CS106B
Spring 2016 Cynthia Lee

Section 1 (Week 2) Handout

Section problems by Marty Stepp, with edits by Cynthia Lee

Welcome to CS106B Section!

Weekly section is your time to get a small-group experience within our ever-growing-enrollment CS courses. Your section leader is an outstanding alum of the CS106 courses, has undergone an entire quarter of intensive teaching training, and has completed at least one quarter of section leading in a different CS106 course before moving up to section leading for CS106B. I hope you take advantage of this opportunity to learn from these outstanding individuals and talented programmers.

How Section Works

Today, you'll spend some time getting to know your section leader and fellow sectionees. Then you'll solve one of the two section problems below. Section handouts (like this one) typically include more problems than can reasonably be expected to complete in section. The other problems are included to help you develop a bank of practice problems to use for extra practice in personal study and exam review.

You should bring a laptop to section, if possible, so you can participate in coding some of the problems alongside your section leader. Most section leaders solve some problems on the whiteboard, and some problems in "lab" style on a laptop. Sometimes I will suggest to the section leader certain approaches to take on certain problems from the handout, and other times I will leave the choice of problems and approaches up to your section leader.

1. Parameter Mystery #1. What is the console output of the following program?

```
string mystery(string a, string& b) {
    b += a[0];
    b[0] = 'X';
    a = "Cynthia";
    return a;
}

int main() {
    string a = "Mehran";
    string b = "Marty";
    b = mystery(b, a);  // What if this line were:  mystery(b, a);  ?
                        // Bonus: what if it were:  a = mystery(b, a);  ?
    cout << a << endl;
    cout << b << endl;
    return 0;
}
```

2. Even Average. *(console input/output)*
Write a complete program that prompts the user for a sequence of non-zero integers, and then prints the average of all even numbers typed (ignoring odds). Stop prompting when the user types -1. Sample output:

```
Integer? 1
Integer? 3
Integer? 2
Integer? 6
Integer? 4
Integer? 10
Integer? 9
Integer? -1
Average: 5.5
```

Bonus: Also print the entered integers (odds and evens) were greater than the average. Sample output:

```
Numbers greater than average: 6 10 9
```

3. cumulative. *(Vector)*
Write a function named **`cumulative`** that accepts a reference to a Vector of integers as a parameter and modifies it so each element contains the sum of the original vector up through that index. For example, if a `Vector` variable `v` stores `{1, 1, 2, 3, 5}`, the call of `cumulative(v);` should modify it to store `{1, 2, 4, 7, 12}`. (There are many ways to do this problem. Try to do it in one pass through the original vector.)

```
void cumulative(Vector<int>& vec) { ...
```

4. crossSum. *(Grid)*
Write a function named **`crossSum`** that accepts three parameters: a reference to a `Grid` of integers, an integer row, and an integer column, and prints the sum of all numbers that appear in the row and/or column provided. For example, if a `Grid` variable named `grid` stores the following integers, the call of `crossSum(grid, 1, 1)` should return `2+5+8+4+6` or `25`. If the row and/or column passed is out of bounds, return `0`.

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

int crossSum(Grid<int>& grid, int row, int col) { ...
```

5. splitStack. *(Stack/Queue)*
Write a function named **`splitStack`** that accepts a reference to a `Stack` of integers as its parameter, and re-orders the stack so that all the non-negative numbers are at the top and all the negative numbers are at the bottom. For example, if a `Stack` variable named `s` stores `{4, 0, -1, 5, -6, -3, 2, 7}` from bottom to top, the call of `splitStack(s);` should change `s` to store an ordering such as `{-3, -6, -1, 7, 2, 5, 0, 4}`. You may declare only a single single `Queue` as auxiliary storage, but do not declare any other data structures (e.g. arrays, grids, vectors).

```
void splitStack(Stack<int>& stack) { ...
```

6. Debugging removeDuplicates. *(Vector; debugging)*
The following is a function that is supposed to get rid of any consecutive duplicates from a `Vector`, so for example, `{1, 2, 2, 3, 4, 4, 4, 7}` should become `{1, 2, 3, 4, 7}`. However, there are **two bugs**. What are they, and how can we fix the code?

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