In the multiset program, I created a recursive Max, Min, Insert, Remove function in order to make the later implementations of said functions possible. Because they did not request a node but only a key, a private function that took an input of a node was required in order to traverse the tree and find the correct node. In addition, I ended up overloading the << operator for both Nodes and the BST to allow the prime_factors function to print out every node in numerical order.

In order to test the functionality of the multiset class, I added two tests to the initial tests testing the remaining functions that were not initially tested. This includes the Ceiling, Contains, and Floor functions. In addition, the evaluate what would happen if a command such as Remove, Floor, Ceiling, Max, or Min were used in unideal conditions, such as empty sets, or when the inputs are invalid.

In the prime_factors program, I chose to implement a function that works in O(n) in order to determine if the number is a prime function. I also wrote a similar recursive program that returned one factor of the number, running itself again with the new number divided by its factor. It continues to do this until a prime number is left at which point it puts itself into the tree. The function is utilized in almost all of the different commands, allowing for a min and max to be determined, as well as a key to be found. Utilizing the overloaded << operator, the tree can also be printed out, providing the keys in numerical order as well as their quantities. By running the respective commands for each input (Ceiling for near +prime, Floor for near -prime, Contains for near prime, Min for min, Max for max, and << for all), the program is able to generate the correct answer. Throughout the program however, I use stringstreams in order to test if the inputs are correct and to allow for the correct variable types based on inputs.