**Project in Computer-Aided Verification**

**Tom Shech – 205482771**

In my project I decided to implement the application side. The idea was “mini Facebook” when each user can post posts, and have comments, but I also added a dimension of “ranking” to have the most-commented post and best users. I decided to verify the logical features of the application.

I tried to use all the aspects we saw in class and more.

**Verifications I did:**

1. When getting the user’s credential as an input, I firstly safely allocated a space for the new user. Then I checked it has a suffix of '\0' or it doesn’t overflow the maximal number of characters I allowed. Then I calculated the real *length* of the string and copied only the first *length* characters so no buffer overflow can be executed.
2. I used an abstract set like we saw in class. It means that I used a set of ghosts in order to use that set. To verify a correct behavior of the set I checked that insertion and deletion of set properly works. Firstly, I had to check that the key I wanted to insert is really one of the ghosts, then I had to remember this key (all those choices were done in a non-deterministic way to assure that every possible run will meet the assertions) and then I had to check that after all inserts the ghost I remember is still there – means it was really inserted and nothing has been override it.
3. Another thing I verified is a correct behavior of sorting a set. I wanted to add the option of getting the best (most commented) posts in the system, so I had to sort the set, in order to supply the data. Since just sorting the “abstract” set is not good enough (since all the info is already in the set, the question is whether the data was really inserted), I had to work around it – I sorted the set normally but then I took only the first elements that their field *has* was equal to 1. then I checked that every two elements that are really in the set, if *i>j*, then

*comments[i] >= comments[j]*.

1. In addition, I verified that no override can be executed on the users’ list. Non-deterministically, I chose to remember one of the users I inserted (key and address). Then, I checked that every other user in the list doesn’t overlap the remembered user, and that the unique user has the same address I saved.
2. I tried to imitate a situation that commonly exists in every social media – existence of subsets. For example, someone’s friend in Facebook is a subset of all users in the media. This set must be a subset of all the users, which means that every friend in the list must be a legal user of Facebook. I used the suggestion you gave in class, to assume that I have the list of all users – as I got it from database. Then I assume that no one of them is null, and non-deterministically chose some of them to get into a subset of ranked users (like creating someone’s friends list).

Then I chose one of the users in the subset and checked whether it exists in the big set.

**Difficulties**

1. First of all, I had a difficulty to understand the new ideas we saw in the course (ghosts, non-deterministic choice and abstract representation). After a lot of practice, I finally got the point and the and find the root cause for the counter examples generated by the tool.
2. I had an issue with the tool that run endlessly. Only after adding some assumptions, I finally got a result.
3. It took me a while to make the example of the ghosts set working. It kept finding a counter example even when I was sure there is nothing with my assertion. After debugging the counter examples, I found the “correctness bugs” I had – like more than one appearance of the same ghost. I added conditions and constrains to avoid those un-desired and it finally worked.

**Notes:**

I wrote it also in the code, but I focused on verifying logical correctness and I assumed that I get the application’s info from DB (part of the time with assumptions as you showed in class). The deletion I had at the end was just to assure no memory leaks.