1. I was able to finish the project.
2. No bugs detected.
3. No banned STL items used!

**TESTING**

User Class

Created several *User* objects in the *main* Class with different names, email addresses, and vectors of watch histories. I then used the *assert()* method to verify their data.

Movie Class

Similar to the *User* Class, I created several *Movie* objects and used *assert* to verify their data.

UserDatabase Class

I first ensured that two, identical, files could not be loaded into one *UserDatabase* object (created in the *Main* class). I then limited my *users.txt* file to one user’s information with only a couple of valid movies. Line by line, I tested if the data read by the program was the correct data from the *users.txt* file. After the data was read and a new *User* was created, I called several *User* functions on this object (i.e. *.get\_email)* to ensure the *User* object was created correctly. My tests on the *treemm* object created in this class are listed below.

MovieDatabase Class

Similar to the *UserDatabase* Class, I ensure that two, identical files could not be loaded into the object. I also limited my *movies.txt* file to one movie’s information and, as the *load* function read the text file’s data, I printed each line. I ensured that the string of directors, actors, and genres was split correctly into vectors using my own *split\_str* function and also called several *Movie* functions on the new *Movie* object created. Each *Movie*, along with its respective actor(s), director(s), and genre(s) was passed into a *treemm* object, which is described below.

Recommender Class

I again limited my *users.txt* and *movies.txt* files to one user and a couple of “similar" movies. As I gathered my *User’s* information, I ensured that it was correct by simply editing the function headers to return that information. Then, I repeated the following as well:

- For ALL the movies that the user had watched (which I initially limited to 3), I printed ALL of their directors, actors, and genres.

- For EACH of the movies associated with EACH director, actor, and genre, I printed their IDs.

- After adding EACH movie from above with their scores attached, print their scores

- For EACH duplicated movie (movies recommended that the *User* has already watched), print their IDs and delete them

TreeMultiMap Class

I first mapped several pairs of *strings* to *ints*, ensuring that there were duplicate *strings* that would have mapped to a vector of corresponding *ints*. I then tested this with *Movie* IDs and *Movie* pointers, ensuring that the same ID was not inserted twice and that I could call *Movie* functions (i.e. *.get\_title*) to the values returned by the *Iterator*.