treemap.h:

In this program, I used the implications given by the typical binary breach tree during the class and made changes on them to make this treemap. The insert, remove, find, and get functions are quite similar with before, but not the finding ceil and finding floor functions. For these two functions, I added the same name functions but receive more parameters like node's pointer in my private part. Because only in this way I can make the effective operate and follow the complexity restriction. Typically, for those functions, I added one more parameter, the start node pointer. Just like what professor did in the print function in binary tree. As a result, I can make the function work nicely by only accepting only the key parameter.

For the testing part, I added 7 more test cases to test my functions separately. To test my exceptions, I first create one empty treemap and use those functions which need a none empty tree. And then I test the segmentations fault and the insert duplicate key error. Finally, I test the out of bond values when passing under floor numbers or over ceil numbers. And in the other test case, I do the normal job and test the correctness of each functions by using google test.

eff_donations:

In this program, I used the treemap I have just created to solve the problem, and transform the command to their corresponding operations from the treemap header file. For the input and scan form the file. I use the get line function and made the comma as the separation of name and amount of money they donate. And for the new line character, I used the .get() function after scanning the first line from the file to dismiss the scanning of that character in to the names and cause there is a '\n' in front of every names besides the first one. And for the plus or minus sign which need to be definite in the "who" command, I used the argv[3][0] to see if the sign is plus or minus or else(find value).