1. Each team member (without consulting with other team members) will create their own SRS document containing 15 functional requirements and 3 non-functional requirements.

2. Submit these documents WITHOUT NAMES to the scrum master assigned to this sprint.

3. The scrum master will divide the team into two groups and assign half of the documents to each section.

4. Each sub-team will discuss the documents they've been given and "merge" them into one document. (The merged documents should still have at least 15 functional requirements and 3 non-functional requirements in the end but can have more if you find the submitted requirements distinct and valuable enough apart from each other. Don't just copy and paste one into another, though -- you'll need to discuss how to add/combine/rewrite the contents of both files to create the new one so it is internally consistent without repeats, redundancy, or contradiction between all the requirements.)

5. Each sub-team submits its merged document back to the scrum master.

6. As a team, meet and examine the resulting documents, and create one final SRS document based on your group's discussion and feelings about the results of the previous exercises. (The final document should have at least the same 15/3 content as before after any merging/rewrites, but it can have more. As before, make sure your final document is consistent after the merge with no repeats, redundancy, or contradictions)

7. Submit the final document and the 6-7 working documents created in the earlier steps. (One per team member, plus one per subgroup, in addition to the final document)

1. User will use a filepicker pop-up in order to select the file they would like to run in the program.
2. Program will prompt a user to input a word when read is ran and store that input into a location in memory
3. Program will output to users screen the word stored at a specified location in memory when Write operation is called.
4. Program will take a word stored in a specified place in memory and load the accumulator with that word when the Load operation is called.
5. Program will take a word stored in the accumulator and store that word into a specified place in memory when the Store operation is called.
6. Program will take a word stored in a specified place in memory and add that word to the accumulator leaving the result in the accumulator when the Add operation is called.
7. Program will take a word stored in a specified place in memory and subtract that word from the accumulator leaving the result in the accumulator when the Subtract operation is called.
8. Program will take a word stored in a specified place in memory and divide the accumulator by that word. It will then store that result in the accumulator when the Divide operation is called.
9. Program will take a word stored in a specified place in memory and multiply it by the word in the accumulator. Result of the multiplication will than be stored into the accumulator when the Multiply operation is called.
10. Program will branch to a specified place in memory when Branch operation is called.
11. Program will branch to a specified place in memory only when the accumulator value is a negative number when the BranchNeg operation is called.
12. Program will branch to a specified place in memory only when the accumulator’s value is zero when the BranchZero operation is called.
13. Program will exit when the Halt operation is called.
14. Program will have a GUI interface that will have two options for user to input a word when prompted. Either using a visual 10-key pad and mouse or just tying the word with their keyboard into a input box.
15. Program will output any errors during the run of the program to the users GUI so that they can be aware of what failed.

Non-functional requirements:

1. GUI will be color-blind friendly to make sure all users have a good experience.
2. If an improper input is entered, the application will display an error in red text.
3. The Program will function on both Windows and Apple operating systems.