Section Solution

Handout written by Jerry Cain.

Solution 1: 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.

```
static bool isKeith(int n) {
   Vector<int> sequence;
   string str = integerToString(n);
   int numDigits = str.size();
   for (int i = 0; i < numDigits; i++) {
      sequence.add(str[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);
   }

   return sequence[sequence.size() - 1] == n;
}
```

Solution 2: 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.

```
const string generateStory(const string& storyTemplate,
                           const 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 a modern **for** loop dialect 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 substituteToken(string story,
```

```
                                const string& token, const 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(const string& storyTemplate,
                     const Map<string, string>& data) {
   string story = storyTemplate;
   for (const string& token: data) {
      story = substituteToken(story, '{' + token + '}', data[token]);
   }

   return story;
}
```

Solution 3: Topswopping and Topswop Numbers

```
static int getTopswopNumber(Stack<int> s) {
   int count = 0;
   while (true) {
      int top = s.peek();
      if (top == 1) return count;
      Queue<int> q;
      for (int i = 0; i < top; i++) q.enqueue(s.pop());
      while (!q.isEmpty()) s.push(q.dequeue());
      count++;
   }
}
```

Solution 4: Shunting-Yard Algorithm

```
static bool isOperator(char ch) {
   return ch == '+' || ch == '*';
}

static int precedence(char op) {
   return op == '+' ? 1 : 2;
}

static bool shouldReduce(const Stack<char>& stack, char op) {
   return !stack.isEmpty() && precedence(stack.peek()) >= precedence(op);
} // to reduce is to pop something from the stack and append it to the output

static string infixToPostfix(const string& infix) {
   string postfix;
   Stack<char> stack;
   for (size_t i = 0; i < infix.size(); i++) {
      if (isdigit(infix[i])) {
         postfix += infix[i];
      } else if (isOperator(infix[i])) {
         while (shouldReduce(stack, infix[i]) && stack.peek() != '(')
            postfix += stack.pop();
         stack.push(infix[i]);
      } else if (infix[i] == '(') {
         stack.push(infix[i]);
      } else { // infix[i] == ')'
         while (stack.peek() != '(')
```

```
            postfix += stack.pop();
         stack.pop();
      }
   }

   while (!stack.isEmpty())
      postfix += stack.pop();
   return postfix;
}
```