**Program Description**

**Introduction:**

This is my shell program. In this shell, you can use all the function that a \*nix shell can do including some built-in commands: cd, dir, clr, etc. Enter “help” for a complete list of built-in commands. External comands: ls, cat, etc. Redirection, background execution and piping.

**Program Design:**

Shell will keep printing out myshell> as a prompt for user to enter input. I have functions like:

* **read\_command**: to read in input from user
* **parse\_command**: to put input into tokens for execution
* **execute\_args**: to execute the commands along with arguments

After reading command, I will perform a check to see what the user wants to do so I have:

* **isRedirection**: to check if user put in ‘>’ or ‘<’ for redirection
* **isBackground**: to check if user put in ‘&’ for background execution
* **isPipe**: to check if user put in ‘|’ for piping

Shell has another usage is to read in a file containing lines of command and execute them all. I create a method called read\_from\_batch and takes in argv as the argument to fetch the file name.

All the Builti-in commands are created from scracth and executed by calling them directly inside of the program.

All External commands are executed using execvp and fork.

If user wants to do background execution. In parent, instead of using wait, I will use clr to clear the screen and print out prompt so user can keep using the shell while the command is executing in child.

For Redirection, I created functions to remove the special character ‘>’ and ‘<’ and to remove the file name from the tokens aray so I can execute the command. Using the code provided in the slide and fix it up a little bit so it will adapt to my own program.

For Piping, I would do the same as redirection: a function to detect if it’s piping, 1 to delete special charater and 2 functions to put commands into different arrays for execution. I will use fork for Piping.

**Test Plan**

* Keep entering input to see if program can handle it
* Printf every variable to debug and check if the read\_command() and parse\_command() work properly
* Test some built-in functions repeatedly to see if shell breaks
* Test external commands, printf the pid to check if fork actually works
* Put & after every command to see if it actually triggers background execution
* Try using external commands with redirection and a file name several times to see if it does append, create file, read form file, etc.
* Pretty much print out every variable after every call to make sure everything is recorded correctly
* Try putting in commands inside of a file and try executing shell with that file to see if shell executes all commands inside