CSE231 - Operating Systems
Assignment 1
Exercise 1.2

Name: Ansh Arora
Roll Number: 2019022
Section: B
Branch: CSE

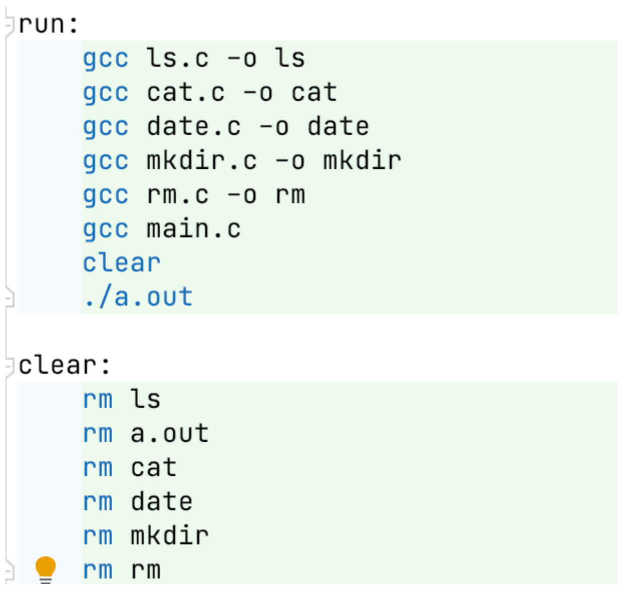
Task: Basic Unix Shell implementation in C using syscalls and file handling, 10 commands and 2 options each with Case Handling.

To run the program, run the following make command in the terminal –

make run

To clear out the files generated and reset the program for a new run, run the following command in the terminal –

make clear

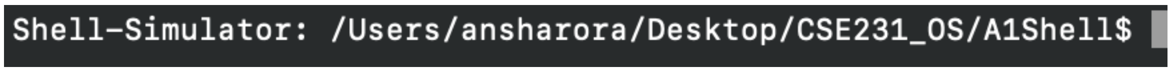


```
run:
	gcc ls.c -o ls
	gcc cat.c -o cat
	gcc date.c -o date
	gcc mkdir.c -o mkdir
	gcc rm.c -o rm
	gcc main.c
	clear
	./a.out

clear:
	rm ls
	rm a.out
	rm cat
	rm date
	rm mkdir
	rm rm
```

Shell Interface –

The interface at all times shows the current position the user is in.


```
Shell-Simulator: /Users/ansharora/Desktop/CSE231_OS/A1Shell$
```

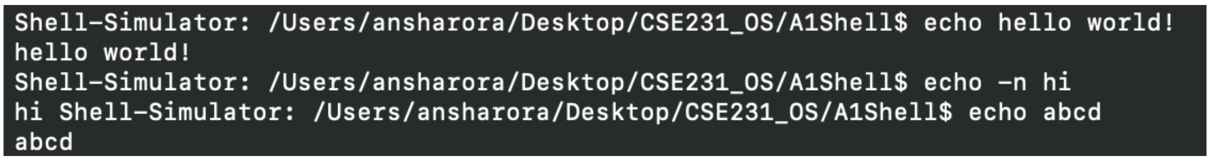
The users can fill in information like they would usually in a terminal, with the specifics of each command laid out here in this documentation and alongside in the docs provided along with terminal (can be accessed using help [command]).

Commands Implemented –

1) echo – The echo command takes in arguments in the echo [options] [args] format and then prints out the args in the terminal.
   This has been implemented using standard input output commands and array data structures.
   Flags –
   - -n – For not going to a new line on executing echo.
   - --help – Shows the documentation for the echo command in the terminal.

   Case Handling –
   - Cases where non-flag multiple words were used have been handled and are treated as normal text.

   Snippet –



```
Shell-Simulator: /Users/ansharora/Desktop/CSE231_OS/A1Shell$ echo hello world!
hello world!
Shell-Simulator: /Users/ansharora/Desktop/CSE231_OS/A1Shell$ echo -n hi
hi Shell-Simulator: /Users/ansharora/Desktop/CSE231_OS/A1Shell$ echo abcd
abcd
```

2) history – The history command shows the history of commands executed in the terminal and maintains a permanent storage of these commands. It follows the history [options] format.
   This has been implemented using file input/output using fprintf and getline. The -d offset flag uses the remove and rename commands for taking the new file and making it

