Chris Gregg
CS106B

Section #2
April 16, 2018

Section Handout #2: Sets, Maps and Recursion

Based on handouts by various current and past CS106B/X instructors and TAs.

Sets

1. Twice

Write a function named **`twice`** that takes a reference to a Vector of integers and returns a set containing all the numbers in the vector that appear exactly twice. **Bonus:** solve this problem using only Sets as auxiliary data structures!

Example: passing **`{1, 3, 1, 4, 3, 7, -2, 0, 7, -2, -2, 1}`** returns **`{3, 7}`**.

2. UnionSets

Write a function named **`unionSets`** that takes a reference to 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}`**.

Maps

1. Rarest

Write a function named **`rarest`** that accepts a reference to a Map from strings to strings as a parameter and returns the value that occurs least frequently in the map. If there is a tie, return the value that comes earlier in ABC order. For example, if a variable called **`map`** contains the following elements:

```
{"Alyssa":"Harding", "Char":"Smith", "Dan":"Smith",
  "Jeff":"Jones", "Kasey":"Jones", "Kim":"Smith",
"Morgan":"Jones", "Ryan":"Smith", "Stef":"Harding"}
```

Then a call of **`rarest(map)`** would return **`"Harding"`** because that value occurs 2 times, fewer than any other. Note that we are examining the *values* in the map, not the keys. You may assume the map passed is not empty.

2. FriendList

Write a function named **`friendList`** that takes in a file name, reads friend relationships from a file, and writes them to a Map (from friend A to friend B) that it then returns. Friendships are bi-directional; if Chris is friends with Nick, Nick is friends with Chris. The file contains one friend relationship per line, with names separated by a single space. You do not have to worry about malformed entries. If an input file named **`buddies.txt`** looked like this:

```
Nick Chris
Chris Mehran
```

Then the call of **`friendList("buddies.txt")`** should return a resulting map that looks like this:
`{"Chris":{"Nick","Mehran"}, "Nick":{"Chris"}, "Mehran":{"Chris"}}`

3. Reverse Map

Write a function named **`reverseMap`** that accepts a reference to 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 **`reverseMap(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)`**.

Recursion

1. Mystery Trace

For each call to the following method, indicate what output is printed.

```
void mystery1(int x, int y) {
    if (y == 1) {
        cout << x;
    } else {
        cout << (x * y) << ", ";
        mystery1(x, y - 1);
        cout << ", " << (x * y);
    }
}
```

Call:	Output:
`mystery1(4, 1)`	______________________________
`mystery1(8, 2)`	______________________________
`mystery1(3, 4)`	______________________________

2. Sum of Squares

Write a recursive function named **`sumOfSquares`** that takes in an integer **`n`** and returns the sum of squares from 1 to **`n`** inclusive. For example, **`sumOfSquares(3)`** should return 14 ($1^2 + 2^2 + 3^2 = 14$). You can assume **`n`** ≥ 1.

3. Reverse String

Write a recursive function reverse that takes in a string **`s`** and returns a string with the same characters in reverse order. For example, **`reverseString("Hi, you!")`** returns **`"!uoy ,iH"`**. You shouldn't modify the original string.

4. Star String

Write a recursive function named `starString` that takes in an integer `n` and returns a string of 2^n asterisks. For example,

```
starString(1)          "**"
starString(2)          "****"
starString(4)          "****************" // 16 stars
```

How many recursive calls does your function end up making (as a function of n)?

5. Is Subsequence

Write a recursive function named `isSubsequence` that takes two strings and returns `true` if the second string is a subsequence of the first string, or `false` otherwise. A string is a subsequence of another if it contains the same letters in the same order, but not necessarily consecutively. You can assume both strings are all lowercase characters. For example,

```
isSubsequence("computer", "core")          false
isSubsequence("computer", "cope")          true
isSubsequence("computer", "computer")      true
```

6. Double Stack

Write a recursive function named `doubleStack` that takes a reference to a stack of ints and replaces each integer with two copies of that integer. For example, if `s` stores `{1, 2, 3}`, then `doubleStack(s);` changes it to `{1, 1, 2, 2, 3, 3}`.

7. Zig Zag

Write a recursive function named `zigzag` that prints `n` characters as follows. The middle character (or middle two characters if `n` is even) is an asterisk (*). All characters before the asterisks are '<'. All characters after are '>'. You can assume that `n` is positive. (You do not need to worry about printing an `endl` at the end.)

```
zigzag(1)        *
zigzag(4)        <**>
zigzag(9)        <<<<*>>>>
```

8. Directory Crawl

Write a function `crawl` that accepts a filename as a parameter, as well as the initial indentation of the output (as a string) and prints information about that file. If the name represents a normal file, just print its name. If the name represents a directory, print its name and information about <u>every file/directory inside it</u>, indented further by 4 spaces. For example, the following could be output printed for the call `crawl("user/documents/course", "")`:

```
course
```

```
handouts
    syllabus.doc
    lecture-schedule.xls
homework
    1-gameoflife
        life.cpp
        life.h
        GameOfLife.pro
```

Note that this problem fits well with recursion, since file systems are recursive! (a directory can itself contain other directories). Also note that you should only print out the "tail" of the path at each step – for instance, for `"user/documents/course"` you should first print out `"course"`, and then print out everything inside of that folder. There are some filesystem methods that may come in handy:

`bool isDirectory(string filename)`	Returns whether this file name represents a directory.
`void listDirectory(string path, Vector<string> &list)`	Adds an alphabetized list of the files in the specified directory to the string vector **`list`**.
`getTail(string filename)`	Returns the last component of a path name. The components of the path name can be separated by any of the directory path separators (forward or reverse slashes). For instance, **`getTail("cs/106b/section2")`** returns **`section2`**, which is the last filepath component.