

Section 1 (Week 2) - SOLUTION

Problem 1: Publishing Stories

There's the one-pass approach that just appends characters from the template to the running story, unless that character is '{', in which case we extract everything up through the matching '}' and replace it with a string from the data map.

```
string generateStory(string storyTemplate, Map<string, string>& data) {
    string story;
    for (int i = 0; i < storyTemplate.size(); i++) {
        if (storyTemplate[i] != '{') {
            story += storyTemplate[i];
        } else {
            int end = storyTemplate.find('}', i + 1);
            string token = storyTemplate.substr(i + 1, end - i - 1);
            story += data[token];
            i = end;
        }
    }
    return story;
}
```

Another approach is to iterate over the data map using the CS106 **foreach** extension and drive the substitution that way. It's less efficient, but it's more straightforward, and a perfectly acceptable answer for the purposes of a discussion section.

```
static string substituteOneToken(string story, string token, string value) {
    int start = 0;
    while (true) {
        int found = story.find(token, start);
        if (found == string::npos) return story;
        story.replace(found, token.size(), value);
        start = found + value.size() + 1;
    }
}

string generateStory(string storyTemplate, Map<string, string>& data) {
    string story = storyTemplate;
    foreach (string token in data) {
        token = '{' + token + '}';
        story = substituteOneToken(story, token, data[token]);
    }
    return story;
}
```

Problem 2: Keith Numbers

My approach: generate the Fibonacci-esque sequence for all numbers, but only enough needed to decide whether a number is Keith or not. The uncreative programmer in me went with a name of **generateKeithSequence**

```
static void generateKeithSequence(Vector<int>& sequence, int n) {
    string numString = integerToString(n);
    int numDigits = numString.size();
    for (int i = 0; i < numDigits; i++) {
        sequence.add(numString[i] - '0');
    }

    while (sequence[sequence.size() - 1] < n) {
        int next = 0;
        for (int i = sequence.size() - numDigits; i < sequence.size(); i++) {
            next += sequence[i];
        }
        sequence.add(next);
    }
}
```

Note the above function assumes the incoming **Vector<int>** is empty and should be populated with the relevant sequence of numbers. Provided you understand what the above function is doing, you'll see how it contributes to the larger program, which I capture in a **main** function right here:

```
int main() {
    for (int n = 1; n < 10000; n++) {
        Vector<int> sequence;
        generateKeithSequence(sequence, n);
        if (sequence[sequence.size() - 1] == n) {
            // sequence ends in n? we have a Keith number!!
            cout << n << ": [";
            for (int i = 0; i < sequence.size() - 1; i++) {
                cout << sequence[i] << ", ";
            }
            cout << n << "]" << endl;
        }
    }

    cout << endl;
    cout << "That's all of them! " << endl;
    return 0;
}
```