Sanjana Aithal
ID: 917496419

Programming Homework 3: Report

The first program was implementing a multiset as a binary search tree, and modifying the typical API given in order to allow for duplicate keys, a defining feature of a multiset. In order to allow for duplicate keys, I modified the API provided in the Lecture notes so that the struct Node includes an additional component—the internal counter variable (count). Therefore, when performing Insertions or Removals, if a node of a particular key already existed within the bst, all I did was increment/decrement the count variable to reflect the change in size of the bst.

Moreover, within the implementation, I used internal helpers for Min, Insert, and Remove to either allow for a recursive process to occur or to provide limited accessibility to a user utilizing the public API. I implemented the Find method as a private method to allow for abstraction within the class—the Contains, Count, Floor, and Ceil methods all use Find to find the appropriate node in their implementation.

When throwing exceptions, I utilized the out of range exception for the "Empty Multiset" exception. This is because attempting to access items of an empty multiset is out of the multiset's range. Moreover, I used the invalid argument exception for the "Invalid Key" exception. This is because in most cases, the key did not exist within the bst or was either too high or too low to be considered an appropriate argument for the Floor and Ceil functions.

The second program was implementing the multiset.h header file to create a working prime factors calculator. While checking the command line arguments, I used the number of arguments as a first indicator of the type of methods that can be called, and then either used the std::string compare function or indexed the string to determine which function should be called. I used a series of if-else statements to catch the appropriate errors and outputted all to std::cerr. Within the more complex functions of Ceil and Floor, I used a while loop to remove any instances of the inputted key in the bst before calling the appropriate methods to allow for the correct inputs. In the all method, in which I print out all the prime factors, I used the Min() and Ceil() methods to mimic an in-order traversal. Every time you add 1 to the value of n, finding the Ceil() of that will take you to the next numerical value in the bst.

In the unit testing, I dedicated one test focusing on one particular method. I tried to throw all possible exceptions and made tests to see if the method would deal with the exception properly. Thus, there are 10 tests dedicated solely to testing the functions of the method, with the remaining tests observing whether the multiset behaves properly when inserting and deleting keys in a randomized order. While the tests are relatively short, they are designed to test the different functions of multiset—holding duplicate keys.