**Code Instructions**:

**Compile-**

To compile the code, run these commands in bash, with the current directory set to the folder in which the files reside.

Here are the compile commands for my serial implementation:

g++ -O3 -w lcovingtAssignment2Serial.cpp -lpthread -o lcovingtAssignment2Serial

Here are the compile commands for my multithreaded implementation:

g++ -O3 -w lcovingtAssignment2.cpp -lpthread -o lcovingtAssignment2

IE the terminal will look something like this

user@computer:~/folder/Assignment2$ g++ -O3 -w lcovingtAssignment2.cpp -lpthread -o lcovingtAssignment2

**Run-**

Next, to run the code utilize the following format for the serial program:

./lcovingtAssignment2Serial filePath

For the multithreaded code, utilize the following format:

./lcovingtAssignment2 filePath numberOfThreads

**Example Commands -**

Here are a couple example commands for ease of use:

./lcovingtAssignment2Serial networkDatasets/HcNetwork.txt

./lcovingtAssignment2 networkDatasets/graphV15000.txt 8

**How the code works:**

The code starts by almost identically the same as assignment 1. It initializes a two dimensional vector to store all the friendships of the graph. Next we initialize our threads. The work they will be doing is as follows. They will run a for loop executing a chunk of all of the nodes individual clustering coefficients. In an example of 6 nodes and 2 threads. The first thread will call calcAndSaveIndividualClustering on its designated nodes 1, 2, and 3. The second thread will do the same for nodes 4, 5, and 6. The way calcAndSaveIndividualClustering works is it takes a node as an argument. It then creates a list of all of its neighbors. From knowing how many neighbors we know the nodes degree. The next value for our individual clustering coefficient is how many connections the node of interest’s friends share with eachother. We iterate over all of the friends and compare them with eachother to see their friendships, and intelligently add them and store them as our m value. If m is equal to zero, our thread exits immediately as it will not have anything to add to our global sum of coefficients. Otherwise, once we have the values necessary to calculate our clustering coefficient we carry it out. The last operation to run is the method addToGlobalSum with a mutex implemented. The parameter passed is the individual cluster coefficient value that needs to be added to the global sum. To add this individual clustering coefficient to the global sum the mutex must be available. If it is not, the thread will be blocked until another thread is finished adding their individual clustering coefficient. This ensures that no individual clustering coefficient will be lost due to problems that can arise when threads are reading and writing to the same resource. Once all threads have finished their work, they all are joined in the main thread. Lastly the global clustering coefficient is calculated in the main thread, and then printed to the terminal.