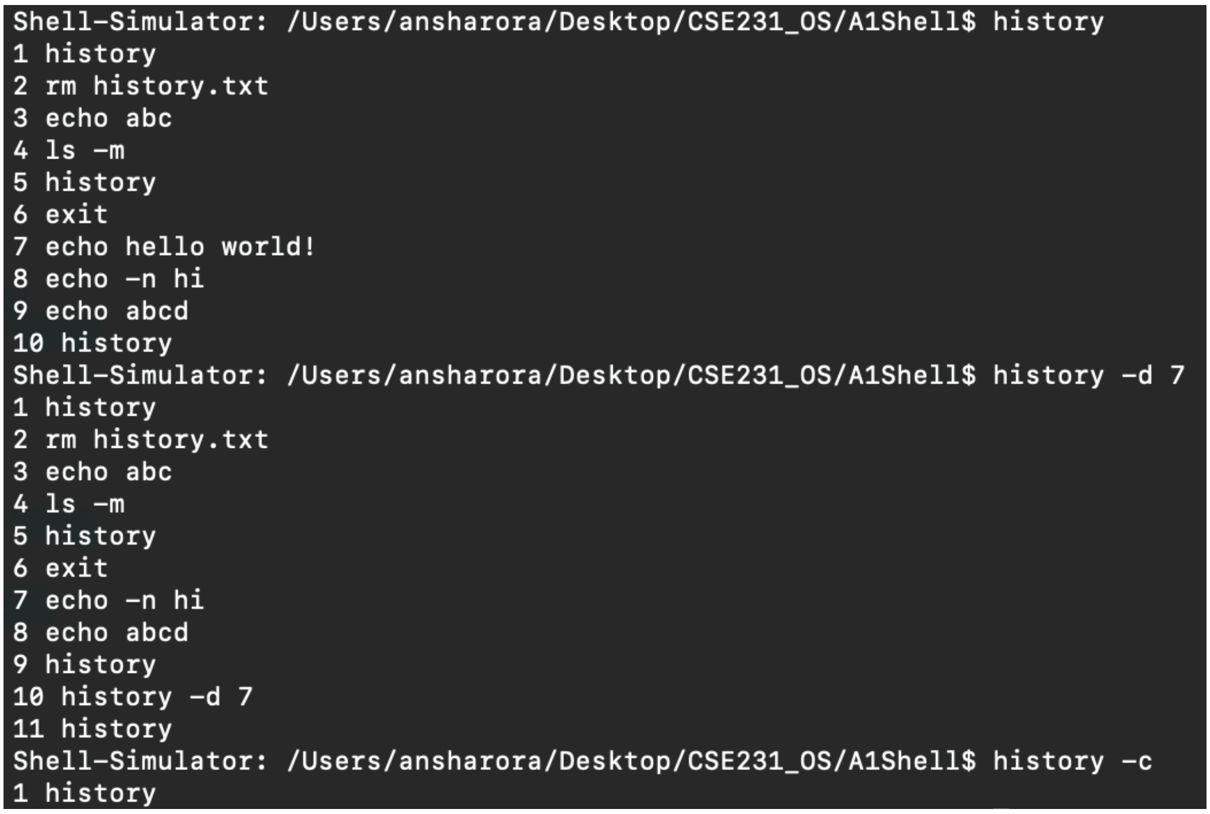
into the new file for removing a given line’s stored command.
Flags –

- -c – For clearing out the whole history and then displaying it.
- -d offset – For clearing out the given offset’s history from storage and then displaying it.

Case Handling –

- Invalid flags/entries are checked for and display invalid syntax command if occurrence happens.
- The permanent history case has been handled with the help of a text file history.txt that resided in the root directory of the shell.

Snippet –



```
Shell-Simulator: /Users/ansharora/Desktop/CSE231_OS/A1Shell$ history
1 history
2 rm history.txt
3 echo abc
4 ls -m
5 history
6 exit
7 echo hello world!
8 echo -n hi
9 echo abcd
10 history
Shell-Simulator: /Users/ansharora/Desktop/CSE231_OS/A1Shell$ history -d 7
1 history
2 rm history.txt
3 echo abc
4 ls -m
5 history
6 exit
7 echo -n hi
8 echo abcd
9 history
10 history -d 7
11 history
Shell-Simulator: /Users/ansharora/Desktop/CSE231_OS/A1Shell$ history -c
1 history
```

3) pwd – The pwd command is used to print out the current directory in which the user is in. It follows the pwd [options] format.
This has been implemented using the getcwd function syscall which returns the current location to the allotted char array variable. The -L flag has been implemented using the getenv command.

