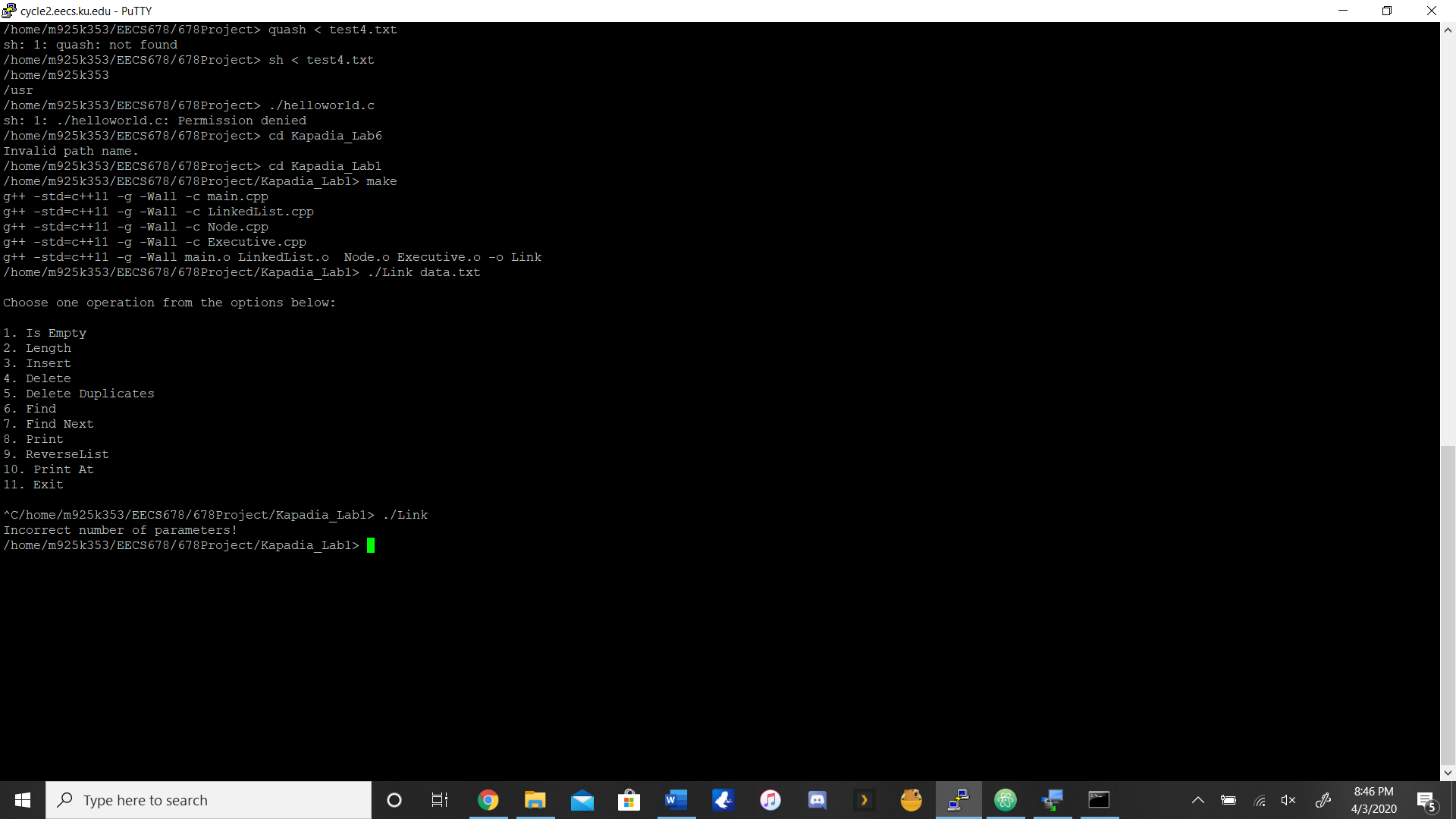
**Quash Lab Report**

**Apurva Rai and Meet Kapadia**

Below you will find how we tested for everything that is required for the quash. For us it seems to be stable and working for the most part. We included screenshots so you can see the exact commands that are needed to be typed to achieve this. To run quash itself first, type in make in the terminal and then type in “./quash” in the terminal. Some warnings will show up but they do not matter as the program still compiles and runs fine. What to do once you’re in the program can be found below in the screenshots and within the texts. It is crucial that you follow all the commands in the screenshots exactly as they appear or they might not work. Read “grading policy #10” for a bug before you start testing.

