CSCI-1200 Data Structures — Spring 2017
Lab 12 — Superhero Friends

This lab is a fun (& silly) departure from our focus on the practical use and implementation of complex STL data structures and iterators. You will create two new classes that represent individual superheroes (`Superhero`) and teams of superheroes (`Team`). We provide sample code in the `main.cpp` file demonstrating how these classes should behave and interact. This lab will give you practice overloading operators.

http://www.cs.rpi.edu/academics/courses/spring17/ds/labs/12_operators/main.cpp
http://www.cs.rpi.edu/academics/courses/spring17/ds/labs/12_operators/team.h
http://www.cs.rpi.edu/academics/courses/spring17/ds/labs/12_operators/team.cpp

Checkpoint 1

First create the `Superhero` class that stores the hero's name, their true identity, and their superhuman power, all as strings. You should write read-only accessor functions to get the hero's name and power. However, it is important that each superhero's true identity remain a secret, so *it must not be accessible through the public interface*. The only way to discover a superhero's true identity is to correctly guess their true identity by using the `operator==` and `operator!=` functions. See the examples in the `main.cpp` file. Complete the necessary implementation so that the code compiles and runs successfully (with no assertion failures).

To complete this checkpoint: Show one of the TAs your implementation.

Checkpoint 2

In the sample code we use the unary `operator-` to negate (a.k.a. corrupt) a superhero. Superheroes are initially good, but turn to evil if corrupted. Likewise, the operation can be applied in reverse to turn an evil *supervillain* into a good superhero. Another fun example is `operator>` that can be used to settle debates about which hero's superpower is greater. Our test cases do not fully specify the differences in greatness for all powers. Your task is to define and implement a ranking system for the remaining powers. However, it is important to note that this property is not necessarily *transitive*; that is, if `a > b` and `b > c`, it *does not necessarily hold that* `a > c`.

To complete this checkpoint: Show a TA these additions and the test output.