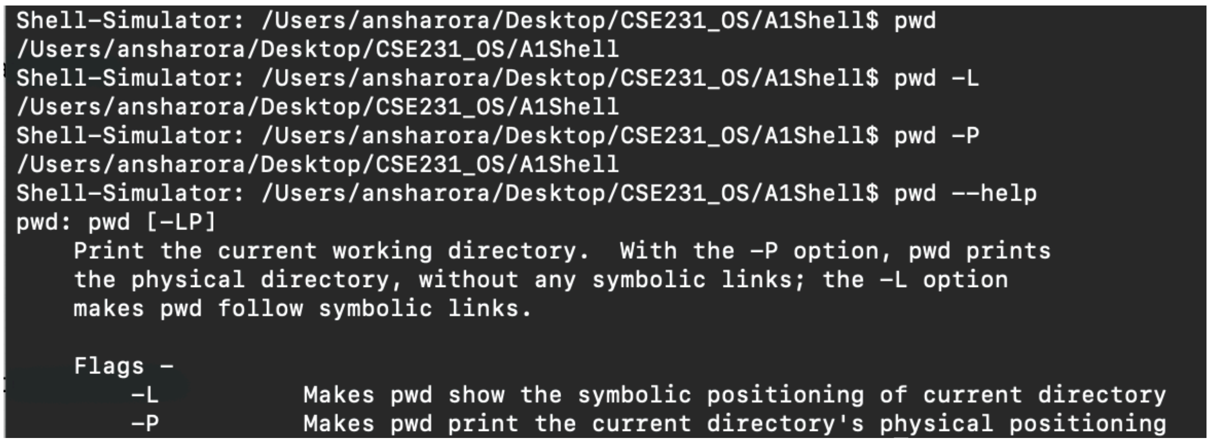
Flags –

- -L – For showing the symbolic positioning of current directory.
- -P – For showing the physical positioning of current directory.
- --help – For showing the help documentation for the command.

Case Handling –

- Invalid flags/entries are checked for and display invalid syntax command if occurrence happens.
- The case of pwd and -P behaving similarly and -L behaving differently has been taken into account by using different commands.

Snippet –



```
Shell-Simulator: /Users/ansharora/Desktop/CSE231_OS/A1Shell$ pwd
/Users/ansharora/Desktop/CSE231_OS/A1Shell
Shell-Simulator: /Users/ansharora/Desktop/CSE231_OS/A1Shell$ pwd -L
/Users/ansharora/Desktop/CSE231_OS/A1Shell
Shell-Simulator: /Users/ansharora/Desktop/CSE231_OS/A1Shell$ pwd -P
/Users/ansharora/Desktop/CSE231_OS/A1Shell
Shell-Simulator: /Users/ansharora/Desktop/CSE231_OS/A1Shell$ pwd --help
pwd: pwd [-LP]
    Print the current working directory.  With the -P option, pwd prints
    the physical directory, without any symbolic links; the -L option
    makes pwd follow symbolic links.

    Flags -
        -L          Makes pwd show the symbolic positioning of current directory
        -P          Makes pwd print the current directory's physical positioning
```

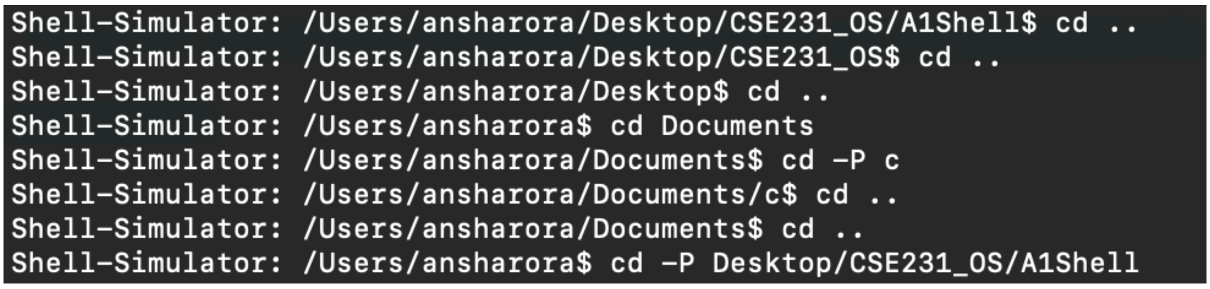
4) cd – The cd command is used to change directories. It follows the cd [options] [args] format, where it changes the shell’s current directory to the one given in args, if it is accessible.
cd has been implemented using a combination of getcwd and chdir commands, where the strings are operated upon.
Flags –

- -P - Instructs the shell to use the physical directory structure instead of following symbolic links.
- --help - Displays this help page

Case Handling –

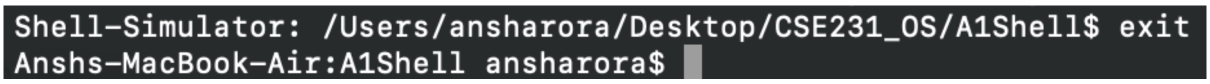
- The chdir commands return values have been utilized to return the user with error if given folder doesn't exist.
- Multiple argument cases, such as .. have been handled by recursion.
- Invalid flags/entries are checked for and display invalid syntax command if occurrence happens.

