GTU Department of Computer Engineering
CSE 344 - Spring 2023
Homework 2 Report

Mehmet Mete Şamlıoğlu
200104004093

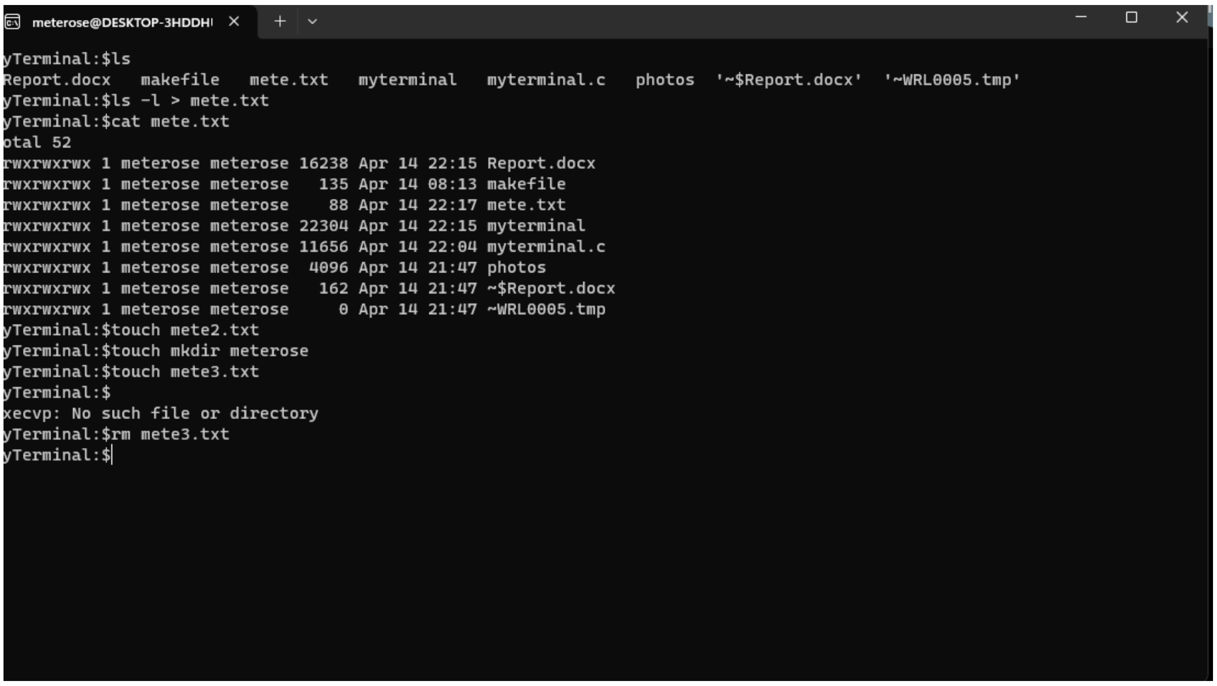
Problem Solution Approach

I started analyzing the problem and decided to do a parser first. I spent most of my time on the parser but in the end, I realized is my parser is so complex for the problem. I mean only the parsing of the command would be enough but I parsed every piece of command such as “ls” and “-l” etc. It took so much time. After that I started executing one line without a pipe, my plan was to transform that into a multiline task. I handled the single-line execution but when it comes to piping, I realized that is not related to single-line command execution. I started writing a particular function for performing piping. I handled a case where we have only one pipe, my plan was to find an algorithm to transform that only one-pipe solution into multiple-pipe problems but this was the part that I was not able to handle. I tried to fix my algorithm but it was not sufficient so I decided to perform only one pipe. There were some materials on the internet about how to solve it but since they are so complicated and also not my answers, I did not use them.

In conclusion, my code performs 2 commands in a single at most, redirections and other signal handling are performed in the code.