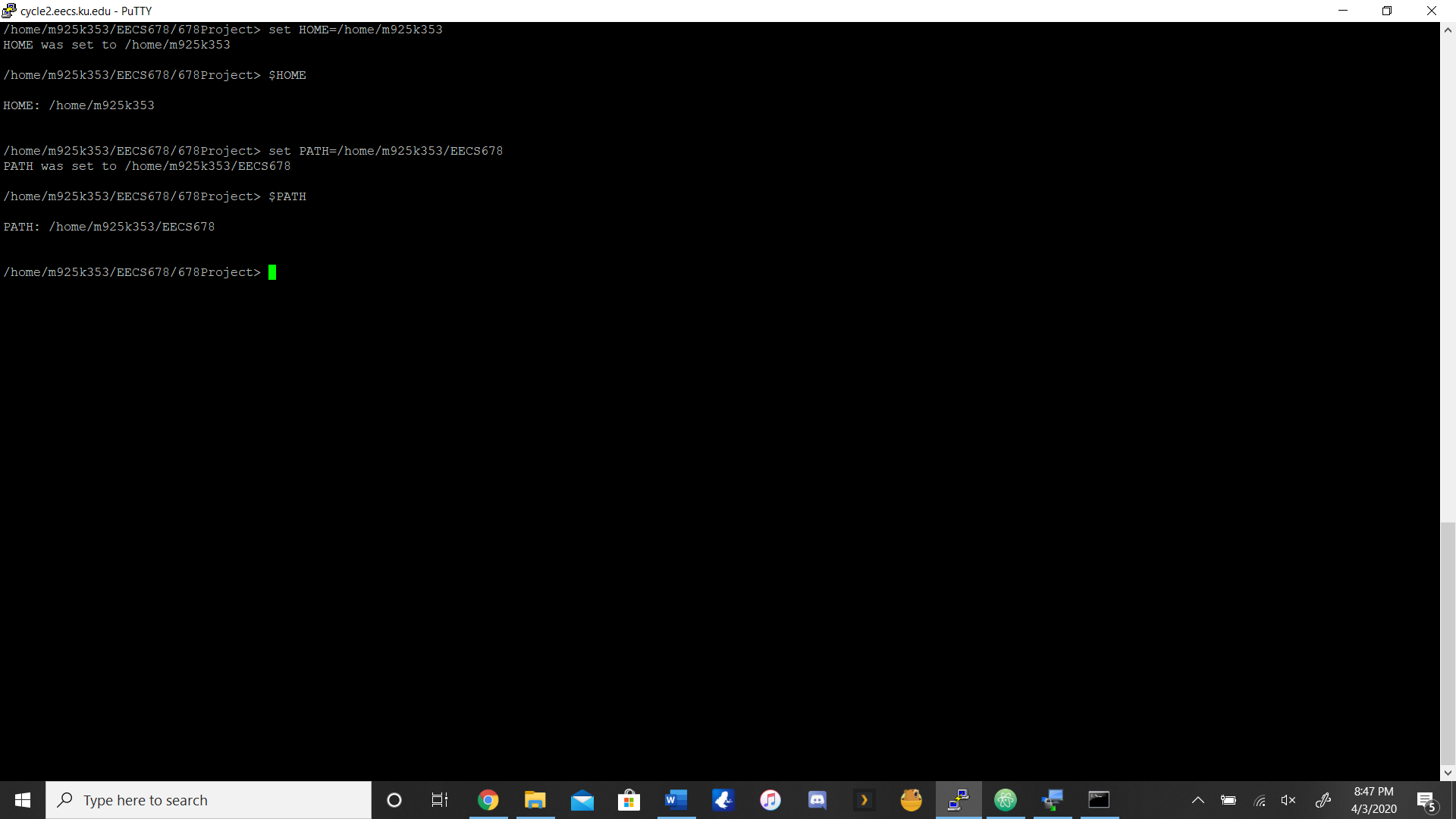
**Grading Policy #1 and #2 (Run executables with or without arguments):**

This program allows executables to run with or without any arguments. As you can see in the picture below, once ran, the quash can take in any executable file and run it. In the below code we included one of Meet’s EECS 560 labs and as you can see, you can cd over to it, make it, and the run it by using ./Link or ./Link data.txt. This means the executables are ran with out with arguments. ./Link doesn’t work in this case because data.txt is required by the EECS 560 program itself. We were able to achieve this because we just passed it through the system using if and else statements.