Snippet –



```
Shell-Simulator: /Users/ansharora/Desktop/CSE231_OS/A1Shell$ cd ..
Shell-Simulator: /Users/ansharora/Desktop/CSE231_OS$ cd ..
Shell-Simulator: /Users/ansharora/Desktop$ cd ..
Shell-Simulator: /Users/ansharora$ cd Documents
Shell-Simulator: /Users/ansharora/Documents$ cd -P c
Shell-Simulator: /Users/ansharora/Documents/c$ cd ..
Shell-Simulator: /Users/ansharora/Documents$ cd ..
Shell-Simulator: /Users/ansharora$ cd -P Desktop/CSE231_OS/A1Shell
```

5) exit – Simply used to exit from the running shell iteration. Exiting doesn't affect the history store. It simply calls the exit(0) system call and ends the process.



```
Shell-Simulator: /Users/ansharora/Desktop/CSE231_OS/A1Shell$ exit
Anshs-MacBook-Air:A1Shell ansharora$ 
```

6) ls – ls command is used to display the content of the directory which we currently are in at a given time in the shell. It follows the ls [options] format.
This has been implemented as an external command, where the external c file is called as a child process using the execvp command. The execvp takes in arguments passed from the parent process and passes them to the external C file. The parent process waits using the wait() function until the child ends its process using exit() function.
The ls command implementation makes use of the dirent structure, which stores directory entries. Firstly, a directory is opened using opendir, and then individual entries are read using the readdir function.
Flags –

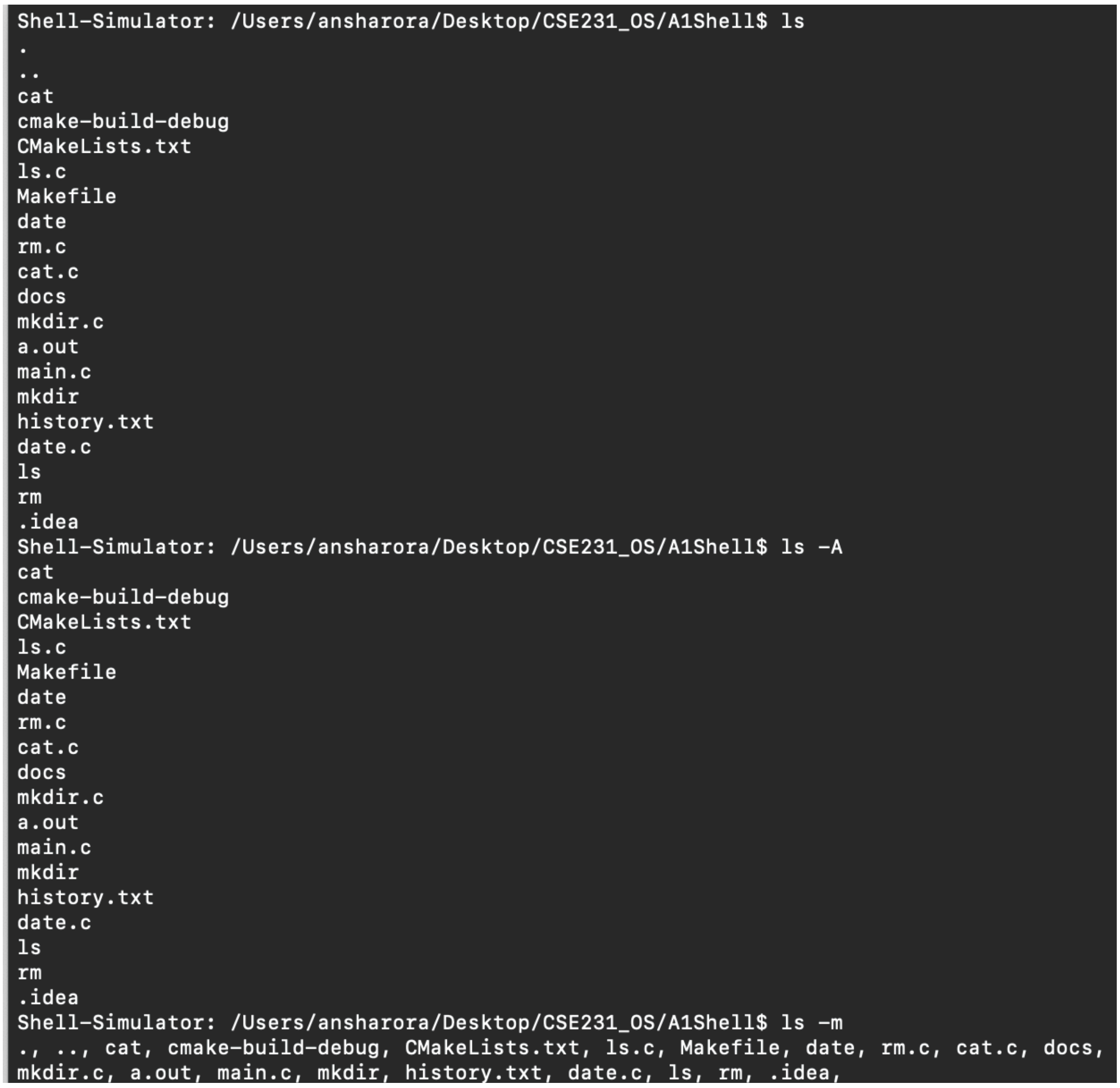
- -A – Displays all files and directories minus the . and .. files in a directory.