The running commands

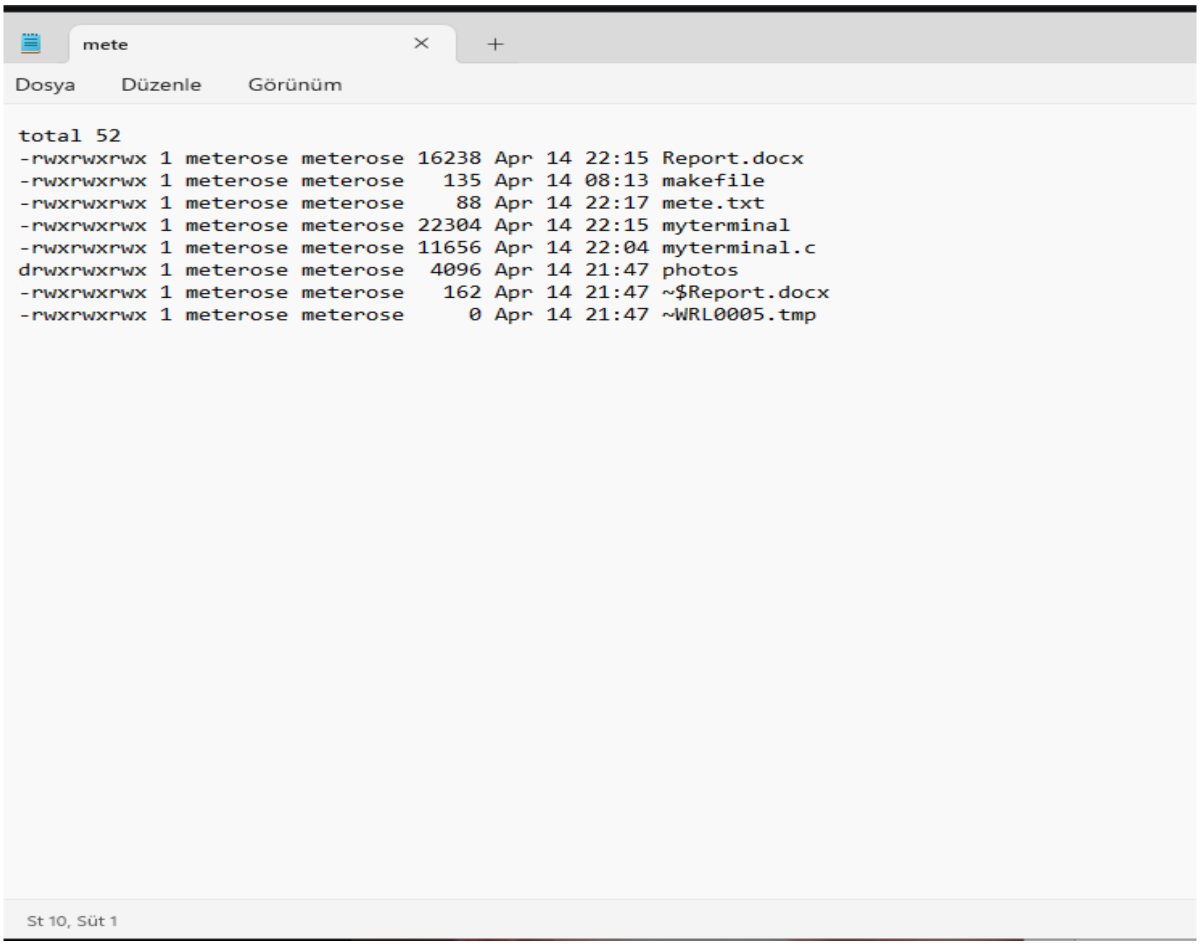
1-) Running single command



```
yTerminal:$ls
Report.docx   makefile   mete.txt   myterminal   myterminal.c   photos  '~$Report.docx'  '~WRL0005.tmp'
yTerminal:$ls -l > mete.txt
yTerminal:$cat mete.txt
otal 52
rwxrwxrwx 1 meterose meterose 16238 Apr 14 22:15 Report.docx
rwxrwxrwx 1 meterose meterose   135 Apr 14 08:13 makefile
rwxrwxrwx 1 meterose meterose    88 Apr 14 22:17 mete.txt
rwxrwxrwx 1 meterose meterose 22304 Apr 14 22:15 myterminal
rwxrwxrwx 1 meterose meterose 11656 Apr 14 22:04 myterminal.c
rwxrwxrwx 1 meterose meterose  4096 Apr 14 21:47 photos
rwxrwxrwx 1 meterose meterose   162 Apr 14 21:47 ~$Report.docx
rwxrwxrwx 1 meterose meterose     0 Apr 14 21:47 ~WRL0005.tmp
yTerminal:$touch mete2.txt
yTerminal:$touch mkdir meterose
yTerminal:$touch mete3.txt
yTerminal:$
xecvp: No such file or directory
yTerminal:$rm mete3.txt
yTerminal:$
```

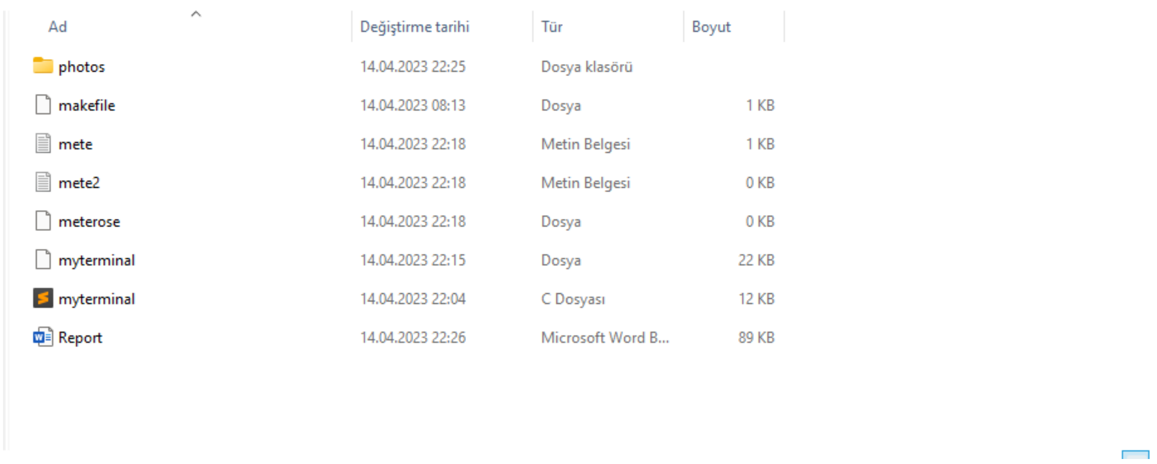
It is the single line command, by using it you can perform every(in bin/sh) operation including redirections, it works just as expected. In the first line by using the redirection symbol(>) it writes ls -l content to the mete.txt file. I performed it by using the dup2() function. First I created a file descriptor which is file_fd, and after that whenever a parser detected ‘>’ I redirected

