Task 2: Create a shell

Conditions:

1. Run all commands that a shell can execute.
2. Arrow keys to get previous commands.
3. "!" will get history in sequence.
4. "!!" will get history in reverse.
5. "!g" will get first command starting with "g".
6. "!!g" will get last command starting with "g".
7. Use system calls only.

Our Approach:

We divided creating a shell into smaller sections to make things easier:

1. Check for key events.
2. Read input from the user.
3. Check if the input is a call to history i.e. !!, !, !1, !g etc.
4. Save that input in a text file if it is not a call to history.
5. Parse that input so that execvp can use it as an argument and execute.

1. Check for Key Events:

For key events we used <termios.h> to change the terminal settings to not display characters printed by arrow keys but instead they are compared with their codes to determine what key is pressed.

2. Read Input:

To read input Read() system call is used which read character by character until the user press enter. All of this is done in takeInput() method.

3. Read from history:

The input is compared with "!" and "!!" or if the user presses up or down array key. There are two methods one readByLineNo and readByExclaimationMarks to read from the text file.

4. Save Input:

When the input is taken takeInput method returns a pointer which is then passed into another method called saveInput(), this method uses open and write system calls to open a text file in append mode and writes the command in it line by line.

5. Execution:

When the input is taken it is in a pointer and the arguments are sperated by spaces. Execvp cannot read arguments separated by spaces. So a for loop runs and replaces all spaces by NULL. And then the pointer is parsed in execvp system call.

List of functions in program:

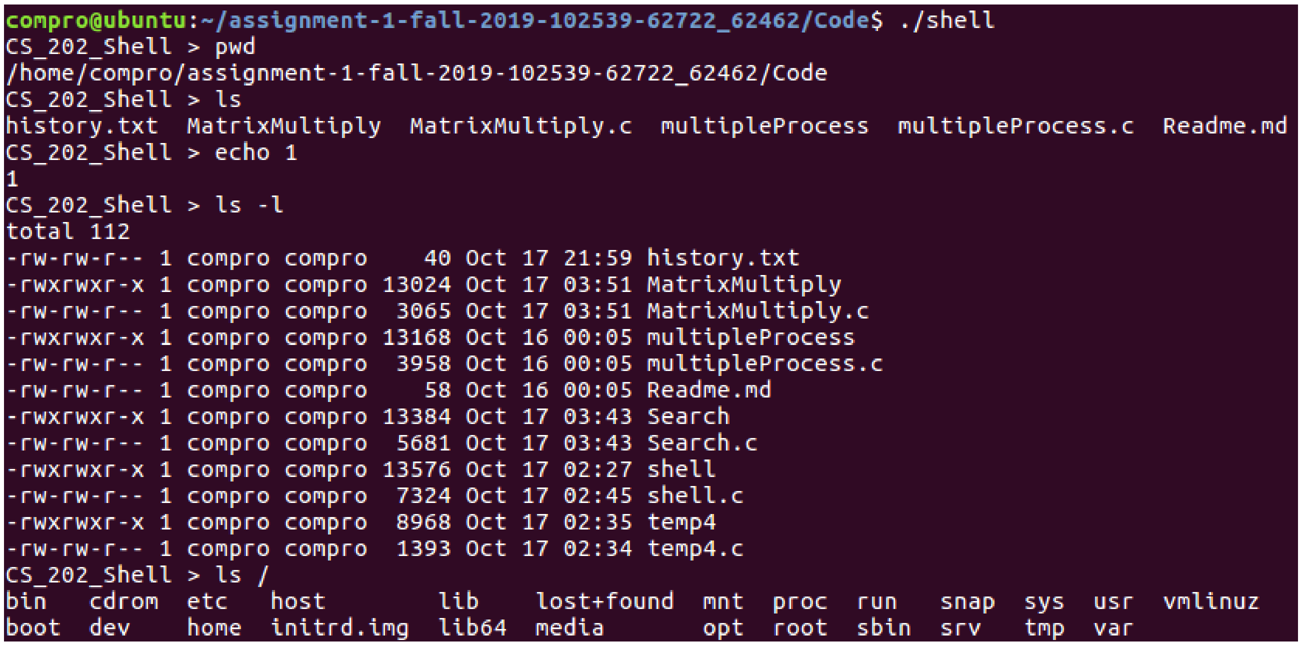
1. **int readKey():** This method reads for the arrow keys and returns 1 if an arrow key is pressed and increments or decrements a global variable named **globalLine**
2. **void reader(int lineNo)**: This method is used to read a specific line from a text file. In the program it is called after **readKey()** returns 1 and it takes globalLine as an argument which contains the line number that will shown. Depending upon the conditions this method also has functionality to read lines starting with a specified character. This method is also called by some other methods.
3. **void readerByLineNo(int reverseLineNo):** This method calls the **reader** with argument **(-2)** to get the total number of lines which is saved in a global variable **lines** and then it subtracts **lines** by its parameter **reverseLineNo** and the result is then passed in reader(**result**) to read the lines from the bottom or LIFO.
4. **void readerByExclaimationMarks(char arg[], int arglen):** This method is used to check if the input is a special input that is not a command but is specific to this shell program. This program first reads the input and then checks if it is reversed order or sequence command, after that it checks if the there is an alphabet or a number after the exclaimation mark/s. If there is an alphabet then this method toggles a boolean and calls **reader(-2)**. Then that boolean runs an if condition inside the **reader** which would not have ran normally and that condition checks the first character of each line and matches it to the input character when a match is found the line is saved in a global variable to be executed.

