Nick Troccoli
CS 106X

Section #1
Week 2

Section #1 Solutions

Based on handouts by various current and past CS106B/X instructors and TAs.

1. Mirror *(Grid)*

```
void mirror (Grid<int>& grid) {
    for (int r = 0; r < grid.numRows(); r++) {
        // start at r+1 rather than 0
        // to avoid double-swapping
        for (int c = r + 1; c < grid.numCols(); c++) {
            int temp = grid[r][c];
            grid[r][c] = grid[c][r];
            grid[c][r] = temp;
        }
    }
}
```

2. reorder *(Stack, Queue)*

```
void reorder(Queue<int>& q) {
    Stack<int> s;
    int size = q.size();
    for (int i = 0; i < size; i++) {   // separate positive and negative numbers
        int n = q.dequeue();
        if (n < 0) {
            s.push(n);
        } else {
            q.enqueue(n);
        }
    }
    size = q.size();                   // enqueue negative numbers in reverse order
    while (!s.isEmpty()) {
        q.enqueue(s.pop());
    }
    for (int i = 0; i < size; i++) {   // move positive numbers to end
        q.enqueue(q.dequeue());
    }
}
```

3. Stacks(s) as a Queue *(Stack, Queue)*

```
void push(Queue<int>& mainQ, int entry) {   //solution 1: Using two Queues
    Queue<int> auxiliaryQ;
    while (!mainQ.isEmpty()) {
        auxiliaryQ.enqueue(mainQ.dequeue());
    }
    mainQ.enqueue(entry);
    while(!auxiliaryQ.isEmpty()) {
        mainQ.enqueue(auxiliaryQ.dequeue());
    }
}

int pop(Queue<int>& mainQ) {
    return mainQ.dequeue();
}
```

```
void push(Queue<int>& q, int entry) {   //solution 2: Using one Queue
    q.enqueue(entry);
    for (int i = 0; i < q.size() - 1; i++) {
        q.enqueue(q.dequeue());
    }
}

int pop(Queue<int>& q) {
    return q.dequeue();
}

Bonus: use two Stacks - one "in" Stack and one "out" Stack. To enqueue, simply push
the new element onto the "in" Stack. To dequeue, if the "out" Stack is not empty,
then pop one element off the "out" Stack. Otherwise, pop everything off the "in"
Stack and push it onto the "out" Stack, then pop one element off the "out" Stack.
```

4. stretch *(Vector)*

```
void stretch(Vector<int>& v) {
    int size = v.size();
    for (int i = 0; i < size * 2; i += 2) {
        int n = v[i];
        v[i] = n / 2 + n % 2;
        v.insert(i + 1, n / 2);
    }
}
```

5. removeConsecutiveDuplicates *(Vector)*

```
void removeConsecutiveDuplicates(Vector<int>& v) {
    for (int i = 0; i < v.size() - 1; i++) {
        if (v[i] == v[i + 1]) {
            v.remove(i + 1);
            i--;
        }
    }
}
```

6. Keith Numbers *(Vector)* – Problem by Keith Schwarz

```
bool isKeithNumber(int n) {
    int sum = 0;
    int digits = n;
    int numDigits = 0;

    Vector<int> sequence; // remove this line for the bonus solution

    while (digits > 0) {
        int digit = digits % 10;
        sum += digit;
        sequence.insert(0, digit);
        digits /= 10;
        numDigits++;
    }
                                            // solution continues on next page
```

```
                                             // solution continued from previous page
while (sequence[sequence.size() - 1] < n) {
        sequence.add(sum);
        sum = sum - sequence[sequence.size() - numDigits - 1] + sum;
    }
    return sequence[sequence.size() - 1] == n;
}


// bonus solution:

/* first, modify isKeithNumber to accept a Vector<int> by reference rather than
   declaring one inside the function: */

bool isKeithNumber(Vector<int>& sequence, int n) {
    // ... see previous page
}

void findKeithNumbers(int min, int max) {
    for (int n = min; n <= max; n++) {
        Vector<int> sequence;
        if (isKeithNumber(sequence, n)) {
            cout << n << ": " << sequence << endl;
        }
    }
}
```

7. Big-O Analysis

a) $\boldsymbol{O}(\mathbf{N})$ b) $\boldsymbol{O}(\boldsymbol{N}^{2})$

c) $\boldsymbol{O}(\boldsymbol{N}^{4})$ d) $\boldsymbol{O}(\mathbf{1})$

e) $\boldsymbol{O}(\boldsymbol{N}\log\boldsymbol{N})$ f) $\boldsymbol{O}(\boldsymbol{N}^{2})$

8. friendList *(Map)*

```
Map<string, Vector<string> > friendList(const string& filename) {
    ifstream infile(filename.c_str());
    Map<string, Vector<string> > friends;
    string s1, s2;
    while(infile >> s1 >> s2) {
        friends[s1] += s2;
        friends[s2] += s1;
    }
    infile.close();
    return friends;
}
```

9. twice *(Set)*

```
// regular problem solution
Set<int> twice(const Vector<int>& v) {
    Map<int, int> counts;
    for (int n : v) {
        counts[n]++;
    }
    Set<int> twice;
    for (int k : counts) {
        if (counts[k] == 2) {
            twice += k;
        }
    }
    return twice;
}
```

```
// bonus solution - only using Sets
Set<int> twice(const Vector<int>& v) {
    Set<int> once, twice, more;
    for (int n : v) {
        if (once.contains(n)) {
            once.remove(n);
            twice.add(n);
        } else if (twice.contains(n)) {
            twice.remove(n);
            more.add(n);
        } else if (!more.contains(n)) {
            once.add(n);
        }
    }
    return twice;
}
```

10. numInCommon *(Set)*

```
int numInCommon(const Vector<int>& v1, const Vector<int>& v2) {
    Set<int> s1, s2;
    for (int value : v1) {
        s1.add(value);
    }
    for (int value : v2) {
        s2.add(value);
    }

    int common = 0;
    for (int value : s1) {
        if (s2.contains(value)) {
            common++;
        }
    }
    return common;
}
```

11. unionSets *(Set)*

```
Set<int> unionSets(const HashSet<Set<int> >& sets) {
    Set<int> all;
    for (Set<int> s : sets) {
        all += s;
    }
    return all;
}
```