Pseudocode for Project 2

- Have global array that keeps all internal commands in an array so I can access them through any function

Getting input function()- returns a char*

- Has char* to hold input
- calls readline() to get input
- calls add_history(input) so user can go back and change input
- returns inputted string(char*)
- found in readline.h

Tokenizing input function(char* input) returns a char**

- for loop iterating through entire input and use strtok(str, " ") to separate string
- categorize each token as a 1. command to be executed 2. Arguments to the command 3. Redirection symbols 4. Piping symbol 5. Background execution symbol 6. Input file 7. Output file
- Add commands and arguments to an array, keep track of order
- returns this array (char**)
- (position of things in array will help share if they are commands, arguments, etc)

Print directory function() returns nothing

- Prints current directory
- Uses getcwd()
- Found in unistd.h

Function to execute internal commands(char* command) returns nothing

- For loop that iterates through array and compares the command passed to each internal command
- When find command use switch case to send the command to the proper function

Built in command functions

Clr function

- clears screen
- Uses printf("\033[H\033[J")

Quit function

- Stops running program
- prints out a goodbye message and then calls exit(0)

Dir function

- list contents of the directory
- opens directory, if directory doesn't exist returns error message
- then uses readdir() and while it doesn't equal null prints out each file(string) in directory
- found in dirent.h

Pause function

- pauses screen and waits for user to press enter
- calls getchar() function

Help function

- prints out welcome to shell message
- explains shell and its internal functions
- explains that it can access external commands through UNIX shell
- explains that it can handle background execution, piping, and redirection

Echo function

- given a string as an argument it prints it to screen
- use printf(string)

Environ function

- lists the environment variables
- uses envp
- While envp[i]!=Null print envp[i], i++
- Found in stdio.h

Cd function

- If contains argument, changes directory to that directory, else prints current directory
- uses chdir(inputted path) (if contains argument)
- else calls print directory function
- found in unistd.h

Processing input function(takes tokenized input(char**)) returns nothing

- If array is empty (no command) return()
- If array contains command that is found internally, send to internal command function
- *To check iterate through internal command array and compare array[0](which should be the command in the tokenized input) with internal commands in the array*
- Else check for pipe (call checking for pipe function)
- If pipe present send to piped command execution function
- Else send to external command execution function

external command execution function(takes tokenized input(char**) returns nothing

- fork child process
- use execvp to execute command
- call wait to wait for child process to finish
- to deal with redirection or background execution…
- Have if statements
- If ampersand present, don't call wait() at the end (background execution)
- if redirection symbol present, depending on signal, close either stdin, stdout, or both. Then change stdin or stdout to file given
- if writing new file use O_Trunc, if appending switch that to O_append
- if neither present execute normally

piped command executing function(takes tokenized input(char**) returns nothing

- if pipe is present executes command in this separate function

- output of first command is input for other
- may call external command execution function within this

POTENTIAL FUNCTIONS

check for ampersand function (takes tokenized input(char**)) Boolean function
- iterates through tokenized input looking for ampersand
- if present returns true
- else returns false

check for redirection function (takes tokenized input(char**)) Boolean function
- iterates through tokenized input looking for redirection symbol
- if present return true
- else return false

check for piping function (takes tokenized input(char**)) Boolean function
- iterate through tokenized input looking for pipe
- if present returns true
- else returns false

main method
- if program was called with no argument run normally
- if two or more arguments it means its reading line by line from file
- to start shell…
- Writes welcome to shell message
- Prints username uses getenv() to get the username from the environment
- Gets ready for input by clearing screen which shows it is time to take input
- While loop…
- while exit() function has not been inputted
- print current directory to simulate terminal (call print directory function)
- get input (call get input function) and save it in a char*
- tokenize input (call tokenize inout function which takes the char*) and save it in a char**
- execute input (call process input function which takes the char**)

I will have own separate tokenizer program to make sure it is splitting up the string the correct way and saving them into the array properly.

Possible problems include redirection. If redirection is present I must figure out if it's input redirection or output redirection. Then if it's output redirection is it appending to file or creating new file. Also figuring out what arguments in the tokenized strings are the files for the I/O redirection.

Another problem involves piping. Figuring out which commands and arguments are to be done first than to be used as the input for the next command will be challenging, which is why I will keep it as a separate function from the normal external command execution function (I may call it in the function) to make sure it is be processed correctly. Will test as its own separate

function in its own program with the tokenizer function as well to make sure it tokenizes it correctly then executes it correctly.
Plan on creating the internal commands in a separate file first then calling each in main to see if they work properly. (must figure out output redirection for specific functions)
Processing input function may cause challenges with argument passing. If find command I must also pass arguments with it to proper executing function. That's why I think passing the hwole tokenized input to each execution function and then accessing them by array elements may be the way to do it. (unsure).