Week 2 Section

This week's section handout has practice with Sets, Maps, and File I/O, as well as several recursive problems. You can practice all the problems on CodeStepByStep: https://codestepbystep.com/problemset/view?id=4.

1. RecursionMysteryComma - Recursion Trace

For each call to the following recursive function, write the output that would be produced, as it would appear on the console.

```
void recursionMysteryComma(int x, int y) {
   if (y == 1) {
       cout << x;
   } else {
       cout << (x * y) << ", ";
       recursionMysteryComma(x, y - 1);
       cout << ", " << (x * y);
   }
}
```

	Output:
`recursionMysteryComma(4, 1);`	______________________
`recursionMysteryComma(4, 2);`	______________________
`recursionMysteryComma(8, 2);`	______________________
`recursionMysteryComma(4, 3);`	______________________
`recursionMysteryComma(3, 4);`	______________________

2. Rarest - Map

Write a function named **`rarest`** that accepts a reference to a `Map` from strings to strings as a parameter and returns the value that occurs least frequently in the map. If there is a tie, return the value that comes earlier in ABC order. For example, if a variable called `map` containing the following elements:

```
{"Alyssa":"Harding", "Char":"Smith", "Dan":"Smith", "Jeff":"Jones",
"Kasey":"Jones",
 "Kim":"Smith", "Morgan":"Jones", "Ryan":"Smith", "Stef":"Harding"}
```

Thanks to Marty Stepp and other CS106B and X instructors and TAs for contributing problems on this handout.

Then a call of `rarest(map)` would return `"Harding"` because that value occurs 2 times, fewer than any other. Note that we are examining the values in the map, not the keys. If the map passed is empty, throw a string exception.

---

3. biggestFamily - Map, File I/O
Write a function named **`biggestFamily`** that reads an input file of people's names and prints information about which family has the most people in it. Your function accepts a string parameter representing a filename of input.

The input file contains a collection of names, one per line, in the format of the example shown to the right. Each line of the file contains a first name (given name), a single space, and a last name (surname / family name). For example, in the name "Ned Stark", the word "Ned" is the first name and "Stark" is the last name. You may assume that every line follows this exact format and that first and last names are single words.

```
Jon Snow
Ned Stark
Gregor Clegane
Cersei Lannister
Tyrion Lannister
Jaime Lannister
Catelyn Stark
Theon Greyjoy
Arya Stark
Cersei Smith
```

Your function should open and read the contents of this input file and figure out which last name(s) occur most frequently in the data, and print the members of that family in ABC order in exactly the format shown below. If multiple families are tied for the most members, print each of the tied families in the same format.

For example, if the input above is in `families.txt`, then the call of `biggestFamily("families.txt");` should print:

```
Lannister family: Cersei Jaime Tyrion
Stark family: Arya Catelyn Ned
```

If the file is missing or unreadable, your function should throw a string exception. If the file exists, you may assume that it contains at least one name, that every line of input in the file is in the exact valid format described above, and that no two lines of the file will be exactly the same (though a given first or last name might occur multiple times).

---

4. isHappyNumber - Set

Write a function named **`isHappyNumber`** that returns whether a given integer is "happy". An integer is "happy" if repeatedly summing the squares of its digits eventually leads to the number 1.
For example, 139 is happy because:

- $1^2 + 3^2 + 9^2 = 91$
- $9^2 + 1^2 = 82$
- $8^2 + 2^2 = 68$
- $6^2 + 8^2 = 100$
- $1^2 + 0^2 + 0^2 = 1$

By contrast, 4 is not happy because:

- $4^2 = 16$
- $1^2 + 6^2 = 37$
- $3^2 + 7^2 = 58$
- $5^2 + 8^2 = 89$
- $8^2 + 9^2 = 145$
- $1^2 + 4^2 + 5^2 = 42$
- $4^2 + 2^2 = 20$
- $2^2 + 0^2 = 4$
- ...

5. stutterStack - Recursion

Write a recursive function named **`stutterStack`** that accepts a `Stack` of integers as a parameter and replaces every value in the stack with two occurrences of that value. For example, suppose a stack named `s` stores these values, from bottom => top:

```
{13, 27, 1, -4, 0, 9}
```

Then the call of `stutterstack(s);` should change the stack to store the following values:

```
{13, 13, 27, 27, 1, 1, -4, -4, 0, 0, 9, 9}
```

Notice that you must preserve the original order. In the original stack the 9 was at the top and would have been popped first. In the new stack the two 9s would be the first values popped from the stack. If the original stack is empty, the result should be empty as well.

6. reverseLines - Recursion, File I/O

Write a recursive function named **`reverseLines`** that accepts as its parameter a reference to a file input stream (`ifstream`) and prints the lines of that file in reverse order. For example, if an input file named `poem.txt` contains the following text:

```
Roses are red,
Violets are blue.
All my base
Are belong to you.
```

Then the call of `reverseLines("poem.txt");` should produce the following console output:

```
Are belong to you.
All my base
Violets are blue.
Roses are red,
```

You may assume that the input file exists and is readable.

7. evaluateMathExpression - Recursion

Write a recursive function named **`evaluateMathExpression`** that accepts a string parameter representing a math expression on integers and returns the result of that math expression. The expression will consist of single-digit integers and possible operators. All operators will be surrounded by parentheses; we would say that the expression is fully parenthesized. The operators will be either `+` or `*`. For example, the call of `evaluateMathExpression("(((1+2)*(3+1))+(1*(2+2)))")` should return `16`. You may assume that the string is non-empty and does not contain any other characters.