CS 106B Section 1 Handout (Week 2)

C++ Functions

1. **Reference Parameter Mystery 1**: What output is produced by the following code?

```
void mystery(int c, int& a, int b) {
    cout << b << " + " << c << " = " << a << endl;
    a++;
    b--;
}

int main() {
    int a = 4;
    int b = 7;
    int c = -2;

    mystery(b, a, c);
    mystery(c, b, 3);
    mystery(b, c, b + a);
    return 0;
}
```

2. **swapPairs**. Write a function named `swapPairs` that accepts a string reference as a parameter and modifies that string so that each pair of adjacent letters will be reversed. If the string has an odd number of letters, the last letter is unchanged. For example, if a string variable s stores `"example"`, the call of `swapPairs(s);` should change the string to `"xemalpe"`. If `s` had been `"hello there"`, the call would produce `"ehll ohtree"`.

Streams

3. **Even Sum/Max**. Write code to prompt the user for integers and print the total even sum and the maximum of the even numbers typed. You may assume that the user types at least one non-negative even integer.

```
how many integers? 4
next integer? 2
next integer? 9
next integer? 18
next integer? 4
even sum = 24
even max = 18
```

4. **inputStats**. Write a function named `inputStats` that accepts a `string` parameter representing a file name, then opens/reads that file's contents and prints information to the console about the file's lines. Report the length o each line, the number of lines in the file, the length of the longest line, and the average characters per line. You ma assume that the input file has at least one line of input. For example, if the input file contains the following data:

```
"Beware the Jabberwock, my son,
the jaws that bite, the claws that catch,
Beware the JubJub bird and shun
the frumious bandersnatch."
```

Your function should produce the following console output:

```
Line 1 has 31 chars
Line 2 has 41 chars
Line 3 has 31 chars
Line 4 has 27 chars
4 lines; longest = 41, average = 32.5
```

CS 106B Section 1 Handout (Week 2)

Collections: Vector, Grid

5. **Vector Mystery**: Write the final contents when the following function is passed each vector below:

```
void vectorMystery(Vector<int>& list) {
    for (int i = 0; i < list.size(); i++) {
        int n = list[i];
        if (n % 10 == 0) {
            list.remove(i);
            list += n;
        }
    }
    cout << list << endl;
}
```

6. **stretch:** Write a function named `stretch` that accepts a reference to a vector of integers as a parameter and modifies it to be twice as large, replacing every integer with a pair of integers, each half the original. If a number in the original vector is odd, then the first number in the new pair should be one higher than the second so that the sum equals the original number. For example, if a variable named `v` refers to a vector storing the values `{18, 7, 4, 24, 11}`, the call of `stretch(v)` should change `v` to contain `{9, 9, 4, 3, 2, 2, 12, 12, 6, 5}`. (The number 18 is stretched into the pair 9, 9, the number 7 is stretched into 4, 3, the number 4 is stretched into 2, 2, the number 24 is stretched into 12, 12, and 11 is stretched into 6, 5.)

7. **mirror**. Write a function `mirror` that accepts a reference to a grid of integers as a parameter and lips the grid along its diagonal, so that each index `[i][j]` contains what was previously at index `[j][i]` in the grid. You may assume the grid is square, 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
```

8. **splitStack:** Write a function named `splitStack` that takes a stack of integers as a parameter by reference and splits it into negative and non-negative numbers. Specifically, the numbers in the stack should be arranged so that a negative numbers appear on the bottom of the stack, and all non-negative numbers appear on the top.

   In other words, if you called **pop** after calling this method, you would first get all non-negative numbers and then all negative numbers. It doesn't matter what order the numbers appear in so long as all negative numbers appear below all non-negative numbers. You may use a single queue as auxiliary

9. **Numbers Above Average:** Write a function:

   `int numbersAboveAverage(string filename);`

   The first parameter, **filename**, gives the name of a file that contains a list of real numbers, one per line. The function should return the number of values in the file which are above the average value.