CMSC 331 Principles of Programming Language

Homework - 2

1. Write a function called "repeatAndMerge", with the following type spec:

   ```
   repeatAndMerge: int -> string -> string -> string
   ```

   Here are a few sample runs to illustrate the expected behavior:

   ```
   repeatAndMerge 0 "a" "b"
   val it = "" : string

   repeatAndMerge 2 "a" "b"
   val it = "aabb" : string

   repeatAndMerge 3 "x" "y"
   val it = "xxxyyy" : string

   repeatAndMerge 1 "hello" "world"
   val it = "helloworld" : string
   ```

2. a) Write a function called "doubleAndSum", with the following type spec:

   ```
   doubleAndSum: int list -> int
   ```

   This function should take a list of integers, double each integer, and then sum the doubled values.

   Sample runs:

   ```
   doubleAndSum [1, 2, 3];
   val it = 12 : int

   doubleAndSum [5, -1, 2];
   val it = 12 : int
   ```

Given the functions:

```
fun curry f x y = f (x, y);
fun uncurry f (x, y) = f x y;
```

Determine the type specs of:

b) fun amplify (factor, nums) = doubleAndSum(List.map (fn x => x * factor) nums);

c) val amplifyCurried = curry amplify;

3. Given the following data type:

```
datatype iTree = EMPTY |
                 NODE of int * iTree * iTree;
```

In this definition, each node in the tree (NODE) holds an integer and has two children, which are also of type iTree (either NODE or EMPTY).
You need to write a function called sumTree with the type specification:

sumTree : iTree -> int list

The function should traverse the tree in a pre-order fashion (root, then left, then right) and perform the following operations:

a. For each node, calculate the sum of its value and all values in its right subtree.
b. Only include the result of step a in the output list for the root and all right nodes (ignore the left subtree of each node after the root).

Sample Runs:

```
sumTree (NODE(10, EMPTY, NODE(5, EMPTY, NODE(2, EMPTY, EMPTY))));
val it = [17, 7, 2] : int list;
```

Explanation: The root node 10 plus the sum of its right subtree 5 + 2 = 7, making 17. Then, for the right node 5 plus its right subtree 2 makes 7, and finally, the leaf node 2 is just 2.

```
sumTree (NODE(3, NODE(2, EMPTY, EMPTY), NODE(4, EMPTY, NODE(1, EMPTY,
EMPTY))));
val it = [8, 5] : int list;
```

```
sumTree EMPTY;
val it = [] : int list;
```

4. a) Write a function which inserts strings into the cTree datatype.

   Your function should have this type spec: string -> cTree -> cTree

   Note:

   i) the strings should be stored in-order in the tree.

   ii) inserting another copy of a string should just increase the count field on that node.

   b) After that, Write a function that searches for a string in a cTree data type and returns the count associated with that string. If the string is not found, the function should return 0.

   Your function should have this type spec: string -> cTree -> int