STDOUT_FILENO to the mete.txt file. Thanks to that after execution of command, the output will be redirected to mete.txt



```
total 52
-rwxrwxrwx 1 meterose meterose 16238 Apr 14 22:15 Report.docx
-rwxrwxrwx 1 meterose meterose   135 Apr 14 08:13 makefile
-rwxrwxrwx 1 meterose meterose    88 Apr 14 22:17 mete.txt
-rwxrwxrwx 1 meterose meterose 22304 Apr 14 22:15 myterminal
-rwxrwxrwx 1 meterose meterose 11656 Apr 14 22:04 myterminal.c
drwxrwxrwx 1 meterose meterose  4096 Apr 14 21:47 photos
-rwxrwxrwx 1 meterose meterose   162 Apr 14 21:47 ~$Report.docx
-rwxrwxrwx 1 meterose meterose     0 Apr 14 21:47 ~WRL0005.tmp
```

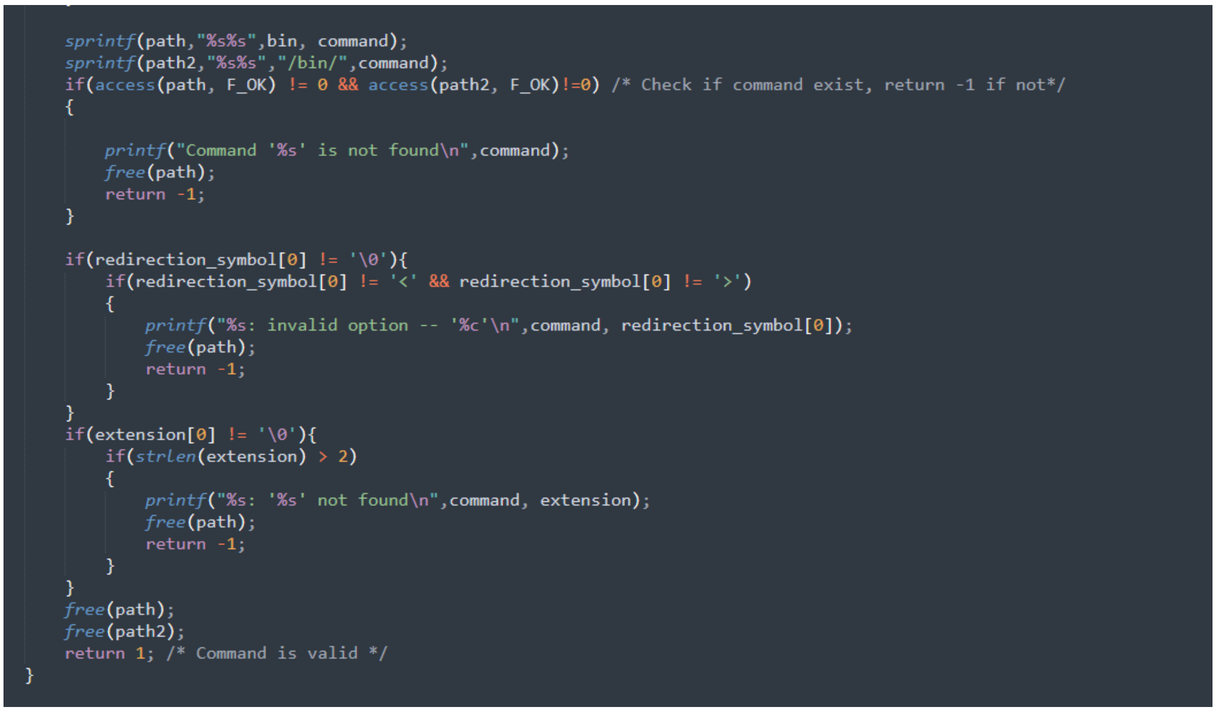
- As you can see the output of ls -l is written to the file name mete.txt



Ad	Değiştirme tarihi	Tür	Boyut
photos	14.04.2023 22:25	Dosya klasörü	
makefile	14.04.2023 08:13	Dosya	1 KB
mete	14.04.2023 22:18	Metin Belgesi	1 KB
mete2	14.04.2023 22:18	Metin Belgesi	0 KB
meterose	14.04.2023 22:18	Dosya	0 KB
myterminal	14.04.2023 22:15	Dosya	22 KB
myterminal	14.04.2023 22:04	C Dosyası	12 KB
Report	14.04.2023 22:26	Microsoft Word B...	89 KB