- -m – Displays all files and directories inside current directory seperated by commas.

Case Handling –

- The opendir command’s return value is used to check whether a directory can sucessfully be opened or not (NULL check).
- Invalid flags/entries are checked for and display invalid syntax command if occurrence happens.

Snippet –



```
Shell-Simulator: /Users/ansharora/Desktop/CSE231_OS/A1Shell$ ls
.
..
cat
cmake-build-debug
CMakeLists.txt
ls.c
Makefile
date
rm.c
cat.c
docs
mkdir.c
a.out
main.c
mkdir
history.txt
date.c
ls
rm
.idea
Shell-Simulator: /Users/ansharora/Desktop/CSE231_OS/A1Shell$ ls -A
cat
cmake-build-debug
CMakeLists.txt
ls.c
Makefile
date
rm.c
cat.c
docs
mkdir.c
a.out
main.c
mkdir
history.txt
date.c
ls
rm
.idea
Shell-Simulator: /Users/ansharora/Desktop/CSE231_OS/A1Shell$ ls -m
., .., cat, cmake-build-debug, CMakeLists.txt, ls.c, Makefile, date, rm.c, cat.c, docs,
mkdir.c, a.out, main.c, mkdir, history.txt, date.c, ls, rm, .idea,
```

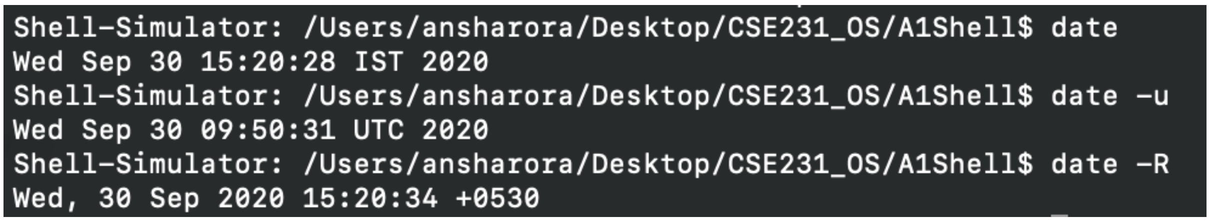
7) date – The date command is used to display the current date and time in accordance with the system’s set date. It follows the date [options] format.
This has been implemented as an external command, where the external c file is called as a child process using

the execvp command. The execvp takes in arguments passed from the parent process and passes them to the external C file. The parent process waits using the wait() function until the child ends its process using exit() function. The date command has been implemented by using C’s time.h function library and the tm data structure. It makes use of the localtime() and getime() functions.

Flags –
- -u – Shows the date and time in GMT.
- -R – Use RFC 2822 date and time format.

Case Handling –
- Invalid flags/entries are checked for and display invalid syntax command if occurrence happens.
- The exact specific syntax of the UNIX system has been imitated by using string operations.

