Project 1
OSC 345-01: Operating Systems
The College of New Jersey
Riya Patel

Requirements Implemented:

1. Child process creation and executing commands
    a. When running the program, child processes are created and their appropriate commands are also executed. This is done by checking if the child was created successfully (`pid<0`) and throwing an error if true, and then executing the instructions of the child if in child process (`pid==0`) and also checking to see if currently in parent process (`pid>0`) and need to wait for child process to complete.
2. Command history management
    a. This was implemented with the use of the `!!` command. When run at the start of the program, there are no commands to call back on, so the user will receive an error message, after a command has been entered, that command is then readily available for recall and execution with the use of `!!`.
3. Add support of input and output redirection
    a. Input and output redirection is also supported. Input direction was tested by the use of a .txt that contained lines of numbers that were tested with `sort < in.txt` which then reads back those numbers in lowest-to-highest order. The output direction is also supported and can be tested by running ls to check the contents of the directory and then when `ls > out.txt` is, the previous commands returns are then seen written to the out.txt text file.
4. Allow the parent and child processes to communicate via a pipe
    a. The parent and child process communication via pipe is a bit buggy but has been implemented to the best of my ability. It seems that sometimes the command `ls -l |less` command executes as desired and sometimes it does not. I was unable to find the root cause of this.
5. Keep working directory information (change directory by [cd] command)
    a. The `cd` command is also implemented as after executing `ls` if you attempt to cd into an existing territory and then execute `ls` again, you can see the contents of the inner folder that you were trying to access.
6. Up-to-date display the current directory in the prompt.
    a. The program also displays the current directory (even when changed by `cd`). This was done by using a string to keep track of the location and having an external method help mimic the syntax of a traditional UNIX Shell UI. As a result, one limitation of the program is that it cannot display the current directory if the current directory is the parent directory of the starting directory. Ex. you are running program in Project1 directory, if there are subdirectories in Project1, it can traverse through and display accurately in the shell but if you attempt to go to "Downloads" where "Project1" may be located, the cd

command will give you the accurate results but the terminal will show `osh>` still. You can navigate back to the previous directory with the `cd  .` command.