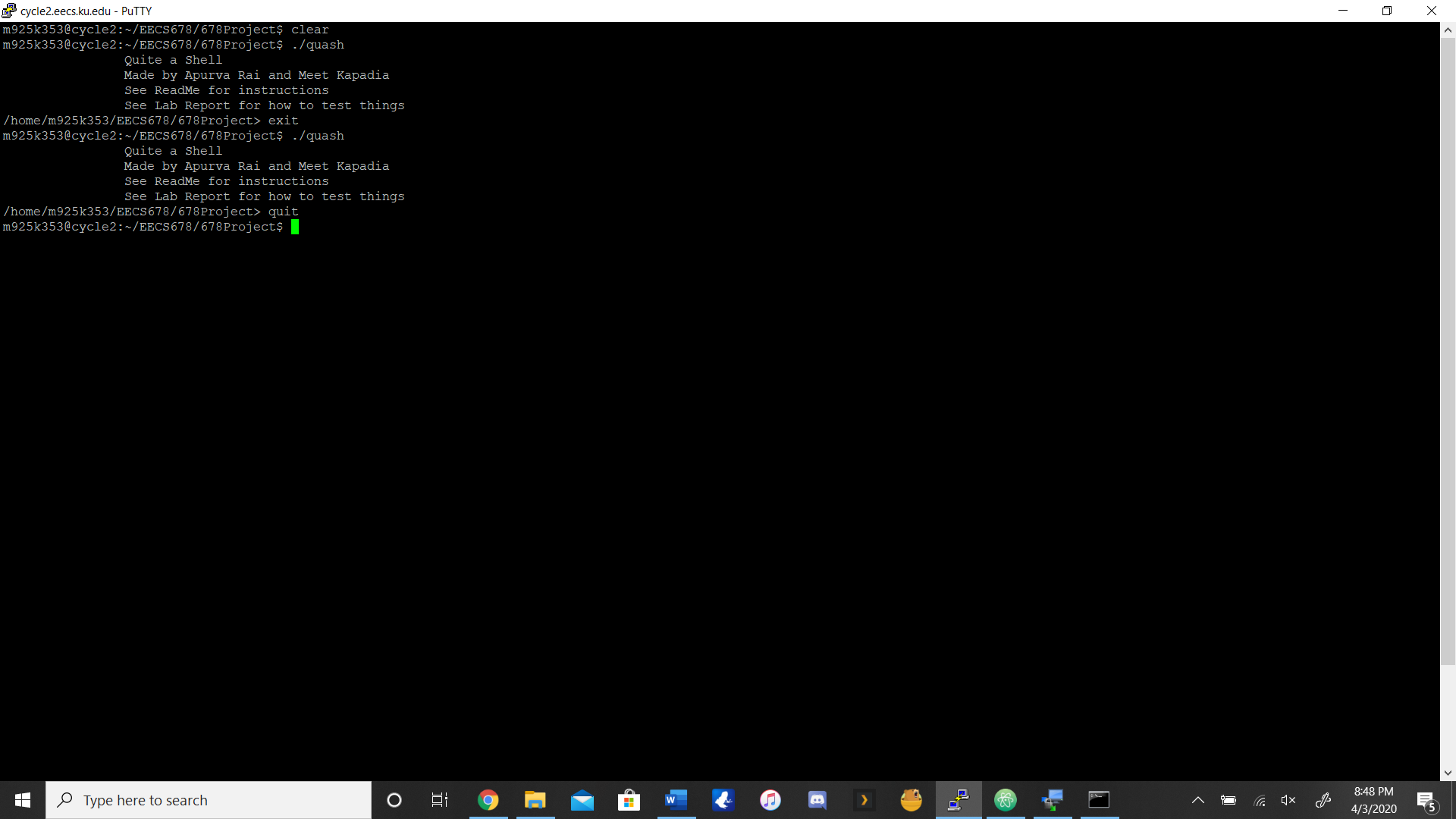
**Grading Policy #3 and #6 (set HOME and set PATH. PATH runs properly and throws error):**

As you can see in the picture below, the set home and set path are executed that way and they work. We achieved this by using getenv and setenv functions for both path and home. Setenv and getenv are inlcuded in the stdlib.h library. Below you can also see how the path works properly as well and if the file isn’t found an error message will be thorwn. The error isn’t shown in the picture but you can try to set path to something random and it will not work.



