CS 106B Section 2 (Week 3)

This week has more practice with data structures (Stacks, Queues, [Hash]Sets, [Hash]Maps), as well as a taste of writing functions recursively. As they say, in order to understand recursion, you first need to understand recursion.

Recommended problems: #2, #4, #7
Extra practice problems: 5.13, 5.22, 7.2 (from Textbook)

---

1. reorder. *(Stacks/Queues)*
Write a function named **reorder** that takes a queue of integers that are already sorted by absolute value, and modifies it so that the integers are sorted normally. You are not allowed to declare any data structures other than a single Stack<int>.

```
void reorder(Queue<int>& queue) { ...
```

---

2. twice. *(Sets)*
Write a function named **twice** that takes a vector of integers and returns a set containing all the numbers in the vector that appear exactly twice. Bonus: do the same thing, but you are not allowed to declare any kind of data structure other than sets.

```
Set<int> twice(Vector<int>& numbers) { ...
```

---

3. unionSets. *(Sets)*
Write a function named **unionSets** that takes a set of sets of ints, and returns the union of all of the sets of ints. (A union is the combination of everything in each set.) For example, if a `Set` variable named `sets` stores the following set of integers, the call of `unionSets(sets)` should return `{1,2,3,4,5,6,7}`.

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

```
Set<int> unionSets(HashSet<Set<int> >& sets) { ...
```

---

4. reverse. *(Maps)*
Write a function named **reverse** that takes a map from ints to strings, and returns a map with the associations reversed. For example, if a `Map` variable named `map` stores `{ 1: "a", 2: "b", 3: "c" }`, the call of `reverse(map)` should return `{ "a": 1, "b": 2, "c": 3 }`. If there are duplicate values (k1, v) and (k2, v) in the original map, your returned map may contain either (v, k1) or (v, k2).

```
Map<string, int> reverse(Map<int, string>& map) { ...
```

CS 106B Section 2 (Week 3)

5. print2grams. *(Maps)*

Write a function named **print2grams** that takes a data structure containing 2-grams and a score for each one, where the first word in the 2-gram is a key in the outer Map and the second word is a key in the inner Map. First, discuss how the twoGrams map is structured. Then, print each 2-gram and its score. Printing the map with key/value pairs `{ "a": { "b": 1.0, "c": 2.0 }, "x": { "y": 3.0, "z": 4.0 } }` should yield the following output:

```
a b: 1.0
a c: 2.0
x y: 3.0
x z: 4.0
```

```
void print2grams(Map<string, Map<string, double> >& twoGrams) { ...
```

---

6. mystery. *(recursion I)*

What does the following function return if you pass in 1? 15? 314? 271828? -1414?

```
1  int mystery(int n) {
2      if (n < 0) {
3          return mystery(-n);
4      else if (n < 10) {
5          return n;
6      } else {
7          return n % 10 + mystery(n/10);
8      }
9  }
```

---

7. cannonballs. *(recursion II)*

Write a function named **cannonballs** that returns the number of cannonballs in a square pyramid of cannonballs of the given height. For example, in a square pyramid of height 3, the bottom layer has 9 balls, the middle layer has 4, and there is one ball on top, so `cannonballs(3)` returns 14. (You can assume that height won't be negative).

```
int cannonballs(int height) { ...
```

---

8. reverse. *(recursion III)*

Write a function named **reverse** that takes a string and reverses it. For example, reversing "Hello World" returns "dlroW olleH".

```
string reverse(string s) { ...
```