Programming Assignment #3
CS 202 Programming Systems

*****This program is about operator overloading in inheritance hierarchies. *****

Remember our goals:

This term, the key idea is to break down the problem outlined (below) into small pieces and assign those responsibilities to individual classes. For each assignment, your job will be to create an OO design and program that shows how Object Oriented constructs could be used to solve the problem. You will want to focus on how to design classes that are well structured, efficient, that work together, and where each class has a specific **"job"**. This time you are adding operator overloading as the new syntax for your solution!

Operator overloading is best implemented when we create new abstract data types. So, look back at your first two programs. Did you tie the data structures into the midst of your OO designs? Did the designs really become all about the data structure and not so much about the problem at hand? This time, we want to create our own abstract data types implemented with a full set of operators and have them used by our OO programs.

Program #3 - The Abstract Data Type

We just experienced Halloween 2020. Monsters have now been let loose around Portland. Luckily, monsters are only active for certain periods of time. For example, a werewolf becomes a wolf on a full moon but a vampire only comes out at night. Different types of monsters may be friends or enemies depending on their state. So, werewolves and vampires are enemies. But vampires and bats are friends. Other properties could involve their lifelines as well as attack and defense capabilities.

For this assignment, you will be creating four breeds of your own monsters. You will need to specify when they are active, how they relate to other monsters and their special abilities. For example, what if Boris (the spider) came to life (from our intro video Thursday…). Push up the common elements and derive the differences. To make it fun, we will have monster bashes where they will battle each other.

You will need two different types of data structures for this program:

1. **Support the monsters available to battle:** have a Binary Search Tree (BST) of all of the monsters available. **You have an option to implement a BALANCED Tree (of your choice) for extra credit**. (If you do not choose to implement a balanced tree for this assignment, then a balanced tree will be required in your last programming assignment).
2. **Support for the list of available attacks and defenses:** Select an array of doubly linked lists. _

The primary purpose is to support operators with your classes: =, +, +=, ==, !=, the relational operators, and the ability to input/output data. Think about how these operators might apply to your classes. Also think about how you will break your program down into smaller components.

- For the operators, think about how each operator will work. Will you use the + and += to add a new set of armor to the available defenses? Certainly << will allow you to display the list of monsters available. As you decide how to apply the operators, make sure to stay within the rules of how the operators are expected to behave. You may find that some operators don't apply at all (and therefore shouldn't be implemented).
- Don't forget your copy constructor!
- Yes, you CAN now write your own STRING data type, but it can't be the only place you use operator overloading!.
- For each of your operators make sure to fully test them.

Questions to ask…about operator overloading

When using operator overloading, remember to ask yourself the following questions:

a) What should be the residual value (and what data type is this)?

b) Should it be a modifiable lvalue or an rvalue? Lvalues are returned by reference, most rvalues are returned by value.

c) What are the data types of the operator's operands? Remember to pass as a constant reference whenever possible.

d) Is the first operand always an object of class? If so, then it should be a member function. If not, then it should be a friend function.

e) Can the operator be used with constant operands? If the first operand can be a constant, and IF it is a member function, then it should be a constant member function.