CS106B Chris Gregg
Summer 2017

Section Handout #2

This week has more practice with data structures, in particular Queues, Stacks, Sets, and Maps. There are also some practice string problems.

1. Twice *(Sets)*
Write a function named `twice` that takes a vector of integers and returns a set containing all of the numbers in the vector that appear exactly twice. You can only use Sets as auxiliary storage. For example, passing `{1, 3, 1, 4, 3, 7, -2, 0, 7, -2, -2, 1}` returns `{3, 7}`.

2. UnionSets *(Sets)*
Write a function named `unionSets` that takes a set of sets of ints, and returns the union of all of the sets of ints. (A union is the combination of everything in each set.) For example, if a `Set` variable named `sets` stores the set of integers `{{1, 3}, {2, 3, 4, 5}, {3, 5, 6, 7}}`, the call of `unionSets(sets)` should return `{1, 2, 3, 4, 5, 6, 7}`.

3. Reorder *(Stacks/Queues)*
Write a function named reorder that takes a queue of integers that are already sorted by absolute value, and modifies it so that the integers are sorted normally. The only auxiliary data structure you can use is a single `Stack<int>`. For example, passing the queue `{1, -2, 3, 4, -5, -6, 7}` changes it to `{-6, -5, -2, 1, 3, 4, 7}`.

4. CheckBalance *(Stacks/Strings)*
Write a function named `checkBalance` that accepts a string of source code and uses a Stack to check whether the braces/parentheses are balanced. Every `(` or `{` must be closed by a `}` or `)` in the opposite order. Return the index at which an imbalance occurs, or `-1` if the string is balanced. If any `(` or `{` are never closed, return the string's length.

Here are some example calls:

```
//    index   01234567890123456789012345678901234567890
checkBalance("if (a(4) > 9) { foo(a(2)); }")         // returns -1 because balanced
checkBalance("for (i=0;i&lt;a(3};i++) { foo{); )")   // returns 14 because } out of order
checkBalance("while (true) foo(); }{ ()")            // returns 20 because } doesn't match any {
checkBalance("if (x) {"}                             // returns 8 because { is never closed
```

Constraints: Use a single stack as auxiliary storage.

5. Friend List *(Maps)*
Write a function named `friendList` that takes in a file name, reads friend relationships from a file, and writes them to a map. You should return the populated map. Friendships are bi-directional; if Chris is friends with Anton, Anton is friends with Chris. The file contains one friend relationship per line. The names are separated by a single space. You don't have to worry about malformed entries (assume all entries are formatted correctly).

Thanks to Anton Apostolatos, Aaron Broder, Marty Stepp, Victoria Kirst, Jerry Cain, and other past CS106B and X instructors / TAs for contributing content on this handout

If an input file named `buddies.txt` looked like this:

```
Caesar Chris
Chris Jason
```

Then the call of `friendList("buddies.txt")` should return a resulting map that looks like this:

```
{"Caesar": {"Chris"}, "Jason": {"Chris"}, "Chris": {"Caesar", "Jason"}}
```

6. Reverse *(Maps)*
Write a function named `reverse` that accepts a map from ints to strings, and returns a map with the associations reversed. For example, if a `Map` variable named `map` stores `{1:"a", 2:"b", 3:"c"}`, the call of `reverse(map)` should return `{"a":1, "b":2, "c":3}`. If there are any duplicate values `(k1, v)` and `(k2, v)` in the original map, your returned map may contain either `(v, k1)` or `(v, k2)`.

7. CrazyCaps *(Strings)*
Write a function named `crazyCaps` that accepts a string reference as a parameter and changes that string to have its capitalization altered such that the characters at even indexes are all in lowercase and odd indexes are all in uppercase. For example, if a variable s stores `"Hey!! THERE!"`, the call of `crazyCaps(s)`; should change s to store `"hEy!! tHeRe!".`

8. SwapPairs *(Strings)*
Write a function named `swapPairs` that accepts a string reference as a parameter and changes that string so that each pair of adjacent letters will be reversed. If the string has an odd number of letters, the last letter is unchanged. For example, if a variable s stores `"example"`, the call of `swapPairs(s)`; should change s to store `"xemalpe"`. If `s` had been named `"hello there"`, the call would produce `"ehll ohtree"`.