Snippet –



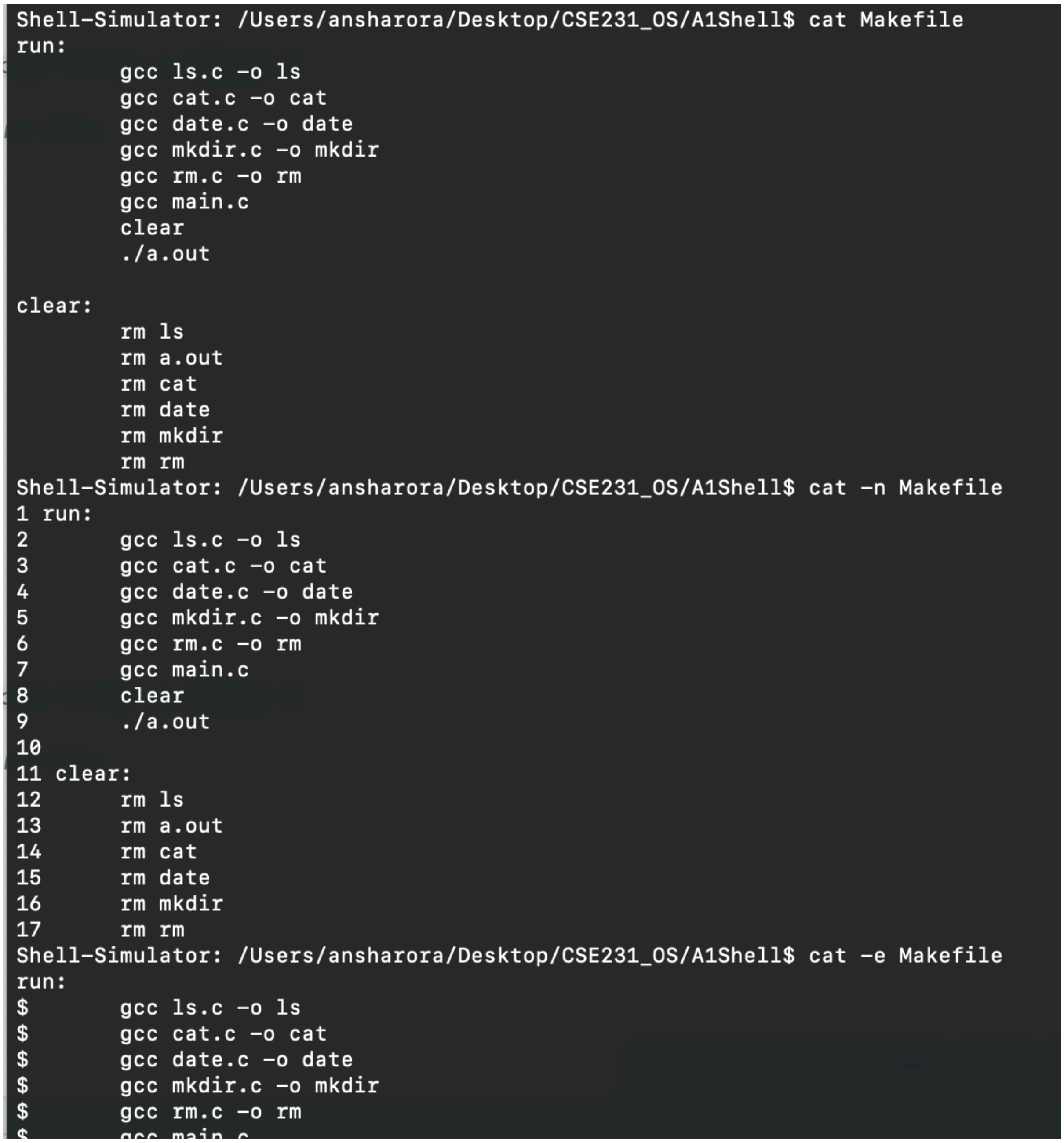
```
Shell-Simulator: /Users/ansharora/Desktop/CSE231_OS/A1Shell$ date
Wed Sep 30 15:20:28 IST 2020
Shell-Simulator: /Users/ansharora/Desktop/CSE231_OS/A1Shell$ date -u
Wed Sep 30 09:50:31 UTC 2020
Shell-Simulator: /Users/ansharora/Desktop/CSE231_OS/A1Shell$ date -R
Wed, 30 Sep 2020 15:20:34 +0530
```

8) cat – The cat command is used to display the contents of the selected files, line by line. It follows the cat [options] [args] format.
This has been implemented as an external command, where the external c file is called as a child process using the execvp command. The execvp takes in arguments passed from the parent process and passes them to the external C file. The parent process waits using the wait() function until the child ends its process using exit() function. It makes use of standard C file handling commands fopen(), getline() and fclose().
Flags –
- -n – Numbers each line of the file being displayed.
- -e – Demarcates the difference between two lines using a $ sign.

Case Handling –

- Makes use of the fopen command’s NULL return if a case arises where a particular file that has to be displayed doesn’t exist.
- Invalid flags/entries are checked for and display invalid syntax command if occurrence happens.

