Week 2 Section

This week's section handout has practice with Grids, Vectors, Stacks, and Queues. You can practice all the problems on CodeStepByStep.

---

1. Collection Mystery (CollectionMystery6 on CodeStepByStep) - Stacks and Queues

Write the output produced by the following function when passed each of the following stacks. Note that stacks and queues are written in *front to back* order, with the oldest element on the left side of the queue/stack.

```
void collectionMystery(Stack<int>& s) {
    Queue<int> q;
    Stack<int> s2;

    while (!s.isEmpty()) {
        if (s.peek() % 2 == 0) {
            q.enqueue(s.pop());
        } else {
            s2.push(s.pop());
        }
    }

    while (!q.isEmpty()) {
        s.push(q.dequeue());
    }
    while (!s2.isEmpty()) {
        s.push(s2.pop());
    }

    cout << s << endl;
}
```

Stacks:	Output:
{1, 2, 3, 4, 5, 6}	_____
{42, 3, 12, 15, 9, 71, 88}	_____
{65, 30, 10, 20, 45, 55, 6, 1}	_____

Thanks to Marty Stepp and other CS106B and X instructors and TAs for contributing problems on this handout.

---

2. Mirror - Grid

Write a function **`mirror`** that accepts a reference to a `Grid` of integers as a parameter and flips the grid along its diagonal. You may assume the grid is square; in other words, that it has the same number of rows as columns. For example, the grid below at left would be altered to give it the new grid state at right:

```
{{ 6,  1,  9,  4},              {{6, -2, 14, 21},
 {-2,  5,  8, 12},               {1,  5, 39, 55},
 {14, 39, -6, 18},      -->      {9,  8, -6, 73},
 {21, 55, 73, -3}}               {4, 12, 18, -3}}
```

Bonus: How would you solve this problem if the grid were not square?

---

3. CrossSum - Grid

Write a function named **`crossSum`** that accepts three parameters - a reference to a `Grid` of integers, and two integers for a row and column - and returns the sum of all numbers in the row/column cross provided. For example, if a grid named `g` stores the following integers:

```
  col    0  1  2
row
 0      {{1, 2, 3},
 1       {4, 5, 6},
 2       {7, 8, 9}}
```

Then the call of `crossSum(g, 1, 1)` should return (4+5+6+2+8) or `25`. You may assume that the row and column passed are within the bounds of the grid. Do not modify the grid that is passed in.

---

4. RemoveConsecutiveDuplicates - Vector

Write a function named **`removeConsecutiveDuplicates`** that accepts as a parameter a reference to a `Vector` of integers, and modifies it by removing any consecutive duplicates. For example, if a vector named `v` stores `{1, 2, 2, 3, 2, 2, 3}`, the call of `removeConsecutiveDuplicates(v);` should modify it to store `{1, 2, 3, 2, 3}`.

---

5. Stutter - Queue

Write a function named **`stutter`** that accepts a reference to a queue of integers as a parameter and replaces every element with two copies of itself. For example, if a queue named `q` stores `{1, 2, 3}`, the call of `stutter(q);` should change it to store `{1, 1, 2, 2, 3, 3}`.

---

6. CheckBalance - Stack

Write a function named **`checkBalance`** that accepts a string of source code and uses a `Stack` to check whether the braces/parentheses are balanced. Every `(` or `{` must be closed by a `}` or `)` in the opposite order. Return the index at which an imbalance occurs, or `-1` if the string is balanced. If any `(` or `{` are never closed, return the string's length.
Here are some example calls:

```
//    index   01234567890123456789012345678901234567890
checkBalance("if (a(4) > 9) { foo(a(2)); }")  // returns -1 (balanced)

checkBalance("for (i=0;i<a;(3};i++) { foo{); )")
     // returns 14 because } is out of order

checkBalance("while (true) foo(); }{ ()")
     // returns 20 because } doesn't match any {

checkBalance("if (x) {")
     // returns 8 because { is never closed
```