

ASSIGNMENT 2 - CS250

Chaitanya Bisht, 12040450

PART A : Build a Simple Shell

Execute the shell using the following commands:

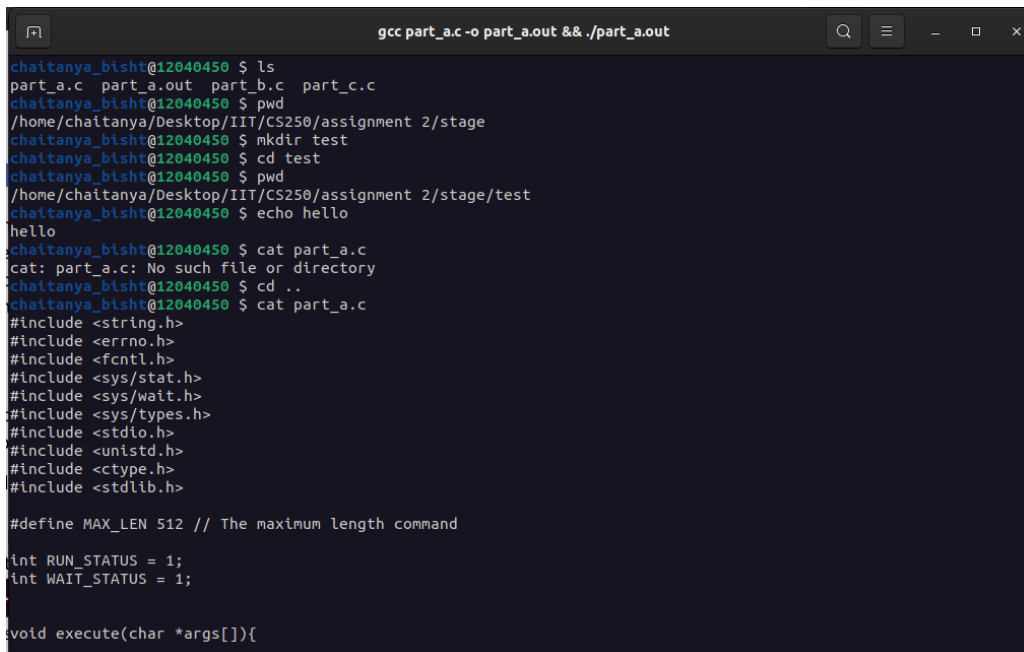
```
$ gcc partA.c -o partA.out && ./partA.out
```

In this part, the shell simply runs commands with arguments and switches(flags) . The input stream is read and then tokenized using the `tokenize()` function. This splits the input string using whitespace as a delimiter.

Then the `execute` function forks a child process from the parent process and executes our given command.

This shell also runs “`exit`” and “`cd`” commands which I had to define on my own.

I also added a **colour** to the shell prompt because it looks nice.



```
gcc part_a.c -o part_a.out && ./part_a.out
chaitanya_bisht@12040450 $ ls
part_a.c  part_a.out  part_b.c  part_c.c
chaitanya_bisht@12040450 $ pwd
/home/chaitanya/Desktop/IIT/CS250/assignment 2/stage
chaitanya_bisht@12040450 $ mkdir test
chaitanya_bisht@12040450 $ cd test
chaitanya_bisht@12040450 $ pwd
/home/chaitanya/Desktop/IIT/CS250/assignment 2/stage/test
chaitanya_bisht@12040450 $ echo hello
hello
chaitanya_bisht@12040450 $ cat part_a.c
cat: part_a.c: No such file or directory
chaitanya_bisht@12040450 $ cd ..
chaitanya_bisht@12040450 $ cat part_a.c
#include <string.h>
#include <errno.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
#include <ctype.h>
#include <stdlib.h>

#define MAX_LEN 512 // The maximum length command

int RUN_STATUS = 1;
int WAIT_STATUS = 1;

void execute(char *args[]){
```