Snippet –



```
Shell-Simulator: /Users/ansharora/Desktop/CSE231_OS/A1Shell$ cat Makefile
run:
        gcc ls.c -o ls
        gcc cat.c -o cat
        gcc date.c -o date
        gcc mkdir.c -o mkdir
        gcc rm.c -o rm
        gcc main.c
        clear
        ./a.out

clear:
        rm ls
        rm a.out
        rm cat
        rm date
        rm mkdir
        rm rm
Shell-Simulator: /Users/ansharora/Desktop/CSE231_OS/A1Shell$ cat -n Makefile
1 run:
2       gcc ls.c -o ls
3       gcc cat.c -o cat
4       gcc date.c -o date
5       gcc mkdir.c -o mkdir
6       gcc rm.c -o rm
7       gcc main.c
8       clear
9       ./a.out
10
11 clear:
12      rm ls
13      rm a.out
14      rm cat
15      rm date
16      rm mkdir
17      rm rm
Shell-Simulator: /Users/ansharora/Desktop/CSE231_OS/A1Shell$ cat -e Makefile
run:
$       gcc ls.c -o ls
$       gcc cat.c -o cat
$       gcc date.c -o date
$       gcc mkdir.c -o mkdir
$       gcc rm.c -o rm
$       gcc main.c
```

9) rm – The rm command is used to remove a particular file from the current directory we are in. It follows the rm [options] [arg] format.

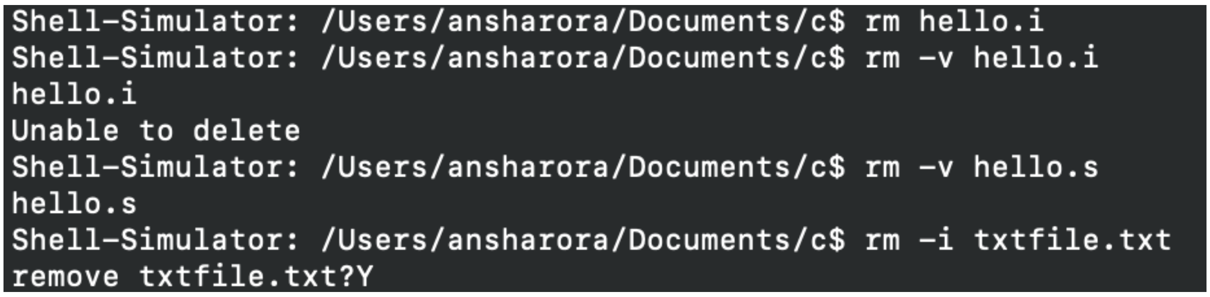
This has been implemented as an external command, where the external c file is called as a child process using the execvp command. The execvp takes in arguments passed from the parent process and passes them to the external C file. The parent process waits using the wait() function until the child ends its process using exit() function. The rm function makes use of the remove function to remove specific files and uses its integer return value for Case Handling.
Flags –

- -v – Verbose, tells the user which file is being removed.
- -i – Confirms with the user whether in fact a file needs to be deleted or not before removing it.

Case Handling –

- Makes use of remove function's return value to check whether a file was succesfully removed or not (due to lack of presence or authority issues).
- Invalid flags/entries are checked for and display invalid syntax command if occurrence happens.

Snippet –



```
Shell-Simulator: /Users/ansharora/Documents/c$ rm hello.i
Shell-Simulator: /Users/ansharora/Documents/c$ rm -v hello.i
hello.i
Unable to delete
Shell-Simulator: /Users/ansharora/Documents/c$ rm -v hello.s
hello.s
Shell-Simulator: /Users/ansharora/Documents/c$ rm -i txtfile.txt
remove txtfile.txt?Y
```

10) mkdir – The mkdir command is used to create a directory with a given name in the directory we presently are in. The format for mkdir is mkdir [options] [args], where args are the names for the new directories that need to be formed. Files are created with an rwxrwxrwx(0777) mode.
This has been implemented as an external command, where the external c file is called as a child process using the execvp command. The execvp takes in arguments passed from the parent process and passes them to the external C file. The parent process waits using the wait() function until the child ends its process using exit() function.

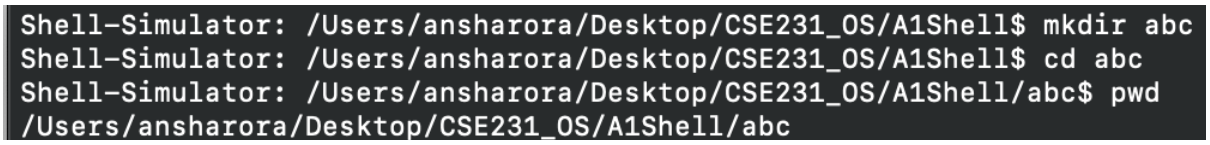
The directories are created by using the standard lib function mkdir where name of the folder and its accessibility modes are passed.
Flags –

- -v – Verbose, tells the user about the status of creation of the directory.
- -p – Takes a path and creates all of the necessary directories on the path that don't exist.