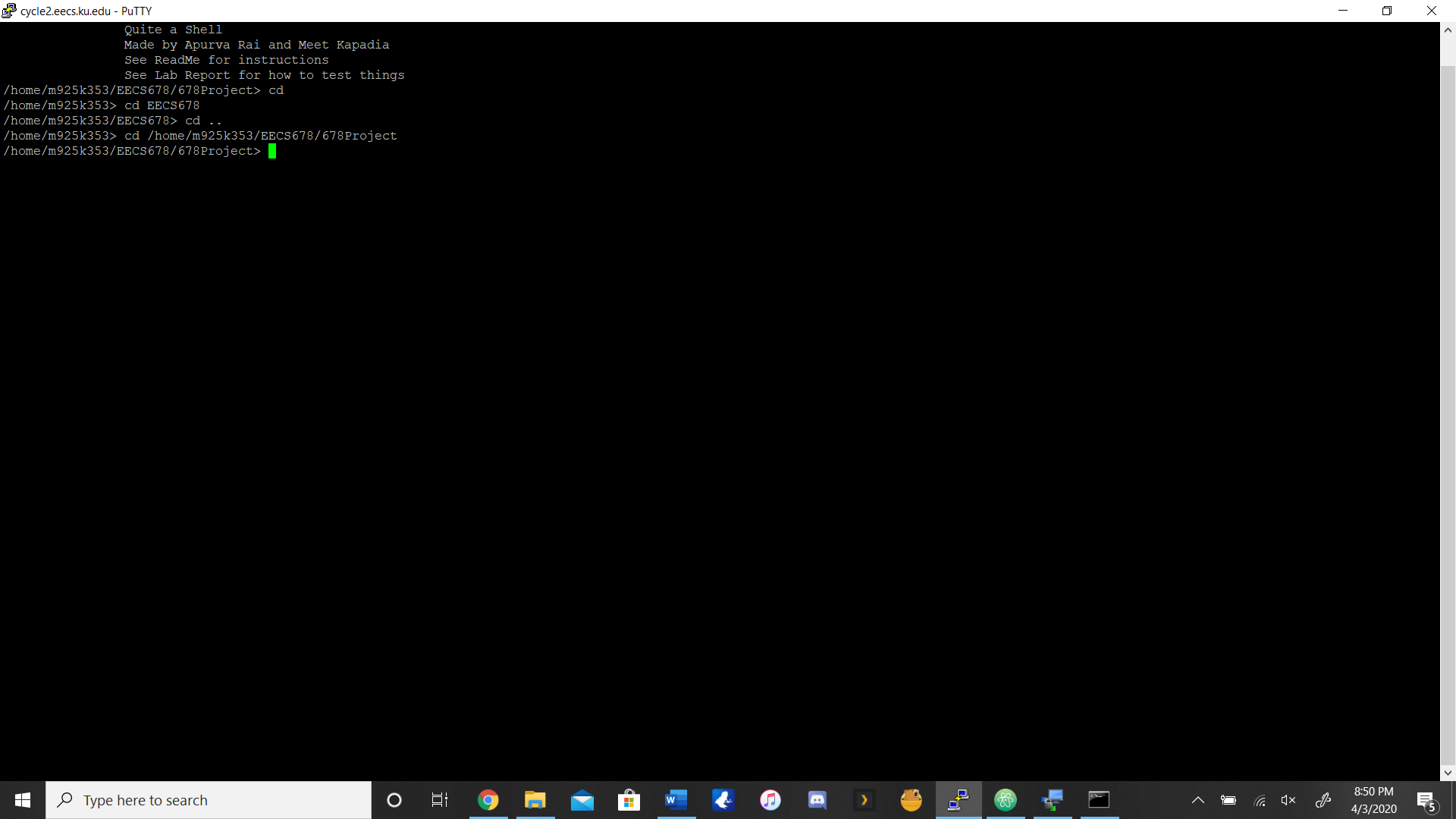
**Grading Policy #4 (exit and quit work properly):**

