**Documentation**

myShell is a program that is built up with multiple function modules.

**Below I have listed my functions from the code and described their functionality**

**Char readline();**

This function reads input from the keyboard character by character and stops reading it when the user hits the return key or EOF. It takes each character and append them into a string pointer and then returns it.

**Parse(char\* input);**

Parse input function takes the output char array from the readline() function and parse it by strtok() using space as a delimiter. Later outputs of strtok are saved in a char array pointer and returned from this function.

**Quit();**

Quit function sets the loop variable to -1 which is a condition to run the loop endlessly. Setting it to a negative number breaks the loop thus ends the program execution.

**Cd();**

**Clr();**

This function prints out “\033[H\033[2J]” to clear the screen.

**Dir(char\*\* args);**

This function creates a Dir to store the output from the system call opendir() to get the directory. It returns as a struct so function also gets the data from the struct and parses it to print the names of each individual file on the screen. In the end closedir() system call is made to close the dir and clear the memory.

**Environ();**

All of the necessary info requirements are saved into the array info[] which consists 7 elements such as USER, HOMES, PATH, SHELL, OATYPE, PWD, GROUP. A system call getenv() is made by passing in the array info[], it return the desired information and prints it on the screen. If the information are not found, an error message is shown.

**Echo();**

Echo simply prints out the arguments taken at the time executing the command echo in terminal.

**Help();**

**makePipe(char\*\* args1, char\*\* args2);**

an array of filedescriptor is passed into the pipe just created in the function. Later these filedescriptors are used to replace the file descriptors of respective programs. A fork is done to create two processes to use the pipe as a bridge for data communication between two processes. In one of the procees dup2 was used to redirect the output file descriptor of one of the programs to the input file descriptor of pipe. On the other process the input file descriptor is changes to the output file descriptor of the pipe. Thuse data is sent from process A to the pipe and then from the pipe to process B.

**halt();**

halt() is just a fancy name to pause. When this function is invoked, it runs a loop while waiting for keyboard input from the user. The loop only breaks when the user inputs return key as an input. Thus the shell pauses for indefinite amount of time until the user presses return key.

**Execute();**

**singleCommandMode();**

singleCommandMode activates the interactive version of the shell where user can use it just like a normal environment instead of passing in a batch file. It starts with clearing the screen using “\033[H\033[2J]” and then starts the master loop which only breaks when loop equals to a negative number. In the loop the function call readline() to read input from the user and then send it to parse() to tokenize the input into arguments. In the end execute the argument using execute function.

**batchExecution();**

This function is invoked when the user chooses to use a batch file as a input method for the shell instead of using the interacting one. In the function a File pointer is opened which hold the batch file as a readable file. Later a while loop goes through the whole file and calls the parse function to parse the batch file just like singleCommandMode and then call the execute function to execute the mentioned commands.

**Testing of Functionalities**

**cd**A screenshot of a cell phone

Description automatically generated

In this shot I “cd ..” out of the directory and “cd Shell” into Shell directory.

**dir**

A screenshot of a cell phone

Description automatically generated

In this shot I used the “dir” command to print all of the contents of current directory.

**environ**

A screenshot of a cell phone

Description automatically generated

used the “environ” command to show all of the environ variable.

**echo**

A screenshot of a cell phone

Description automatically generated

Used the “echo hello world” to print hello world on the screen.

**help**

A screenshot of a computer screen

Description automatically generated

Used “help” command to view the readme file in shell.

**quit**

A screenshot of a computer

Description automatically generated

Used “quit” command to quit from the shell and return back to default shell.

**pause**

A screenshot of a computer screen

Description automatically generated

Used the pause command to pause the shell for unlimited time until the user presses return key on the keyboard.

**I/O Redirection**

**>A screenshot of a computer screen

Description automatically generated**

used “echo hello > out.txt” to echo the word hello into the file out.txt and then used cat to print out the file.

**>>**

**A screenshot of a cell phone

Description automatically generated**

used “echo world >> out.txt” to concatenate the word world with the existing word in out.txt document.