Section #3

Ashley Taylor and Shreya Shankar (Based on handouts by previous CS106B instructors and TAs)

Sets, Maps, and Recursion

1. Remove Duplicates (Sets)

Write a function named `removeDuplicates` that accepts as a parameter a reference to a `Vector` of integers, and modifies it by removing any duplicates. Note that the elements of the vector are not in any particular order, so the duplicates might not occur consecutively. You should retain the original relative order of the elements. Use a `Set` as auxiliary storage to help you solve this problem. For example, if a vector named `v` stores {`4, 0, 2, 9, 4, 7, 2, 0, 0, 9, 6, 6`}, the call of `removeDuplicates(v);` should modify it to store `4, 0, 2, 9, 7, 6`.

2. Friend List (Maps)

Write a function named `friendList` that accepts a file name as a parameter and reads friend relationships from a file and stores them into a compound collection that is returned. You should create a map where each key is a person's name from the file, and the value associated with that key is a set of all friends of that person. Friendships are bi-directional: if Ashley is friends with Shreya, Shreya is friends with Ashley.

The file contains one friend relationship per line, consisting of two names. The names are separated by a single space. You may assume that the file exists and is in a valid proper format. If a file named `buddies.txt` looks like this:

```
Ashley Colin
Shreya Ashley
```

Then the call of `friendList("buddies.txt")` should return a map with the following contents:

```
{"Colin":{"Ashley"}, "Shreya":{"Ashley"}, "Ashley":{"Colin", "Shreya"}}
```

Constraints:

- You may open and read the file only once. Do not re-open it or rewind the stream.
- You should choose an efficient solution. Choose data structures intelligently and use them properly.
- You may create one collection (stack, queue, set, map, etc.) or nested/compound structure as auxiliary storage. A nested structure, such as a set of vectors, counts as one collection. (You can have as many simple variables as you like, such as ints or strings.)

3. Recursion Mystery

```
int recursionMysteryDivMod(int n) {
    if (n < 0) {
        return recursionMysteryDivMod(-n);
    } else if (n < 10) {
        return n;
    } else {
        return n % 10 + recursionMysteryDivMod(n / 10);
    }
}
```

What is the output of `recursionMysteryDivMod(8)`?

4. Print Stars (Recursion)

Write a recursive function named `printStars` that accepts an integer parameter `n` and prints n occurrences of the `*` character to the console. For example, the call of `printStars(5);` should print `*****`. Do not use loops or auxiliary data structures; solve the problem recursively. You may assume that the value passed is non-negative.

5. Star String (Recursion)

Write a recursive function named `starString` that accepts an integer parameter `n` and returns a string of stars (asterisks) `2n` long (i.e.., 2 to the nth power). For example, `starString(0)` returns `*`; `starString(1)` returns `**`; `starString(2)` returns `****`; `starString(3)` returns `********`.

6. Sum of Squares (Recursion)

Write a recursive function named `sumOfSquares` that accepts an integer parameter `n` and returns the sum of squares from 1 to n. For example, the call of `sumOfSquares(3)` should return 12 + 22 + 32 = 14. If your function is passed 0, return 0. If passed a negative number, your function should throw an `int` as an exception.

7. Subsequence (Recursion)

Write a recursive function named `isSubsequence` that accepts two string parameters, and returns if the second string is a subsequence of the first string. A string is a subsequence of another if it contains the same letters in the same order, but not necessary consecutively. You can assume both strings are already lowercased.