PART B : Extending shell to support “|” and “&&” features

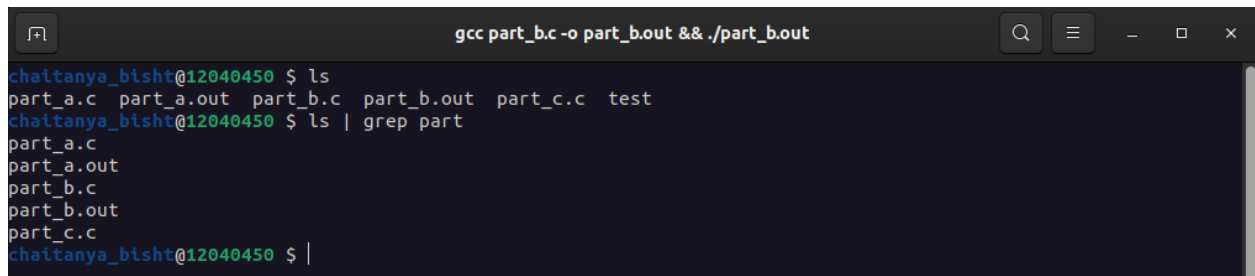
Execute the shell using the following commands:

```
$ gcc partB.c -o partB.out && ./partB.out
```

Here I extend the shell to support pipe (|) and AND(&&) features.

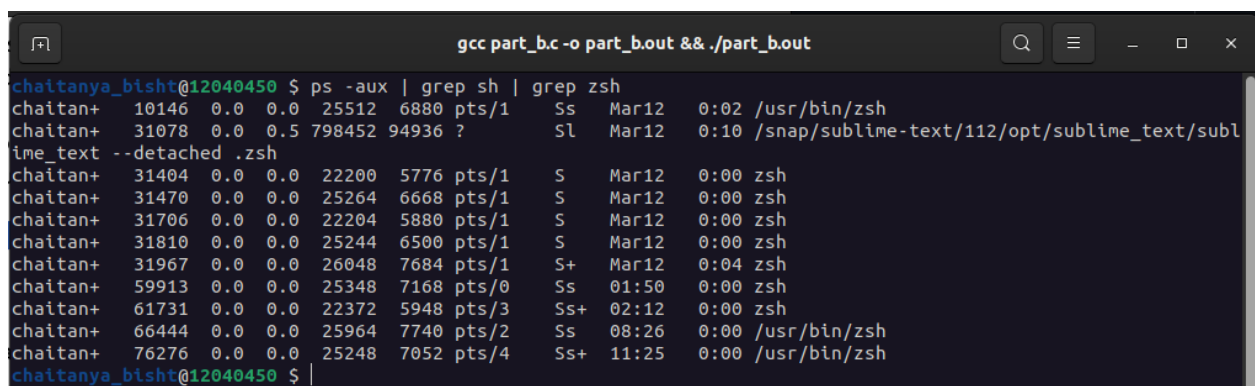
Just for extras I also added in the support for command separator (;) feature.

Piping: For piping I had to create two separate file descriptors, one for input and one for output. Below is the demonstration of the piping.



```
gcc part_b.c -o part_b.out && ./part_b.out
chaitanya_bisht@12040450 $ ls
part_a.c part_a.out part_b.c part_b.out part_c.c test
chaitanya_bisht@12040450 $ ls | grep part
part_a.c
part_a.out
part_b.c
part_b.out
part_c.c
chaitanya_bisht@12040450 $ |
```

It also supports multiple pipes.



