates the existing frequency of `s_new` if there is one or adds it to the list if there isn't. More formally:

- If there is some pair `(s,c)` in `FL` such that `s=s_new` then the returned frequency list contains the pair `(s_new,c+c_new)`.
- If there is no such element, then the returned frequency list contains the pair `(s_new,c_new)` in addition to everything in `FL`.

Given a valid frequency list and a pair `(s_new,c_new)` with `c_new > 0`, `insert` must evaluate to a valid frequency list.

Solution:

```
fun insert ((s_new, c_new) : string * int)
           (L : (string * int) list) =
    case L
     of [] => [(s_new,c_new)]
      | ((s,c)::scs) =>
        case s = s_new
         of true => (s, c_new + c)::scs
          | false => (s,c) :: (insert (s_new,c_new) scs)
```

(b) (6 points) Write the function

```
combine : (string * int) list -> (string * int) list
          -> (string * int) list
```

`combine` takes two frequency lists and combines them into one frequency list pairing each string with the sum of its counts in the inputs. More formally, if `FL1` and `FL2` are two frequency lists, and `CFL` is the result of calling `combine FL1 FL2`:

- If `(s,c1)` appears in `FL1` and `(s,c2)` appears in `FL2`, then `(s,c1 + c2)` appears in `CFL`.
- If `(s,c)` appears in one of the input list and `s` does not appear in the other list then `(s,c)` appears in `CFL`.
- If `s` does not appear in either `FL1` or `FL2`, then `s` does not appear in `CFL`.

For example, if we have the two frequency lists

```
val FL1 =  [("a", 4), ("b", 1)]
val FL2 =  [("a", 12), ("b", 4), ("x", 9) ]
```

then `combine FL1 FL2` should evaluate to the frequency list

```
[("a", 16), ("b", 5), ("x", 9) ]
```

While writing this function, you may assume that the `insert` function satisfies the above specification.

Solution:

```
fun combine (L1 : (string * int) list)
            (L2 : (string * int) list) : (string * int) list =
    case L1
     of [] => L2
      | (s,c)::scs => combine scs (insert (s,c) L2)
```

Recall the datatype of trees that carry data only at their leaves

```
datatype 'a tree = Empty
                 | Leaf of 'a
                 | Node of 'a tree * 'a tree
```

We will now relate frequency lists to trees. Let `T` be a value of type `string tree` and `FL` be a frequency list. We say that `FL` is a frequency list of `T` if and only if

- A string appears in a leaf of `T` if and only if it appears as the first component of a pair in `FL`.
- If a particular string `s` appears in k leaves of `T`, then `(s,k)` appears in `FL`.

For example, one frequency list for the tree

```
Node(Node(Leaf "a", Leaf "b"), Leaf "a")
```

is

```
[("a", 2), ("b", 1)]
```

However,

```
[("a", 1), ("b", 1), ("c", 12)]
```

is not a frequency list for this tree because `"a"` appears with the wrong count and `"c"` appears in no leaf of the tree.

(c) (7 points) Write the function

```
count : string tree -> (string * int) list
```

`count` takes a tree and builds a frequency list for that tree. While writing this function, you may assume that the `combine` function satisfies the above specification.

Solution:

```
fun count (t : string tree) : (string * int) list =
  case t
    of Empty => []
     | Leaf x => [(x,1)]
     | (Node(t1, t2)) => combine (count t1) (count t2)
```

(d) (5 points) Recall the definition of map-reduce on the type `'a tree` from Homework 6

```
fun mapreduce (l : 'a -> 'b)
              (e : 'b)
              (n : 'b -> 'b -> 'b)
              (t : 'a tree) : 'b =
    case t
     of Leaf x => l x
      | Empty => e
      | Node (t1,t2) => n (mapreduce l e n t1) (mapreduce l e n t2)
```

Using `mapreduce`, write the function

```
count_mr : string tree -> (string * int) list
```

`count_mr` takes a tree and builds a frequency list for that tree by calling map-reduce with appropriate arguments. `count_mr` is not a recursive function. While writing this function, you may assume that the `combine` function satisfies the above specification.

You may not use the function `count` while writing `count_mr`.

Solution:

```
fun count_mr t (t : string tree) : (string * int) list =
    mapreduce (fn x => [(x,1)])
              []
              combine
              t
```