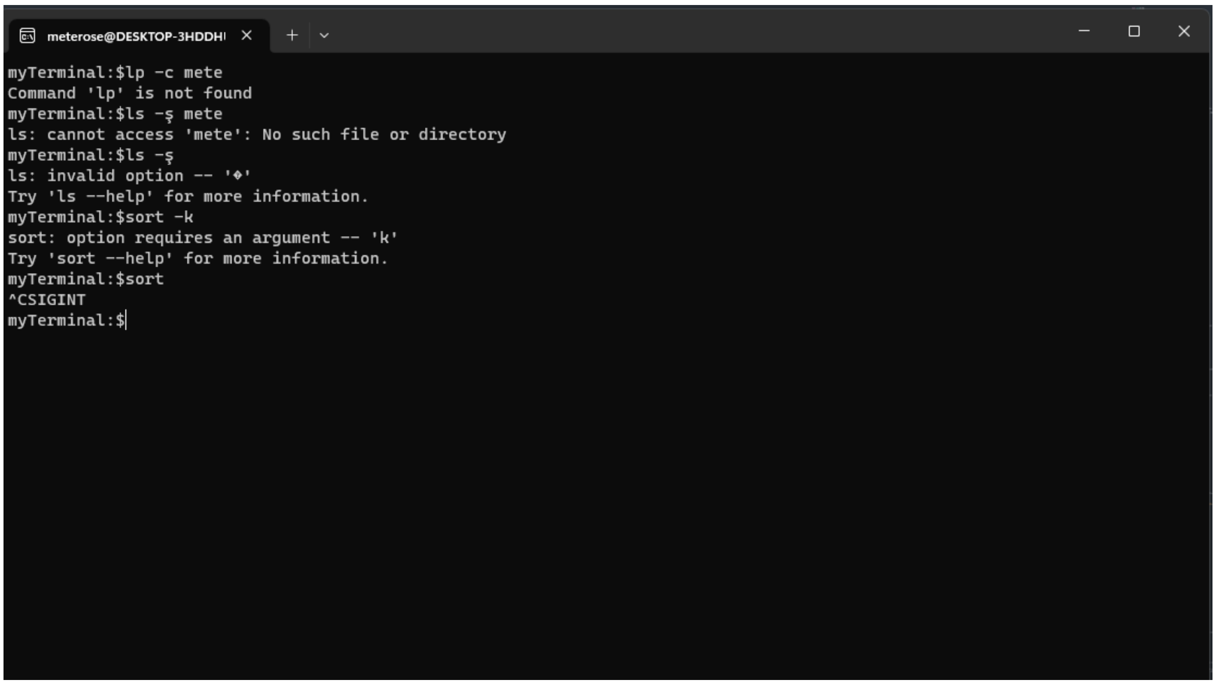
As you can see, touch, rm and mkdir functions also work as expected.

The second problem that I encountered is what is to be done whenever an incorrect command string is entered or how to check if the command exists in /bin/sh or is the file exists. To solve that problem I wrote this piece of code;



```c
    sprintf(path,"%s%s",bin, command);
    sprintf(path2,"%s%s","/bin/",command);
    if(access(path, F_OK) != 0 && access(path2, F_OK)!=0) /* Check if command exist, return -1 if not*/
    {

        printf("Command '%s' is not found\n",command);
        free(path);
        return -1;
    }

    if(redirection_symbol[0] != '\0'){
        if(redirection_symbol[0] != '<' && redirection_symbol[0] != '>')
        {
            printf("%s: invalid option -- '%c'\n",command, redirection_symbol[0]);
            free(path);
            return -1;
        }
    }
    if(extension[0] != '\0'){
        if(strlen(extension) > 2)
        {
            printf("%s: '%s' not found\n",command, extension);
            free(path);
            return -1;
        }
    }
    free(path);
    free(path2);
    return 1; /* Command is valid */
}
```

Since the execvp function handles all other input errors, there was not much to be handled. I just checked if the file exists or redirection symbol is correct etc.



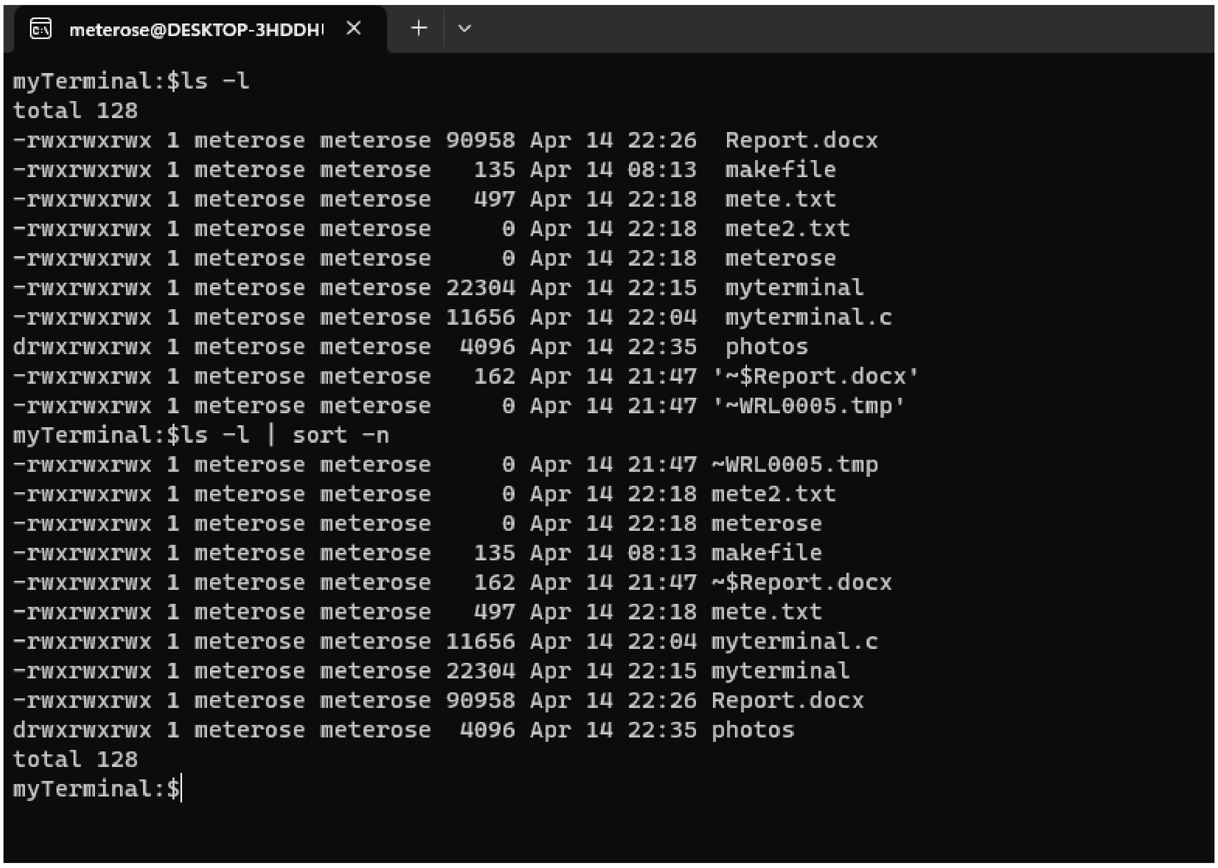
```
myTerminal:$lp -c mete
Command 'lp' is not found
myTerminal:$ls -ş mete
ls: cannot access 'mete': No such file or directory
myTerminal:$ls -ş
ls: invalid option -- '�'
Try 'ls --help' for more information.
myTerminal:$sort -k
sort: option requires an argument -- 'k'
Try 'sort --help' for more information.
myTerminal:$sort
^CSIGINT
myTerminal:$
```