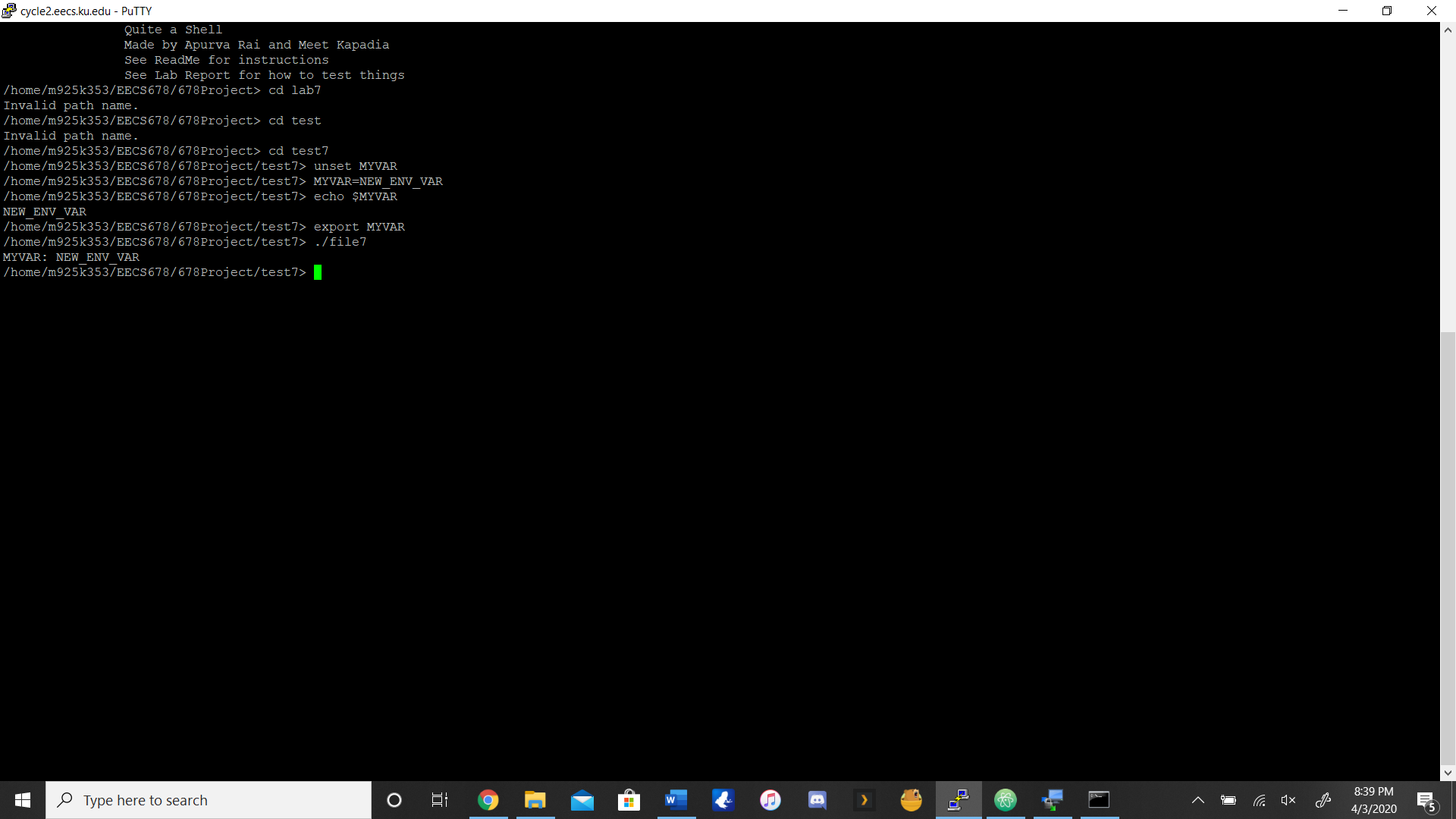
As you can see in the picture below, If you just type quit or exit within the quash, it will exit the program completely. We acheieved this by comparing the input buffer to the string “quit” or “exit” and it breaks the loop if either are true.



**Grading Policy #5 (cd [with and without arguments] works properly):**

Below in the picture you will find all the possible things you can do with cd with quash (cd, cd <path>, cd ..). We were albe to get cd to work by making a cd function. Within the function, we first shortented the string to remove the \n, then we compared whether the command was “cd” with no parameters (if so the directory changed to the home directory), else the buffer is updated so that pointer points to the file address (if so the directory will be changed to the appropriate address).

