CS 106B Section 3 (Week 4)

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.

Recommended problems: #2, #4
Extra practice problems: 7.8, 7.10 (from Textbook)

---

1. 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) { ...
```

---

2. 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 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) { ...
```

---

3. waysToClimb.

Write a function named **waysToClimb** that returns the number of ways to climb the given number of stairs, if you can only move up 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) { ...
```

CS 106B Section 3 (Week 4)

4. 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 necessary 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) { ...
```

5. Debugging Recursion.

The following function recursively finds the maximum integer in a Vector between two indexes (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 indexes 0, 1, and 2, and returns 3 because that's the largest in those indexes. 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 }
```