   But if there is a number after the exclaimation mark/s then this method calls **readerByLineNo** and passes the number as the argument.

5. **char *takeInput():** In this method all above methods are called. This method uses **read** system call to take input from user and depending upon the input it either calls **readerByExclaimationMarks** or simply returns a pointer to the input taken.
6. **void saveInput(char *input):** This method runs after **takeInput** and and saves the pointer returned by the **takeInput** method in a text file by using **open** and **write** system calls.
7. **Int main():** In main method there is an infinite while loop and in that loop **takeInput** and **saveInput** methods are called. After we have the input in a pointer we split that pointer by spaces and save it into another array like an array of strings instead of an array of characters. After this is done we use **execvp** system call to execute the command if it is valid and then the loop runs again.

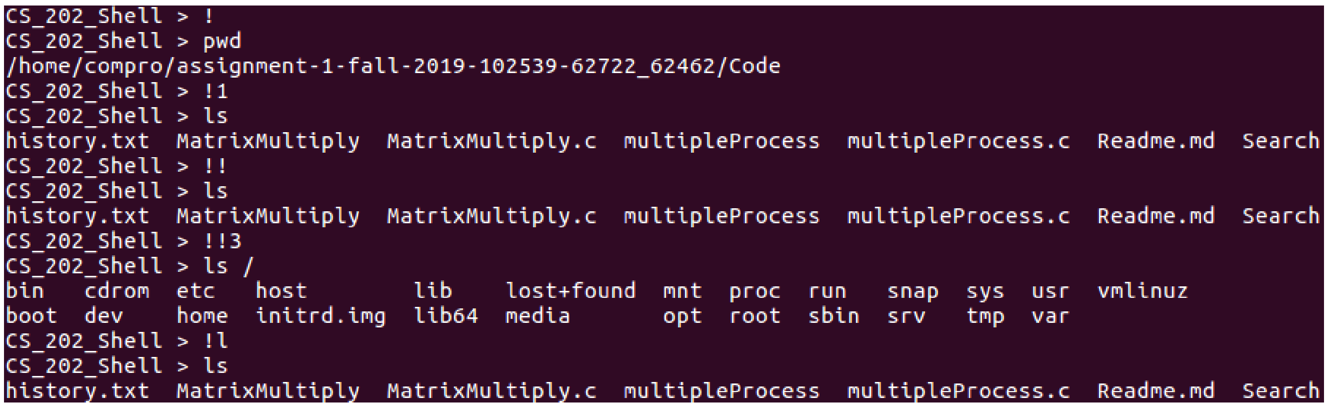
Sample Execution:

Simple command execution:

