**Individual Report, Self and Team Evaluation**

1. **Name:** Jacob Hillebrand
2. **Team Names:** Jacob Hillebrand, Jack Hildebrandt
3. **Date Period:** From: 10/01/2019 To: 10/07/2019
4. **Project or Assignment Name:** Assignment 3 PART 2 (mini-Shell)
5. **Summary of Activities:**

|  |  |  |
| --- | --- | --- |
| **Task/Event/Job/Action** | **Date** | **Time** |
| Group meeting 1 | 09/27/2019 | 11:30 P.M to 3:00 P.M |
| Requirements, Methods and Roles definition | 09/27/2019 | 11:30 P.M to 12:30 P.M |
| Worked on 1 (individually or in group) | 09/27/2019 | 12:30 P.M to 3:00 A.M |
| Group meeting 2 | 09/28/2019 | 1:00 P.M to 5:30 P.M |
| Worked on 1 (individually or in group) | 09/28/2019 | 1:00 P.M. to 4:30 P.M. |
| Tested my implementation (unit testing) | 09/28/2019 | 4:30 P.M to 5:00 P.M |
| Testing Part 1 | 09/28/2019 | 5:00 P.M to 5:30 P.M |
| Submission Part 1 | 10/01/2019 | 10:30 A.M. |
| Group Meeting 2 | 10/06/2019 | 11:00 A.M. to 11:30 A.M. |
| Part 2 Planning | 10/06/2019 | 11:00 A.M. to 11:30 A.M. |
| Work on Section 4 (Jacob) | 10/06/2019 | 11:30 A.M. to 4:00 P.M. |
| Work on Section 5 (Jack) | 10/06/2019 | 11:30 A.M. to 2:00 P.M |
| Work on Section 5 (Jack) | 10/06/2019 | 2:30 P.M. to 8:30 P.M. |
| Testing | 10/07/2019 | 9:00 A.M. to 9:30 A.M. |
| Total hours |  | 19 hours |

**Requirements & Analysis Phase:**

When looking at the design requirements for part 1, we wanted to split the program up into objects for key pieces of functionality. We eventually decided it would be best to split the first part of the program up into two classes; a logger class and a commandspawner class. The logger object would be created upon shell creation and would store information about the shell history. The commandspawner object would be created in the main() function, and would handle calling the logger methods and calling other necessary methods to run programs.

When looking at the design requirements for part 2, we knew that we would need to deal with redirection and piping, but had slightly different ideas about how to approach this task. As such, we chose to split up the requirements (Jacob took #4, and Jack took #5), and made our approaches fit modularly into our existing code to ensure we could work separately and not interfere with each other.

# Design Phase:

During the design phase, we began deciding how to create our various classes. I (Jacob) determined that the logger class would have a constructor that would load a vector with history items from a file, a method to add a history item to both the vector and the file, a method to retrieve the last history item, and a method to print all history items. Jack determined that the commandspawner class would have a constructor that would get the user’s current working directory, a method to grab a command from user input and run it, a method to tokenize an input, a method to print the terminal history, and a method to attempt to run additional commands.

To improve the design for part 2, we added on to the CommandSpawner class. We each looked at the different methods we would need to accomplish our tasks, and added them to the header file accordingly. We were then able to set about implementing these methods without disrupting the already existing working code.

**Implementation Phase:**

During implementation phase, we implemented the design described above. I was successfully able to create my history class, and Jack was successfully able to create his commandspawner class. We were then both successfully able to use both classes as objects in our main program and get a successful “shell” run to complete the part 1 objectives. Jack even had time to make the working directory output colored :).

To implement the second part, we provided implementations of new functions in the CommandSpawner.cpp file. I implemented methods to check for a redirection character, and a method to handle the command if it did involve redirection. Jack implemented a new struct within the CommandSpawner class to handle piping, and modified the existing run method to accommodate both my changes and his.

**\*\*\*\*\*\*Challenges**

1: Tokenizing proved to be a bit tricky, despite our prior knowledge of it.

2: I attempted to use the dup2 command to utilize cout and cin for writing to the history file (unsuccessfully).

3: Character pointers: Jack insisted on using a double pointer, and I faced a few challenges working with them to accomplish my goals

4: File I/O: The fnctl.h used for file I/O in Unix does not function as cleanly as the iostream file I/O. As such, I ended up having to use iostream for file creation, and fnctl for file handling.

\*\*\*\*\*\*\*Things learned

I learned a lot about using c++ during this part of the project. I learned more about the limitations of passing variables to children processes in an object, and also learned about using dup and dup2 a lot more. And finally, I have finally figured out how to program without “using namespace std” :).

With part 2, I learned new things about specific c++ libraries that I wouldn’t have otherwise known. For example, I am now much more comfortable with fnctl and iostream, and have a better grasp on how to use both. Also, I learned about double pointers from Jack, and enjoyed their usability in our program.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Teammate Evaluations (Confidential)\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Member 1: Jack Hildebrandt

* Assigned Tasks/Work: Implementing Piping
* Strength: Accomplishing an implementation in an elegant way
* Weaknesses: Enthusiasm for the extreme
* Recommendation: Don’t listen to me, and keep on coding the way you like!

Member 2: Jacob Hillebrand

* Assigned Tasks/Work: Implementing Redirection
* Strength: Successful feature implementation in a quick and clean way
* Weaknesses: Background knowledge of useful c++ libraries is lacking
* Recommendation: Practice using c++ more to better understand the language