As you can see above, it detects misspelled commands and put the related usage information/error reason on the screen.

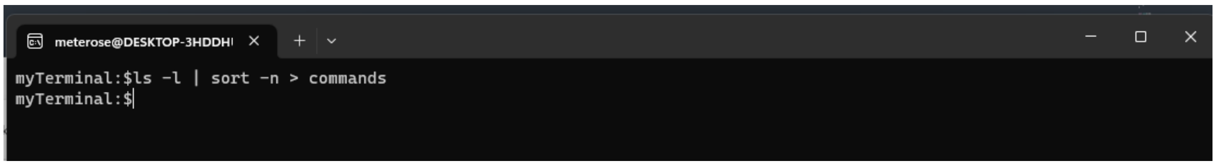
Multiple Commands

As I mentioned in the problem-solution approach section, in the beginning, I was able to perform only one piping, and as a result that my terminal can just perform 2 commands in a single line. If there are 2 commands it performs every operation including redirections

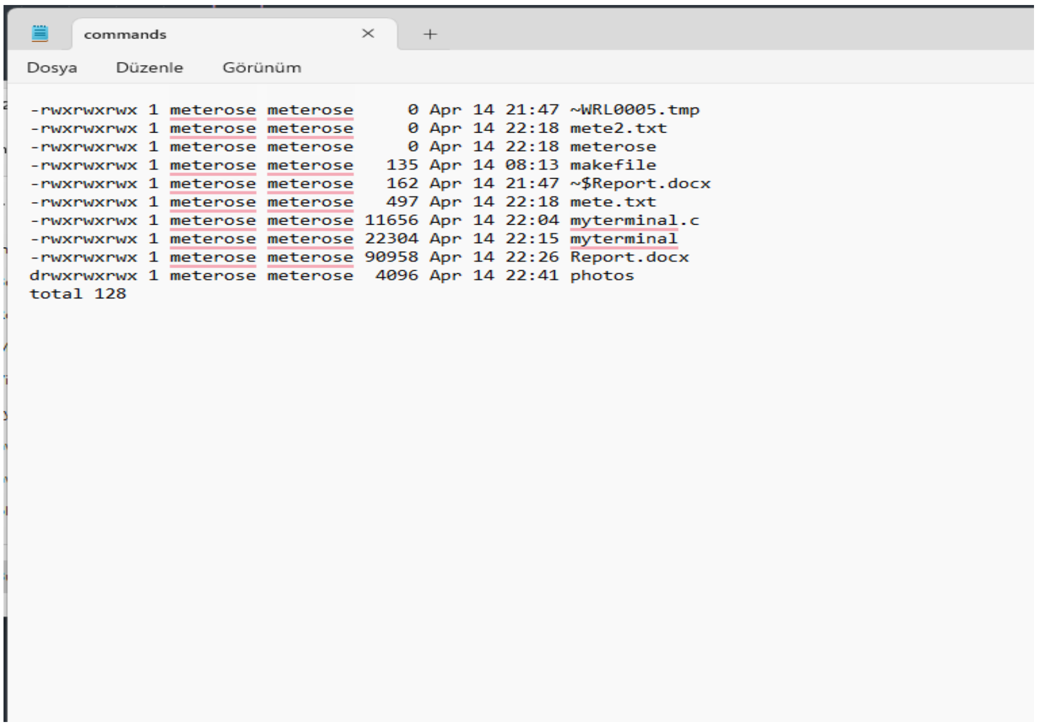
ls -l | sort -n


```
myTerminal:$ls -l
total 128
-rwxrwxrwx 1 meterose meterose 90958 Apr 14 22:26  Report.docx
-rwxrwxrwx 1 meterose meterose   135 Apr 14 08:13  makefile
-rwxrwxrwx 1 meterose meterose   497 Apr 14 22:18  mete.txt
-rwxrwxrwx 1 meterose meterose     0 Apr 14 22:18  mete2.txt
-rwxrwxrwx 1 meterose meterose     0 Apr 14 22:18  meterose
-rwxrwxrwx 1 meterose meterose 22304 Apr 14 22:15  myterminal
-rwxrwxrwx 1 meterose meterose 11656 Apr 14 22:04  myterminal.c
drwxrwxrwx 1 meterose meterose  4096 Apr 14 22:35  photos
-rwxrwxrwx 1 meterose meterose   162 Apr 14 21:47 '~$Report.docx'
-rwxrwxrwx 1 meterose meterose     0 Apr 14 21:47 '~WRL0005.tmp'
myTerminal:$ls -l | sort -n
-rwxrwxrwx 1 meterose meterose     0 Apr 14 21:47 ~WRL0005.tmp
-rwxrwxrwx 1 meterose meterose     0 Apr 14 22:18 mete2.txt
-rwxrwxrwx 1 meterose meterose     0 Apr 14 22:18 meterose
-rwxrwxrwx 1 meterose meterose   135 Apr 14 08:13 makefile
-rwxrwxrwx 1 meterose meterose   162 Apr 14 21:47 ~$Report.docx
-rwxrwxrwx 1 meterose meterose   497 Apr 14 22:18 mete.txt
-rwxrwxrwx 1 meterose meterose 11656 Apr 14 22:04 myterminal.c
-rwxrwxrwx 1 meterose meterose 22304 Apr 14 22:15 myterminal
-rwxrwxrwx 1 meterose meterose 90958 Apr 14 22:26 Report.docx
drwxrwxrwx 1 meterose meterose  4096 Apr 14 22:35 photos
total 128
myTerminal:$
```