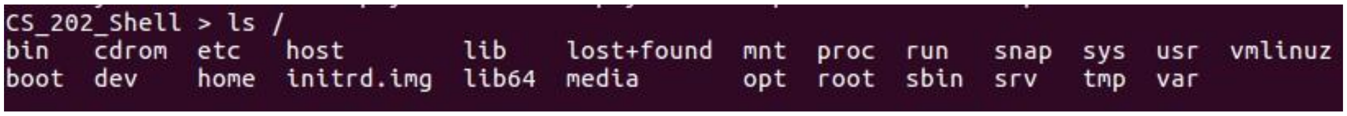


```
compro@ubuntu:~/assignment-1-fall-2019-102539-62722_62462/Code$ ./shell
CS_202_Shell > pwd
/home/compro/assignment-1-fall-2019-102539-62722_62462/Code
CS_202_Shell > ls
history.txt  MatrixMultiply  MatrixMultiply.c  multipleProcess  multipleProcess.c  Readme.md
CS_202_Shell > echo 1
1
CS_202_Shell > ls -l
total 112
-rw-rw-r-- 1 compro compro    40 Oct 17 21:59 history.txt
-rwxrwxr-x 1 compro compro 13024 Oct 17 03:51 MatrixMultiply
-rw-rw-r-- 1 compro compro  3065 Oct 17 03:51 MatrixMultiply.c
-rwxrwxr-x 1 compro compro 13168 Oct 16 00:05 multipleProcess
-rw-rw-r-- 1 compro compro  3958 Oct 16 00:05 multipleProcess.c
-rw-rw-r-- 1 compro compro    58 Oct 16 00:05 Readme.md
-rwxrwxr-x 1 compro compro 13384 Oct 17 03:43 Search
-rw-rw-r-- 1 compro compro  5681 Oct 17 03:43 Search.c
-rwxrwxr-x 1 compro compro 13576 Oct 17 02:27 shell
-rw-rw-r-- 1 compro compro  7324 Oct 17 02:45 shell.c
-rwxrwxr-x 1 compro compro  8968 Oct 17 02:35 temp4
-rw-rw-r-- 1 compro compro  1393 Oct 17 02:34 temp4.c
CS_202_Shell > ls /
bin   cdrom  etc   host        lib    lost+found  mnt  proc  run   snap  sys  usr  vmlinuz
boot  dev    home  initrd.img  lib64  media       opt  root  sbin  srv   tmp  var
```

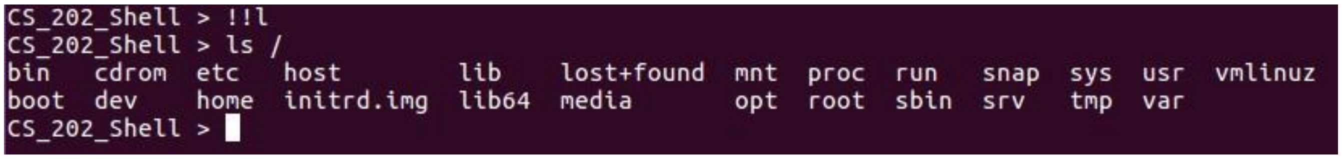
Special commands for history:



```
CS_202_Shell > !
CS_202_Shell > pwd
/home/compro/assignment-1-fall-2019-102539-62722_62462/Code
CS_202_Shell > !1
CS_202_Shell > ls
history.txt  MatrixMultiply  MatrixMultiply.c  multipleProcess  multipleProcess.c  Readme.md  Search
CS_202_Shell > !!
CS_202_Shell > ls
history.txt  MatrixMultiply  MatrixMultiply.c  multipleProcess  multipleProcess.c  Readme.md  Search
CS_202_Shell > !!3
CS_202_Shell > ls /
bin   cdrom  etc   host        lib    lost+found  mnt  proc  run   snap  sys  usr  vmlinuz
boot  dev    home  initrd.img  lib64  media       opt  root  sbin  srv   tmp  var
CS_202_Shell > !l
CS_202_Shell > ls
history.txt  MatrixMultiply  MatrixMultiply.c  multipleProcess  multipleProcess.c  Readme.md  Search
```



```
CS_202_Shell > ls /
bin   cdrom  etc   host        lib    lost+found  mnt  proc  run   snap  sys  usr  vmlinuz
boot  dev    home  initrd.img  lib64  media       opt  root  sbin  srv   tmp  var
```



```
CS_202_Shell > !!l
CS_202_Shell > ls /
bin   cdrom  etc   host        lib    lost+found  mnt  proc  run   snap  sys  usr  vmlinuz
boot  dev    home  initrd.img  lib64  media       opt  root  sbin  srv   tmp  var
CS_202_Shell > █
```

Note: All commands above are executed together with an empty history file.