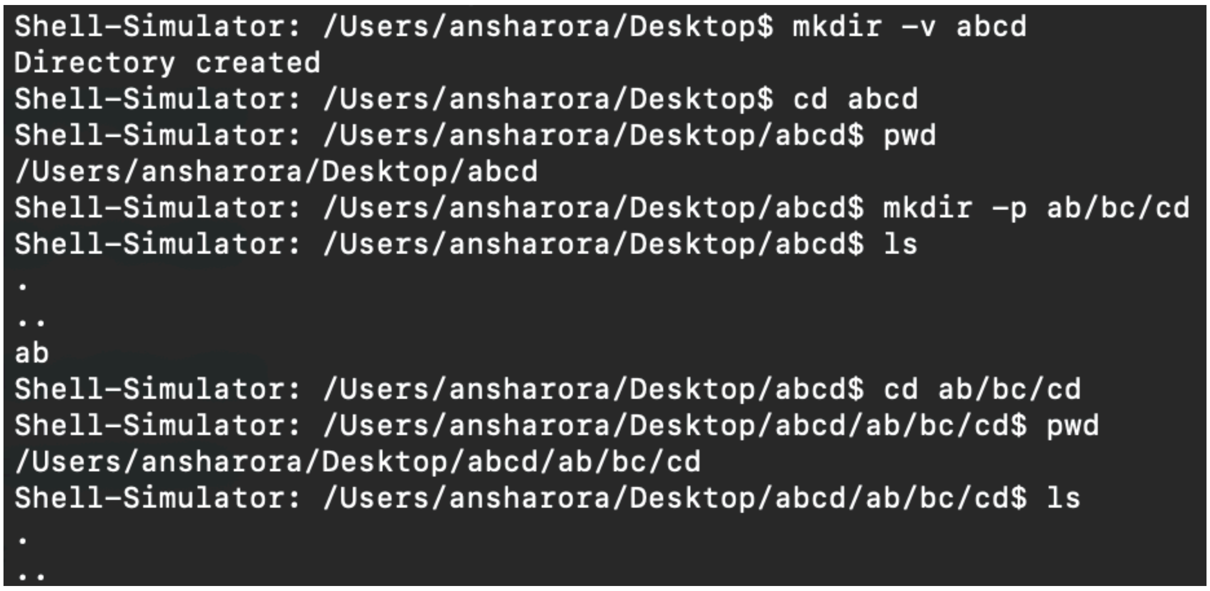
Case Handling –

- The return value of mkdir is an integer 0 if the directory creation was successful. This is used to handle the directory creation related errors.
- Invalid flags/entries are checked for and display invalid syntax command if occurrence happens.

Snippet –



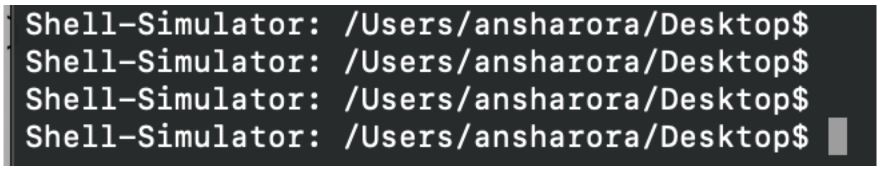
```
Shell-Simulator: /Users/ansharora/Desktop/CSE231_OS/A1Shell$ mkdir abc
Shell-Simulator: /Users/ansharora/Desktop/CSE231_OS/A1Shell$ cd abc
Shell-Simulator: /Users/ansharora/Desktop/CSE231_OS/A1Shell/abc$ pwd
/Users/ansharora/Desktop/CSE231_OS/A1Shell/abc
```



```
Shell-Simulator: /Users/ansharora/Desktop$ mkdir -v abcd
Directory created
Shell-Simulator: /Users/ansharora/Desktop$ cd abcd
Shell-Simulator: /Users/ansharora/Desktop/abcd$ pwd
/Users/ansharora/Desktop/abcd
Shell-Simulator: /Users/ansharora/Desktop/abcd$ mkdir -p ab/bc/cd
Shell-Simulator: /Users/ansharora/Desktop/abcd$ ls
.
..
ab
Shell-Simulator: /Users/ansharora/Desktop/abcd$ cd ab/bc/cd
Shell-Simulator: /Users/ansharora/Desktop/abcd/ab/bc/cd$ pwd
/Users/ansharora/Desktop/abcd/ab/bc/cd
Shell-Simulator: /Users/ansharora/Desktop/abcd/ab/bc/cd$ ls
.
..
```

Other Notable Additions –

- I have added a full fledged help section which works for all commands.
- I have handled the case for blank commands, in which case my shell just passes a new command line, in the same way a real OS terminal functions.



```
Shell-Simulator: /Users/ansharora/Desktop$
Shell-Simulator: /Users/ansharora/Desktop$
Shell-Simulator: /Users/ansharora/Desktop$
Shell-Simulator: /Users/ansharora/Desktop$
```