CS106B Chris Gregg
Summer 2017

Section Handout #3

This week has practice with recursion, in particular exhaustive search and recursive backtracking. For any parameter that is passed by reference, that parameter must be the same when the function returns. You're also welcome to use helper functions for any of these problems. Remember that many of these problems can be found on CodeStepByStep.

1. Tracing a Mystery
For each call to the following method, indicate what value is returned.

```
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` returns the sum of squares from 1 to `n` inclusive. For example, `sumOfSquares(3)` should return 14 (because $1^2 + 2^2 + 3^2 = 14$). You can assume `n` ≥ `1`.

3. Reverse
Write a recursive function `reverse` that takes in a string `s` and returns a string with the same characters in reverse order. For example, `reverse("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
```

What should your function do if `n` is negative? How many recursive calls does your function end up making (as a

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.

function of `n`)?

5. 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. 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. Make Change
Write a recursive function called `makeChange` that takes in a target amount of change and a Vector of coin values and prints out every way of making that amount of change, using only the coin values in coins. For example, if you need to make change using only pennies, nickels, and dimes, the coins vector would be `{1, 5, 10}`. Each way of making change should be printed as the *number of each coin used* in the coins vector. For example, if you were to use the above coins vector to make change for 15 cents, the possibilities would be

```
{15, 0, 0}, {10, 1, 0}, {5, 2, 0}, {5, 0, 1}, {0, 3, 0}, {0, 1, 1}
```

In the outputs for the example, the first element of each vector indicates the number of pennies used, the second indicates the number of nickels, and the third indicates the number of dimes.

7. Print Squares
Write a recursive function named `printSquares` that uses backtracking to find all ways to express an integer as a sum of squares of unique positive integers. For example you can express the integer 200 as the following sums of squares:

```
1^2 + 2^2 + 3^2 + 4^2 + 5^2 + 8^2 + 9^2
1^2 + 2^2 + 3^2 + 4^2 + 7^2 + 11^2
1^2 + 2^2 + 5^2 + 7^2 + 11^2
1^2 + 3^2 + 4^2 + 5^2 + 6^2 + 7^2 + 8^2
1^2 + 3^2 + 4^2 + 5^2 + 7^2 + 10^2
2^2 + 4^2 + 6^2 + 12^2
2^2 + 14^2
3^2 + 5^2 + 6^2 + 7^2 + 9^2
6^2 + 8^2 + 10^2
```

Some numbers can't be represented in this format; if this is the case, your function should produce no output. The sum has to be formed with **unique** integers (note that in a given sum of squares, no integers are repeated).

8. Longest Common Subsequence
A string is a subsequence of another if it contains the same letters in the same order, but not necessarily consecutively. We're going to build on that concept this week. Write a recursive function named `longestCommonSubsequence` that takes in two strings and returns the longest string that is a subsequence of both input strings. For example,

```
longestCommonSubsequence("leslie", "wesley")              "esle"

longestCommonSubsequence("chris", "anupama")              ""

longestCommonSubsequence("she sells", "seashells")        "sesells"
```

9. Ways to Climb
Imagine you're standing at the base of a staircase. A small stride will move up one stair, and a large stride advances two. You want to count the number of ways to climb the staircase based on different combinations of large and small strides. Write a recursive function `waysToClimb` that takes in a positive integer value representing the number of stairs and prints out each unique way to climb a staircase of that height. For example, `waysToClimb(4)` should produce the following output:

```
{1, 1, 1, 1}
{1, 1, 2}
{1, 2, 1}
{2, 1, 1}
{2, 2}
```

10. Twiddle
Write a recursive function named `listTwiddles` that accepts a string `str` and a reference to an English language `Lexicon` and uses exhaustive search and backtracking to print out all those English words that are `str`'s twiddles. Two English words are considered twiddles if the letters at each position are either the same, neighboring letters, or next-to-neighboring letters.

For instance, `"sparks"` and `"snarls"` are twiddles. Their second and second-to-last characters are different, but `'p'` is two past `'n'` in the alphabet, and `'k'` comes just before `'l'`. A more dramatic example: `"craggy"` and `"eschew"` are also twiddles. They have no letters in common, but `craggy`'s `'c'`, `'r'`, `'a'`, `'g'`, `'g'`, and `'y'` are -2, -1, -2, -1, 2, and 2 away from the `'e'`, `'s'`, `'c'`, `'h'`, `'e'`, and `'w'` in `"eschew"`. And just to be clear, `'a'` and `'z'` are not next to each other in the alphabet; there's no wrapping around at all. (Note: any word is considered to be a twiddle of itself, so it's okay to print `str` itself.)

Constraints: Do not declare any global variables. You can use any data structures you like, and your code can contain loops, but the overall algorithm must be recursive and must use backtracking. You are allowed to define other "helper" functions if you like; they are subject to these same constraints. Do not modify the state of the Lexicon passed in.

11. Hacking and Cracking
Write a function that takes in the maximum length a site allows for a user's password to break into an account by using recursive backtracking to attempt all possible passwords up to that length (inclusive). Assume you have access to the function `bool login(string password)` that returns true if a password is correct. You can also assume that the passwords are entirely alphabetic and case-sensitive. You should return the correct password you find, or the empty string, if there isn't one. You should return the empty string if the maximum length passed is 0, or throw an integer exception if the length is negative.