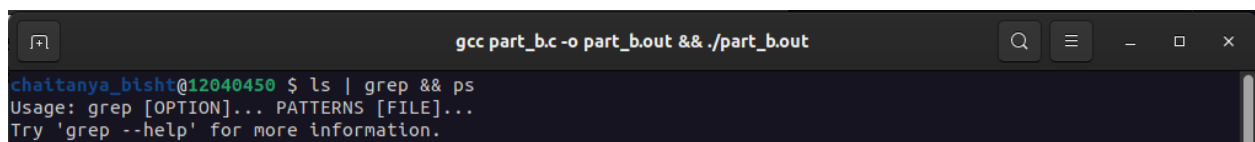
```
gcc part_b.c -o part_b.out && ./part_b.out
chaitanya_bisht@12040450 $ ps -aux | grep sh | grep zsh
chaitan+ 10146 0.0 0.0 25512 6880 pts/1 Ss Mar12 0:02 /usr/bin/zsh
chaitan+ 31078 0.0 0.5 798452 94936 ? Sl Mar12 0:10 /snap/sublime-text/112/opt/sublime_text/subl
ime_text --detached .zsh
chaitan+ 31404 0.0 0.0 22200 5776 pts/1 S Mar12 0:00 zsh
chaitan+ 31470 0.0 0.0 25264 6668 pts/1 S Mar12 0:00 zsh
chaitan+ 31706 0.0 0.0 22204 5880 pts/1 S Mar12 0:00 zsh
chaitan+ 31810 0.0 0.0 25244 6500 pts/1 S Mar12 0:00 zsh
chaitan+ 31967 0.0 0.0 26048 7684 pts/1 S+ Mar12 0:04 zsh
chaitan+ 59913 0.0 0.0 25348 7168 pts/0 Ss 01:50 0:00 zsh
chaitan+ 61731 0.0 0.0 22372 5948 pts/3 Ss+ 02:12 0:00 zsh
chaitan+ 66444 0.0 0.0 25964 7740 pts/2 Ss 08:26 0:00 /usr/bin/zsh
chaitan+ 76276 0.0 0.0 25248 7052 pts/4 Ss+ 11:25 0:00 /usr/bin/zsh
chaitanya_bisht@12040450 $ |
```

AND(&&): For && I had to check the exit code of the previous command before executing the next command.

```
$ command1 && command2 && command3
```

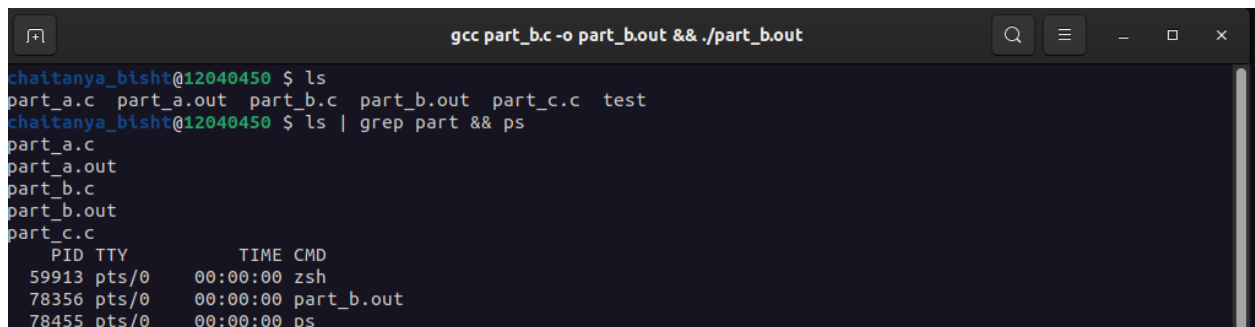
Here if command2 fails, then we do **NOT** execute the command3 and simply terminate.

In the screenshot below, we can see that since the first command (`ls | grep`) did not execute successfully, the `ps` command did not run

A terminal window titled "gcc part_b.c -o part_b.out && ./part_b.out" shows a user prompt "chaitanya_bisht@12040450". The user enters "ls | grep && ps". The terminal output shows the "grep" command's usage information, indicating it failed to find any matches. The "ps" command is not executed.

```
chaitanya_bisht@12040450 $ ls | grep && ps
Usage: grep [OPTION]... PATTERNS [FILE]...
Try 'grep --help' for more information.
```

Now since the command (`ls | grep part`) runs successfully, the `ps` command is also executed.

A terminal window titled "gcc part_b.c -o part_b.out && ./part_b.out" shows a user prompt "chaitanya_bisht@12040450". The user enters "ls", which lists files "part_a.c", "part_a.out", "part_b.c", "part_b.out", "part_c.c", and "test". Then, the user enters "ls | grep part && ps". The terminal output shows the files listed, followed by the output of the "ps" command, which lists the current processes.

