# Section Handout

This first week, I'm only including three problems, because introductions might take 15 or so of the 50 minutes we have. Two of the problems are discussion problems, and the other is written up as an informal coding exercise. However, we'll allow your section leader and your peers to decide if you'd rather adopt the collaborative discussion-problem format for all of them.

### Discussion Problem 1: Publishing Stories

Social networking sites like Facebook, LinkedIn, and Google+ typically record and publish stories about actions taken by you and your friends. Stories like:

> Jessie Duan accepted your friend request.
> Matt Anderson is listening to Green Day on Spotify.
> Patrick Costello wrote a note called "Because Faiz told me to".
> David Wang commented on Jeffrey Spehar's status.
> Mike Vernal gave The French Laundry a 5-star review.

are created from story templates like

> {name} accepted your friend request.
> {name} is listening to {band} on {application}.
> {name} wrote a note called "{title}".
> {name} commented on {target}'s status.
> {actor} gave {restaurant} a {rating}-star review.

The specific story is generated from the skeletal one by replacing the tokens—substrings like **`"{name}"`**, **`"{title}"`**, and **`"{rating}"`**—with event-specific values, like **`"Jessie Duan"`**, **`"Because Faiz told me to"`**, and **`"5"`**. The token-value pairs can be packaged in a **`Map<string, string>`**, and given a story template and a data map, it's possible to generate an actual story.

Write the **`generateStory`** function, which accepts a story template (like **`"{actor} gave {restaurant} a {rating}-star review."`**) and a **`Map<string, string>`** (which might map **`"actor"`** to **`"Mike Vernal"`**, **`"restaurant"`** to **`"The French Laundry"`**, and **`"rating"`** to **`"5"`**), and builds a string just like the story template, except the tokens have been replaced by the text they map to.

Assume the following is true:

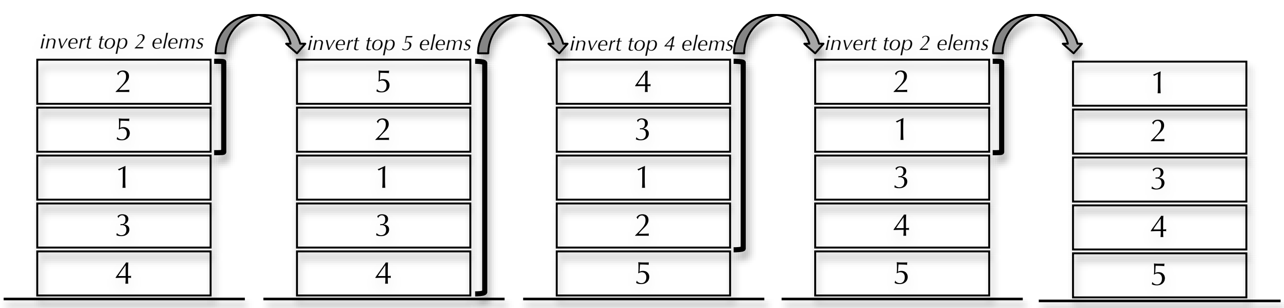
- **`'{'`** and **`'}'`** exist to delimit token names, but won't appear anywhere else. In other words, if you encounter the **`'{'`** character, you can assume it marks the beginning of a token that ends with a **`'}'`**.
- We guarantee that all tokens are in the **`Map<string, string>`**. You don't need to do any error checking.

The prototype is:

```
string generateStory(const string& storyTemplate,
                     const Map<string, string>& data);
```

### Discussion Problem 2: Topswopping and Topswop Numbers

When a **`Stack<int>`** of depth n contains the number 1 through n in some order, its **Topswop number** is defined to be the number of times the top of the stack must be **swopped** before the top element becomes a 1. Each swop amounts to an examination of the topmost element of the stack—let's call it k—and then the inversion of the top k elements. (A stack whose top element is already 1 has a Topswop number of 0.)




Implement the **`getTopswopNumber`** function, which accepts a copy of a **`Stack<int>`** and returns the number of times the **`Stack<int>`** must be swopped before the top element becomes a 1. You may assume the **`Stack<int>`** contains the numbers 1 through n in some order, and you can trust the fact that the process always terminates.

```
static int getTopswopNumber(Stack<int> s);
```

### Lab Problem 1: Keith Numbers

A Keith number is any n-digit number that appears in the Fibonacci-like sequence that starts off with the number's n digits and then continues such that each subsequent number is the sum of the preceding n.

All of the one digit numbers are—trivially so—Keith numbers.

The number 7385 is more interesting. It's a Keith number, because the following sequence says so:

7, 3, 8, 5, 23, 39, 75, 142, 279, 535, 1031, 1987, 3832, 7385

The sequence starts out 7, 3, 8, 5, because those are the digits making up 7385. Each number after the 5 is the sum of the four numbers that precede it (four, because 7385 has four digits). The fact that 7385—the number whose digits spawned it all—happens to be in the sequence is the happy accident that tells us it's a Keith number.

For this exercise, you should write a program that prints out all of the Keith numbers between 1 and 10000, inclusive, and for each also print out the Fibonacci-like sequence that proves it's Keith.

The meat of your program's output should be this:

```
1: [1]
2: [2]
3: [3]
4: [4]
5: [5]
6: [6]
7: [7]
8: [8]
9: [9]
14: [1, 4, 5, 9, 14]
19: [1, 9, 10, 19]
28: [2, 8, 10, 18, 28]
47: [4, 7, 11, 18, 29, 47]
61: [6, 1, 7, 8, 15, 23, 38, 61]
75: [7, 5, 12, 17, 29, 46, 75]
197: [1, 9, 7, 17, 33, 57, 107, 197]
742: [7, 4, 2, 13, 19, 34, 66, 119, 219, 404, 742]
1104: [1, 1, 0, 4, 6, 11, 21, 42, 80, 154, 297, 573, 1104]
1537: [1, 5, 3, 7, 16, 31, 57, 111, 215, 414, 797, 1537]
2208: [2, 2, 0, 8, 12, 22, 42, 84, 160, 308, 594, 1146, 2208]
2580: [2, 5, 8, 0, 15, 28, 51, 94, 188, 361, 694, 1337, 2580]
3684: [3, 6, 8, 4, 21, 39, 72, 136, 268, 515, 991, 1910, 3684]
4788: [4, 7, 8, 8, 27, 50, 93, 178, 348, 669, 1288, 2483, 4788]
7385: [7, 3, 8, 5, 23, 39, 75, 142, 279, 535, 1031, 1987, 3832, 7385]
7647: [7, 6, 4, 7, 24, 41, 76, 148, 289, 554, 1067, 2058, 3968, 7647]
7909: [7, 9, 0, 9, 25, 43, 77, 154, 299, 573, 1103, 2129, 4104, 7909]
```

Of course, you shouldn't actually print this verbatim, but instead include the logic needed to generate these numbers. You should be able to change the range of interest—perhaps from [1, 10000] to [10, 10000000]—and have the program still work.

The starter file for this—cleverly named **`keith-numbers.cpp`**—actually prints out all numbers between 1 and 10000, inclusive. You should update the program to identify which numbers are Keith and print just them.