As you can see above when you command only ls it prints them arbitrarily. However, in the second example, the output of ls -l goes to sort -n as input, and the sorted version of ls is being the original output.


```
myTerminal:$ls -l | sort -n > commands
myTerminal:$
```

The sorted ls -l will be written to the file named commands

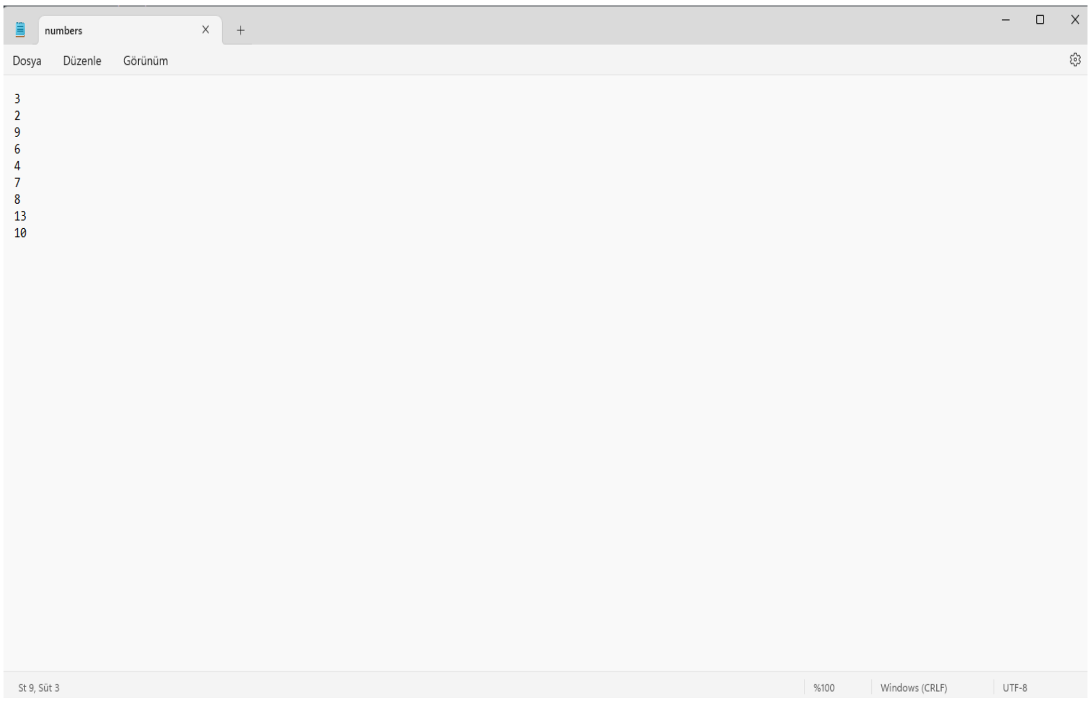


```
-rwxrwxrwx 1 meterose meterose     0 Apr 14 21:47 ~WRL0005.tmp
-rwxrwxrwx 1 meterose meterose     0 Apr 14 22:18 mete2.txt
-rwxrwxrwx 1 meterose meterose     0 Apr 14 22:18 meterose
-rwxrwxrwx 1 meterose meterose   135 Apr 14 08:13 makefile
-rwxrwxrwx 1 meterose meterose   162 Apr 14 21:47 ~$Report.docx
-rwxrwxrwx 1 meterose meterose   497 Apr 14 22:18 mete.txt
-rwxrwxrwx 1 meterose meterose 11656 Apr 14 22:04 myterminal.c
-rwxrwxrwx 1 meterose meterose 22304 Apr 14 22:15 myterminal
-rwxrwxrwx 1 meterose meterose 90958 Apr 14 22:26 Report.docx
drwxrwxrwx 1 meterose meterose  4096 Apr 14 22:41 photos
total 128
```

- As you can see it is written to the file succesfully.