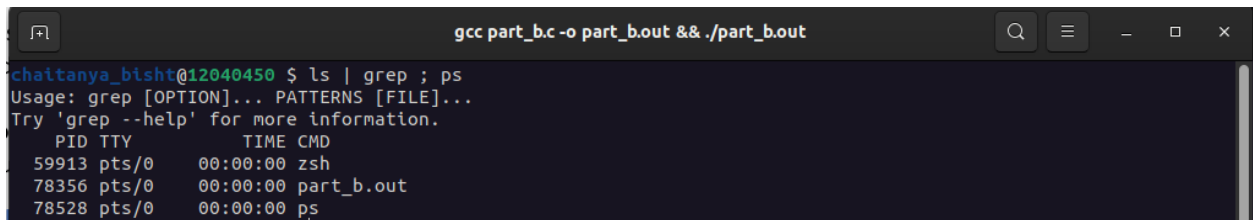
```
chaitanya_bisht@12040450 $ ls
part_a.c  part_a.out  part_b.c  part_b.out  part_c.c  test
chaitanya_bisht@12040450 $ ls | grep part && ps
part_a.c
part_a.out
part_b.c
part_b.out
part_c.c
  PID TTY          TIME CMD
 59913 pts/0    00:00:00 zsh
 78356 pts/0    00:00:00 part_b.out
 78455 pts/0    00:00:00 ps
```

Extra: Command Separator(;) : For ; we simply execute all the commands regardless of the exit code of the previous command.

```
$ command1 ; command2 ; command3
```

Here if command2 fails, even then the command3 is executed.

In the screenshot below we can see that the first command (ls | grep) fails but still we execute the ps command.

A terminal window titled 'gcc part_b.c -o part_b.out && ./part_b.out'. The user 'chaitanya_bisht@12040450' enters the command 'ls | grep ; ps'. The output shows the usage for 'grep' followed by the 'ps' command output, which lists running processes: '59913 pts/0 00:00:00 zsh', '78356 pts/0 00:00:00 part_b.out', and '78528 pts/0 00:00:00 ps'.

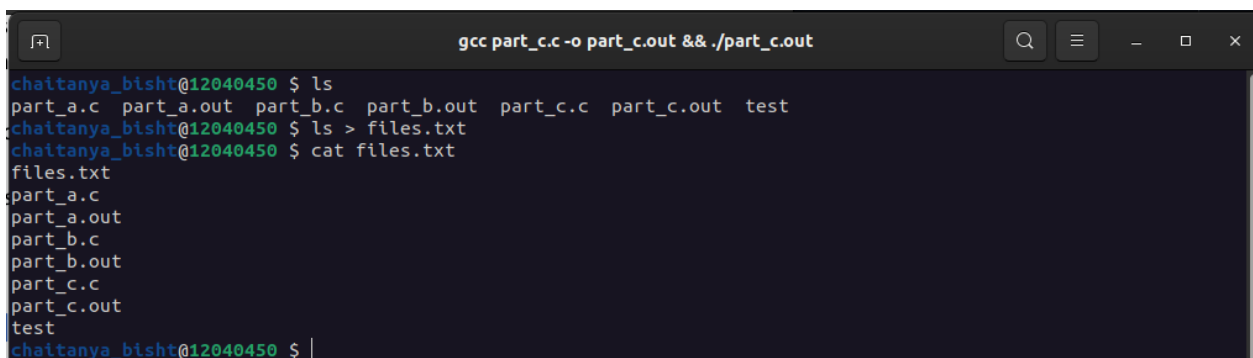
PART C: Extending shell to support the redirection feature

Execute the shell using the following commands:

```
$ gcc partC.c -o partC.out && ./partC.out
```

Here I added the support to redirect files in and out of the shell to files. I had defined two functions, rin() and rout() for this purpose which handles the file I/O.

Here I redirected the output of ls command to a file named files.txt.

A terminal window titled 'gcc part_c.c -o part_c.out && ./part_c.out'. The user 'chaitanya_bisht@12040450' enters a series of commands: 'ls', 'ls > files.txt', and 'cat files.txt'. The output of 'ls' lists files: 'part_a.c part_a.out part_b.c part_b.out part_c.c part_c.out test'. The output of 'cat files.txt' shows the same list of files, confirming that the output of 'ls' was successfully redirected to 'files.txt'.