Chris Gregg
CS106B

Section #1
April 9, 2018

Section Handout #1: Collections, File Reading, Big-O

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

1. Mirror

Write a function `mirror` that accepts a reference to a grid of integers as a parameter and flips 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 is, 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}}
```

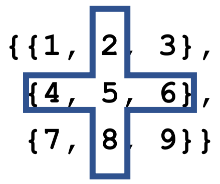
Bonus: How would you solve this problem if the grid were not square?

2. PlusSum

Write a function named `plusSum` 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 **"plus" pattern with sides of length 1 centered at that row and column**. For example, if a grid named `g` stores the following integers:

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

Then the call of `crossSum(g, 1, 1)` should return 25 (4+5+6+2+8):



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

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.

3. 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, passing the vectior `{18, 7, 4, 24, 11}` should modify the vector to contain `{9, 9, 4, 3, 2, 2, 12, 12, 6, 5}`.

4. Consecutive Duplicates

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. Stacks and Queues Mystery

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}`	______________________

6. Duplicate Elements

Write a function named **`duplicateElements`** 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 **`duplicateElements(q)`** should change it to store **`{1, 1, 2, 2, 3, 3}`**.

7. Split Stack

Write a method **`splitStack`** that takes a stack of integers as a parameter and rearranges the numbers so that all negatives appear on the bottom of the stack and all the non-negatives appear on the top. In other words, if after this method is called you were to pop numbers off the stack, you would first get all the nonnegative numbers and then get all of the negative numbers.

Example: passing **`{4, 0, -1, 5, -6, -3, 2, 7}`** could change it to **`{-3, -6, -1, 7, 2, 5, 0, 4}`** (where the left is the bottom, and the right is the top, of the stack). It does not matter what order the numbers appear in as long as all the negatives appear lower in the stack than all the non-negatives. You may use a single queue as auxiliary storage.

8. Big-O Analysis

Give a tight bound on the nearest runtime complexity for each of the following code fragments, using Big-O notation, in terms of the variable N. In other words. Find the growth rate of the code's runtime as N grows. **Hint:** a **`HashSet`**'s add operation is O(1) time, and a **`Set`**'s remove and add operations are each O(logN) time.

```
// a)
int sum = 0;
for (int i = 1; i <= N + 2; i++) {
    sum++;
}
for (int j = 1; j <= N * 2; j++) {
    sum += 5;
}
cout << sum << endl;
```

```
// b)
int sum = 0;
for (int i = 1; i <= N - 5; i++) {
    for (int j = 1; j <= N-5; j+=2) {
        sum++;
    }
}
cout << sum << endl;
```

Continued on next page…

```
// c)
int sum = N;
for (int i = 0; i < 1000000; i++) {
    for (int j = 1; j <= 1; j++) {
        sum += N;
    }
    for (int j = 1; j <= 1; j++) {
        sum += N;
    }
    for (int j = 1; j <= 1; j++) {
        sum += N;
    }
}
cout << sum << endl;
```

```
// d)
HashSet<int> set1;
for (int i = 1; i <= N; i++) {
    set1.add(1);
}

Set<int> set2;
for (int i = 1; i <= N; i++) {
    set1.remove(i);
    set2.add(i + N);
}
cout << “done!” << endl;
```

```
// e)
int sum = 0;
for (int i = 1; i <= N - 2; i++) {
    for (int j = 1; j <= i+4; j++) {
        sum++;
    }
    sum++;
}
cout << sum << endl;
```

```
// f)
int sum = 0;
for (int i = 1; i <= N*2; i++) {
    for (int j = 1; j <= i/2; j+=2) {
        for (int k = 0; i<N*N; k++) {
            sum++;
        }
    }
}
cout << sum << endl;
```

```
// g)
Vector<int> list;
for (int i = 0; i < N; i++) {
    list.insert(0, i*i);
}
Set<int> set;
for (int k : list) {
    set.add(k);
}
cout << “done!” << endl;
```

```
// h)
int sum = 0;
for (int i = 1; i <= 100000; i++) {
    for (int j = 1; j <= i; j++) {
        for (int k=1; k <= N; k++) {
            sum++;
        }
    }
}
cout << sum << endl;
```

9. Keith Numbers

A Keith Number is defined as any n-digit integer that appears in the sequence that starts off with the number’s n digits and then continues such that each subsequent number is the sum of the preceding n. All one-digit numbers are trivially Keith numbers, but there are more interesting ones as well. For example, the number 7385 is a Keith number because of the following sequence:

7, 3, 8, 5, 23, 39, 75, 142, 279, 535, 1031, 1987, 3832, **7385**

Keith numbers are computationally hard to calculate; there are only about 100 known right now. Write a function `findKeithNumbers` that takes a minimum and maximum value and finds all Keith numbers between those values (inclusive). For each number, it should print the sequence that proves it is a Keith number. For example, if you call `findKeithNumbers(1, 50)`, it should print:

```
1: {1}
2: {2}
3: {3}
4: {4}
5: {5}
6: {6}
7: {7}
8: {8}
9: {9}
14: {1, 4, 5, 9, 14}
19: {1, 9, 10, 19}
28: {2, 8, 10, 18, 28}
47: {4, 7, 11, 18, 29, 47}
```

10. Average in File
Write a function named `averageValueInFile` that reads a file and returns the average (mean) of the numbers in that file. The `filename` parameter gives the name of a file that contains a list of real numbers, one per line. You may assume that the file exists and follows the proper format.

```
double averageValueInFile(string filename) { . . . }
```

11. Name Diamond
Write a function called `nameDiamond` that accepts a string as a parameter and prints it in a "diamond" format. For example, `nameDiamond("CHRIS")` should print:

```
C
CH
CHR
CHRI
CHRIS
 HRIS
  RIS
   IS
    S
```