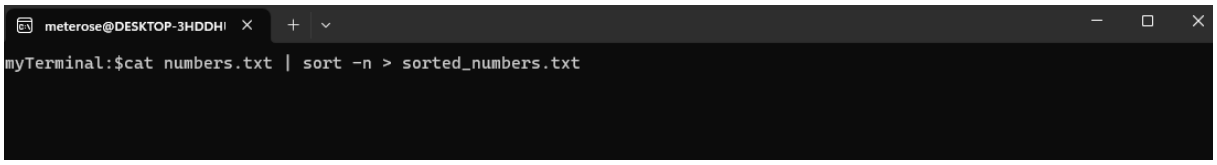
Let’s try getting the input from a file, perform an operation on i,t and send it to the other command as input;

I will create a text file that contains random numbers.



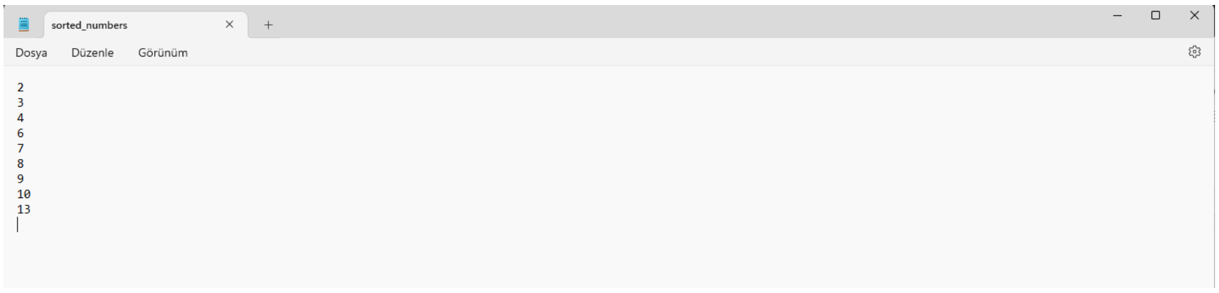
```
3
2
9
6
4
7
8
13
10
```

Now I will sort them and send the output of it as an input to a cat function as following;



```
myTerminal:$cat numbers.txt | sort -n > sorted_numbers.txt
```

Lets look at the sorted_numbers.txt

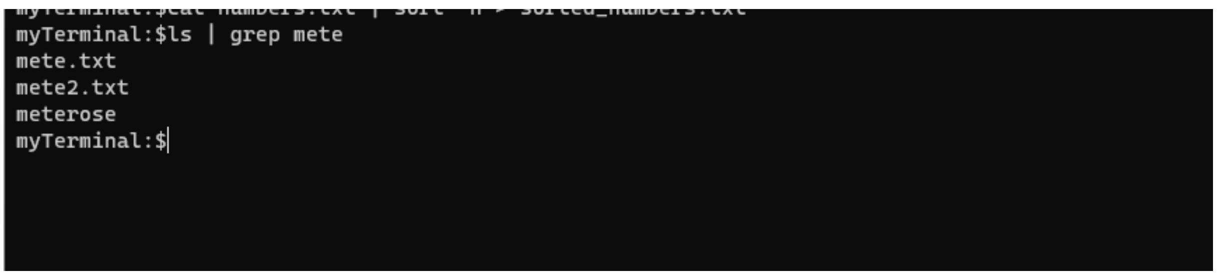


```
2
3
4
6
7
8
9
10
13
```

- What happens here is sort function takes the content of numbers.txt as input from the previous comment and performed the operation on it and redirected to output to the sorted_numbers.txt instead of STDOUT.

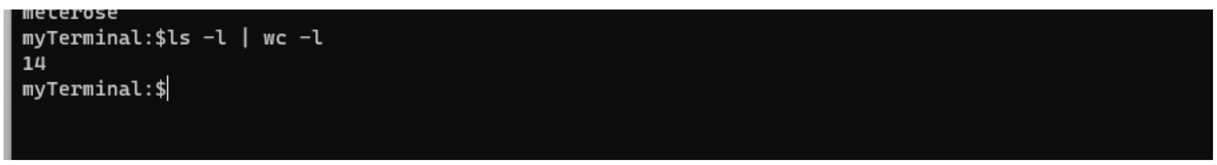
Following you can see some other commands;

- ls | grep mete



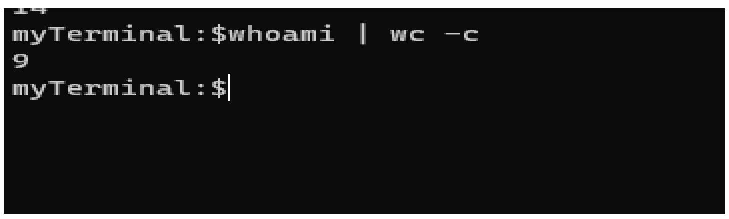
```
myTerminal:$ls | grep mete
mete.txt
mete2.txt
meterose
myTerminal:$
```