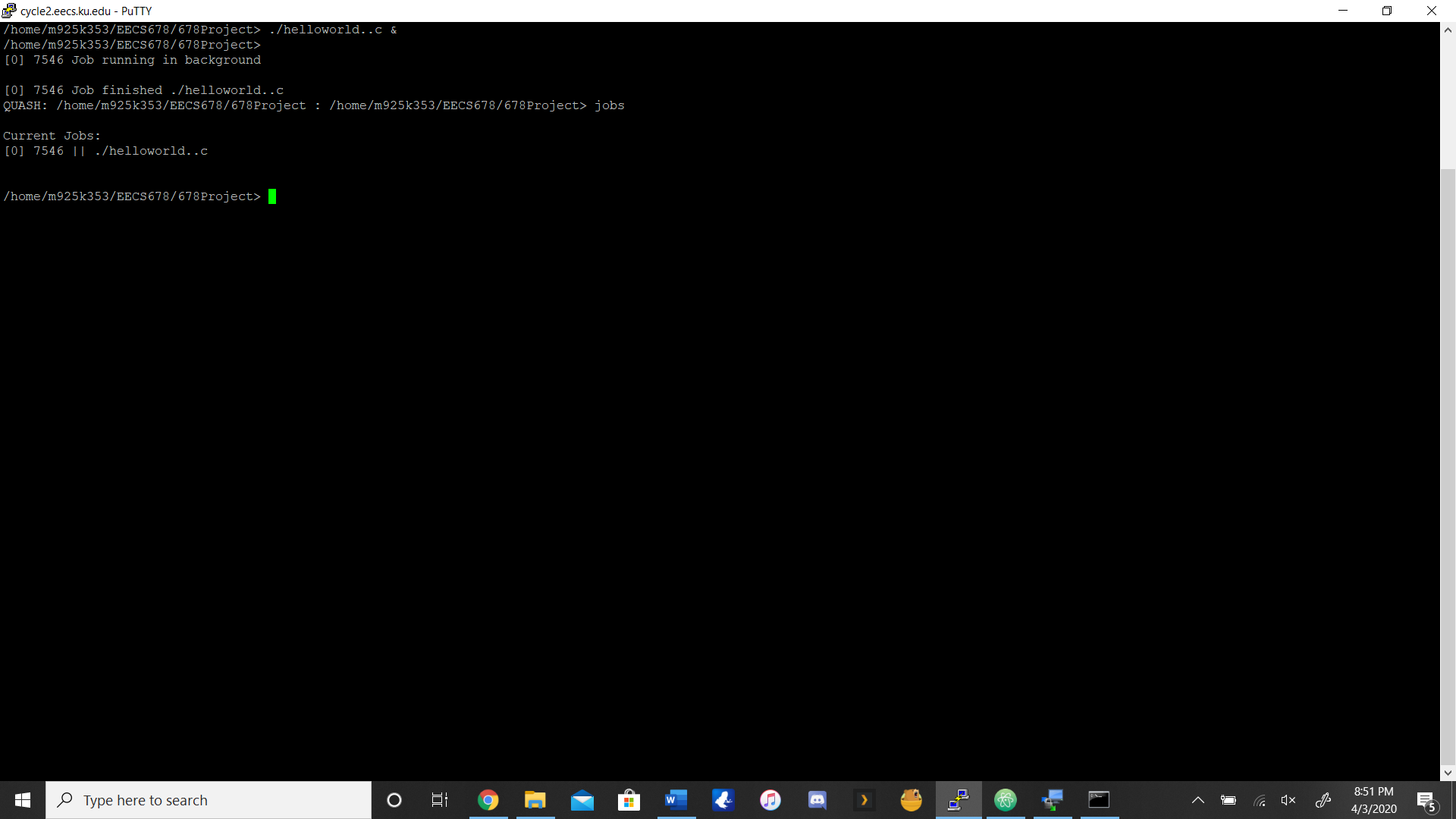


**Grading Policy #7 (Child processes inherit the environment):** 

On the right side you can see how MYVAR can be set equal to NEW\_ENV\_VAR. This is test7 Dr. Kulkarni posted and in the notes these commands were posted and we ended up getting the expected output. We got this work because when we forked, we used execvpe from the unistd.h library to duplicate the actions of the shell.

