CS106B
Spring 2016 Cynthia Lee

Section 3 (Week 4) Handout

Section problems by Marty Stepp, with edits by Cynthia Lee

This week is all about practicing recursion, so all your code should be recursive, even if you can solve the problem iteratively. Try to make your recursion as elegant as possible. Avoid redundant cases and if statements. In addition, think about what kinds of inputs are invalid for each problem; your function should throw an error if it receives invalid input.

1. stutter.
Write a function named **stutter** that takes a stack of integers and replaces each integer with two copies of that integer.

```
stutter({1})                {1, 1}
stutter({1, 2, 3})          {1, 1, 2, 2, 3, 3}
```

```
void stutter(Stack<int>& s) { ...
```

2. starString.
Write a function named **starString** that returns a string of 2^n asterisks. Throw an error if n is negative.

```
starString(1)               "**"
starString(2)               "****"
starString(4)               "****************"
```

```
string starString(int n) { ...
```

3. writeChars.
Write a function named **writeChars** 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 '>'. Throw an error if n is not positive. (You do not need to worry about printing an `endl` at the end.)

```
writeChars(1)               "*"
writeChars(2)               "**"
writeChars(4)               "<**>"
writeChars(9)               "<<<<*>>>>"
```

```
void writeChars(int n) { ...
```

4. isMeasurable.
Write a function named **isMeasurable** that determines whether it is possible to measure out the desired target amount with a given set of weights. For example, if a sample `weights` is {1, 3}, you can measure a `target` of weight 2 by putting the 1 on the left and 3 on the right.

```
isMeasurable(2, {1, 3})          true
isMeasurable(5, {1, 3})          false
isMeasurable(6, {2, 3, 7})       true
```

```
bool isMeasurable(int target, Vector<int>& weights) { ...
```

5. waysToClimb.
Write a function named **waysToClimb** that returns the number of ways to climb the given number of stairs if you can only move either one or two steps at a time. For example, there are five ways to climb four steps: {1, 1, 1, 1}, {1, 1, 2}, {1, 2, 1}, {2, 1, 1}, {2, 2}. Throw an error if `steps` isn't positive.

```
waysToClimb(1)              1
waysToClimb(2)              2
waysToClimb(4)              5
```

```
int waysToClimb(int steps) { ...
```

6. isSubsequence.
Write a function named **isSubsequence** that takes two strings and returns 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 already lowercased.

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

```
bool isSubsequence(string big, string small) { ...
```

7. Debugging Recursion.
The following function recursively finds the maximum integer in a `Vector` between two indices (inclusive) by taking the maximum from the left half, the maximum in the right half, and then returning the max of those two. For example, if a `Vector` variable named `vec` contained the values {1, 2, 3, 2, 3, 4}, the call of `recursiveMax(vec, 0, 2)` looks at the elements in indices 0, 1, and 2, and returns 3 because that's the largest in those indices. Can you find the bug?

```
 1  int recursiveMax(Vector<int>& v, int left, int right) {
 2      if (left == right) {
 3          return v[left];
 4      } else if (left < right) {
 5          int middle = (left + right) / 2;
 6          int leftMax = recursiveMax(v, left, middle);
 7          int rightMax = recursiveMax(v, middle, right);
 8          if (leftMax > rightMax) {
 9              return leftMax;
10          } else {
11              return rightMax;
12          }
13      } else {
14          throw "Invalid range.";
15      }
16  }
```