CS 106B Practice Midterm Exam #3
ANSWER KEY

1. C++ Basics / Parameters

```
0 3 1
2 7 8
8 8 9
8 7 2 9
```

2. File I/O and Strings (write)

As with any programming problem, there are many correct solutions. Here is one:

```
void coinFlip(string filename) {
    ifstream input;
    input.open(filename);
    if (input.fail()) {
        return;
    }

    int heads = 0;
    int tails = 0;
    string token;
    while (input >> token) {
        if (tolower(token[0]) == 'h') {
            heads++;
        } else {
            tails++;
        }
    }
    int percent = 100 * heads / (heads + tails);
    cout << heads << " heads (" << percent << "%)" << endl;
    if (percent >= 50) {
        cout << "You win!" << endl;
    } else {
        cout << "You lose!" << endl;
    }
}
```

3. ADTs / Collections (read)

Queue	Output
a) {1, 2, 3, 4, 5, 6}	q={1, 3, 5} s={2, 4, 6}
b) {42, -3, 4, 15, 9, 71}	q={-3, 15, 9, 71} s={42, 4}
c) {30, 20, 10, 60, 50, 40, 3, 0}	q={3} s={30, 20, 10, 60, 50, 40, 0}

4. ADTs / Collections (write)

As with any programming problem, there are many correct solutions. Here are two:

```
void removeBadPairs(Vector<int>& v) {         // O(N)
    if (v.size() % 2 != 0) {
        v.remove(v.size() - 1);
    }
    for (int i = v.size() - 1; i > 0; i--) {
        if (i % 2 != 0 && v[i - 1] > v[i]) {
            v.remove(i);
            v.remove(i - 1);
        }
    }
}

void removeBadPairs(Vector<int>& v) {         // O(N^2) worst case
    if (v.size() % 2 != 0) {
        v.remove(v.size() - 1);
    }
    for (int i = 0; i < v.size(); i += 2) {
        if (v[i] > v[i + 1]) {
            v.remove(i);
            v.remove(i);
            i -= 2;
        }
    }
}
```

5. Big-Oh Analysis (read)

a) $O(N \log N)$
b) $O(N^2)$
c) $O(N^2)$
d) $O(N)$
e) $O(N^4)$

6. Recursion (read)

Call	Returns
a) recursionMystery3(7)	8
b) recursionMystery3(42)	503
c) recursionMystery3(385)	40906
d) recursionMystery3(-790)	-80001
e) recursionMystery3(89294)	900030005

7. Recursion (write)

Here are two working solutions:

```
// "don't modify the vectors; use a helper function" solution  (O(N))
int matchCount(const Vector<int>& v1, const Vector<int>& v2) {
    return matchCountHelper(v1, v2, 0);
}

int matchCountHelper(const Vector<int>& v1, const Vector<int>& v2, int index) {
    if (v1.size() <= index || v2.size() <= index) {
        return 0;   // base case: past end of one or more vectors
    } else {
        // recursive case: compare current element and continue
        if (v1[index] == v2[index]) {
            return 1 + matchCountHelper(v1, v2, index + 1);
        } else {
            return matchCountHelper(v1, v2, index + 1);
        }
    }
}


// "modify the vectors but then restore them" solution  (O(N^2))
int matchCount(Vector<int>& v1, Vector<int>& v2) {
    if (v1.isEmpty() || v2.isEmpty()) {
        return 0;   // base case: empty vector has no matches
    } else {
        int a = v1[0];   // recursive case: remove/compare first elements, recur over the rest
        int b = v2[0];
        v1.remove(0);
        v2.remove(0);
        int count = matchCount(v1, v2);
        if (a == b) {
            count++;
        }
        v1.insert(0, a);
        v2.insert(0, b);
        return count;
    }
}
```

8. Recursive Backtracking (write)

```
bool isMeasurable(Vector<int>& weights, int target) {
    if (weights.isEmpty()) {
        return target == 0;   // base case; no weights left to place
    } else {
        // recursive case;
        // this call will explore all possible placements for a single one of the weights

        int last = weights[weights.size() - 1];    // could use any index, but last is fastest
        weights.remove(weights.size() - 1);

        // "choose and explore" all of the three possibilities:
        // putting the last element on left side of scale (+), right side (-), or neither
        bool result = isMeasurable(weights, target + last)
                   || isMeasurable(weights, target - last)
                   || isMeasurable(weights, target);

        weights.add(last);                         // un-choose
        return result;
    }
}
```

9. Implementing a Collection Class (write)

```
int ArrayList::longestSortedSequence() const {
    if (mysize == 0) {
        return 0;
    }
    int max = 1;
    int current = 1;
    for (int i = 1; i < mysize; i++) {
        if (elements[i] >= elements[i - 1]) {
            current++;
            if (current > max) {
                max = current;
            }
        } else {
            current = 1;
        }
    }
    return max;
}
```

10. Pointers and Linked Nodes

```
list2->next->next = list;          // 4 -> 1
list->next->next = list2;          // 2 -> 3
list = list2->next;                // list -> 4
list2 = list->next->next;          // list2 -> 2
list->next->next = NULL;           // 1 /
list2->next->next = NULL;          // 3 /
```