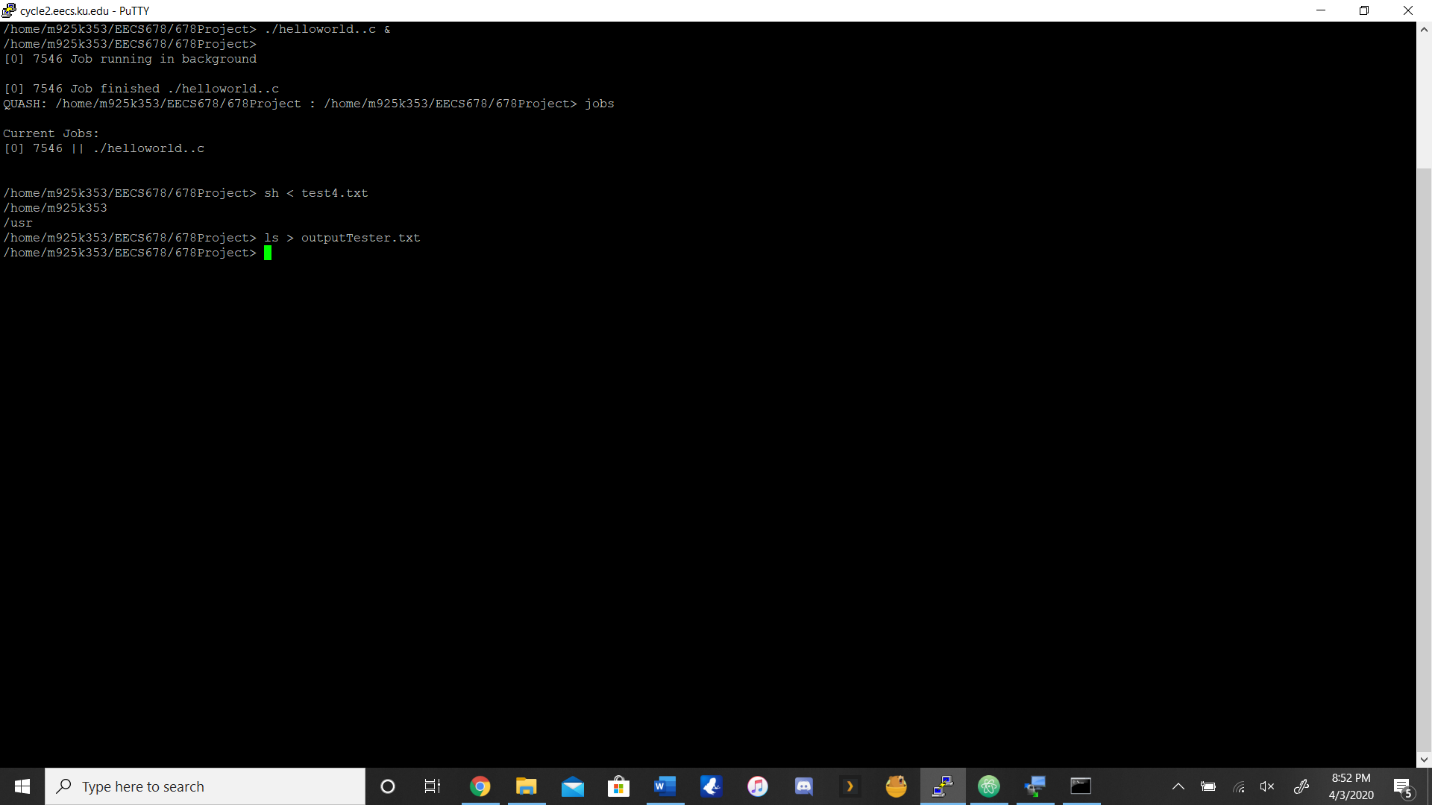
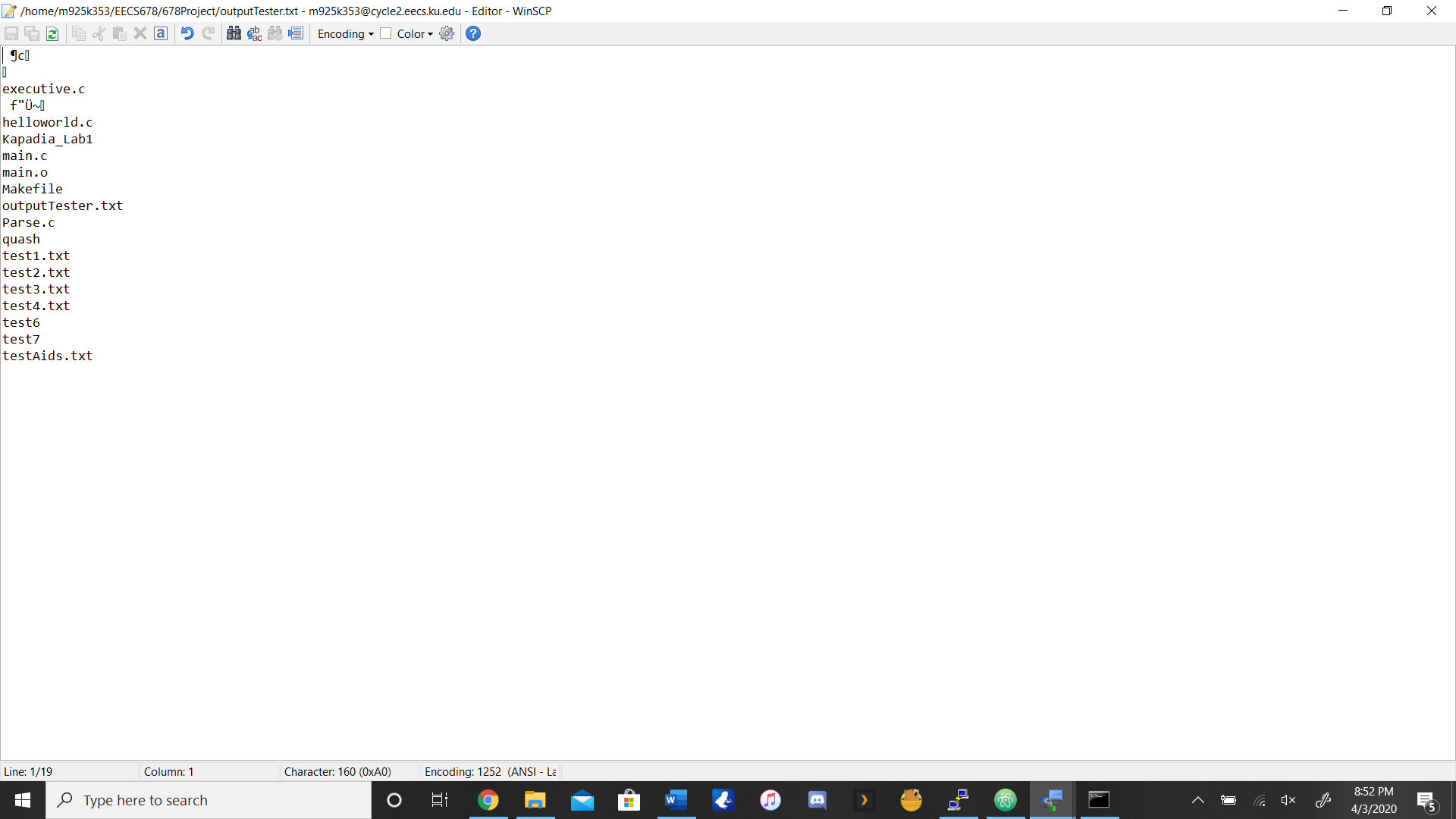
**Grading Policy #8 and #9 (Allow background/foreground execution. Printing/reporting of background processes):**

As you can see in the screenshot below, if you run a file in the background, it reports it with the pid and whether it is running or finished or not. Not only that, you can type in “jobs” in the terminal to see all the current jobs in progress or jobs that are finished as well. We achieved this by making an array of job stuctures that each contain the pid, job nymber assinged in the quash, and a string that is the name of the process. Whenever a parent creates a new child, this job is added to the array of jobs and these details are extracted when the list of jobs are requested for.



**Grading Policy #10 (Allow file redirection [> and <])**

As you can see below in the two pictures, both input and output work. For input, type in sh < <any txt file that has commands> and within the terminal it’ll execute those commands. We have a slight issue here that we couldn’t figure out however. If the txt file contains “quit” at the end, this will just break the program. It will say “not found” for everything. However, if the txt file contains “exit” at the end, the program works perfectly fine. The terminal will execute all the commands in the txt file, but it will not exit the program even though the exit is there. We tried to fix this but we just couldn’t figure it out. All the commands work as long as quit isn’t in the file. We got the input to work by having a character pointer than points to the redirection symbol if there is one in the input, otherwise it points to null. In the case where pointers don’t point to null, the redirection is handled by a separate function. There are different functions for different redirections. For the output, we made a outputTester.txt so you can put <a command> > outputTester.txt and see if it does that command within the txt file. Below you can that the ls command is executed within the txt file. We got the output to work with the redirection function, similar to input redicrection but it has a separate function that writes to file.



**Grading Policy #11 (Allow [1] pipe [|])**

By doing the command “cat main.c | less” we can use a pipe as you can see in the picture below. This can be done with any file, not just main.c. We achived this by creating a function for pipe that forks it to a function that creates a pipe, which in turn transmit the required data to the pipe.