- ls -l | wc -l



```
myTerminal:$ls -l | wc -l
14
myTerminal:$
```

- whoami | wc -c

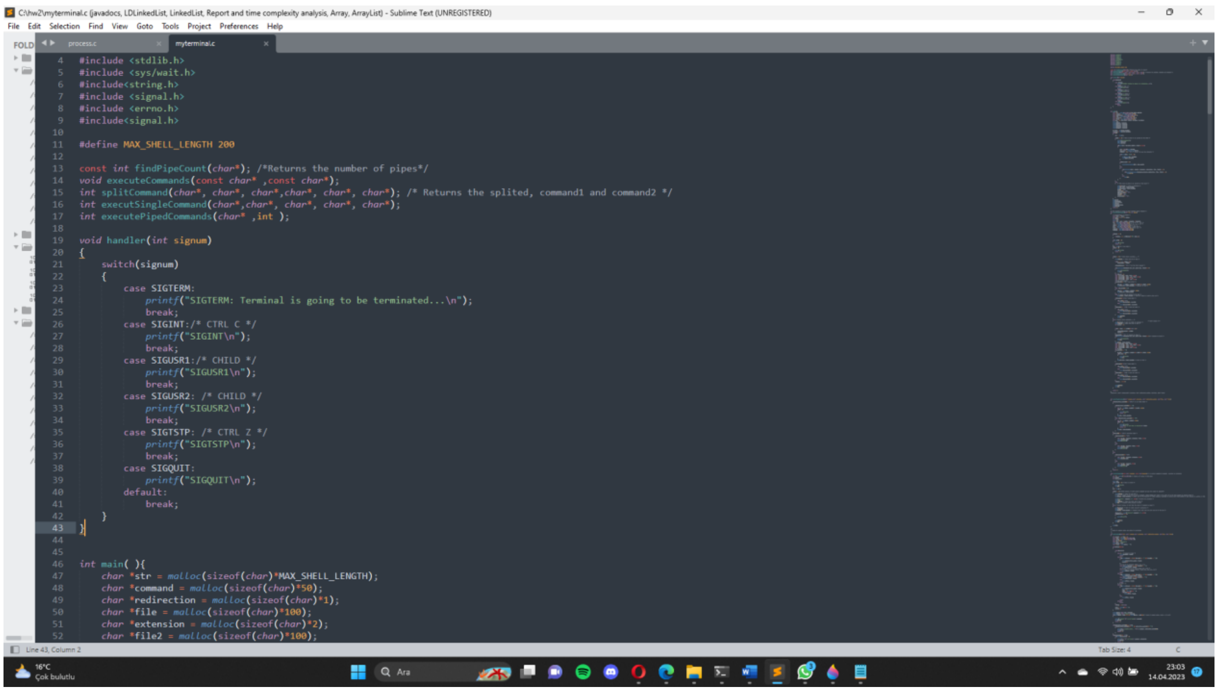


```
myTerminal:$whoami | wc -c
9
myTerminal:$
```

So as you can see above, it performs all operations including redirection if there is 2 commands, I tried to transform this algorithm to perform more than one command in a single line but I could handle it.

Signals

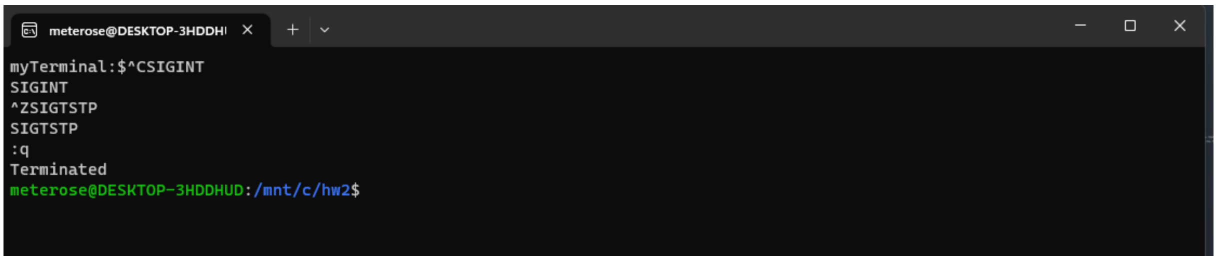
Following you can see my signal handler and the signals that I handled in this code.


C:\hw2\myterminal.c (javadocs, LDLinkedList, LinkedList, Report and time complexity analysis, Array, ArrayList) - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
process.c
myterminal.c
Line 43, Column 2

```c
#include <stdlib.h>
#include <sys/wait.h>
#include<string.h>
#include <signal.h>
#include <errno.h>
#include<signal.h>

#define MAX_SHELL_LENGTH 200

const int findPipeCount(char*); /*Returns the number of pipes*/
void executeCommands(const char* ,const char*);
int splitCommand(char*, char*, char*,char*, char*, char*); /* Returns the splited, command1 and command2 */
int executSingleCommand(char*,char*, char*, char*, char*);
int executePipedCommands(char* ,int );

void handler(int signum)
{
    switch(signum)
    {
        case SIGTERM:
            printf("SIGTERM: Terminal is going to be terminated...\n");
            break;
        case SIGINT:/* CTRL C */
            printf("SIGINT\n");
            break;
        case SIGUSR1:/* CHILD */
            printf("SIGUSR1\n");
            break;
        case SIGUSR2: /* CHILD */
            printf("SIGUSR2\n");
            break;
        case SIGTSTP: /* CTRL Z */
            printf("SIGTSTP\n");
            break;
        case SIGQUIT:
            printf("SIGQUIT\n");
        default:
            break;
    }
}


int main( ){
    char *str = malloc(sizeof(char)*MAX_SHELL_LENGTH);
    char *command = malloc(sizeof(char)*50);
    char *redirection = malloc(sizeof(char)*1);
    char *file = malloc(sizeof(char)*100);
    char *extension = malloc(sizeof(char)*2);
    char *file2 = malloc(sizeof(char)*100);
```

I handled SIGINT (ctrl -c), SIGUSR1 and SIGUSR2 (child), SIGTSTP(ctrl z), SIGQUIT, and SIGTERM. Instead of using SIGTERM to print something I used it to terminate the program whenever ":q" occurs. So whenever a string ":q" is entered by the user I send a SIGTERM signal and it terminates the program. I thought this is an efficient way to use it.

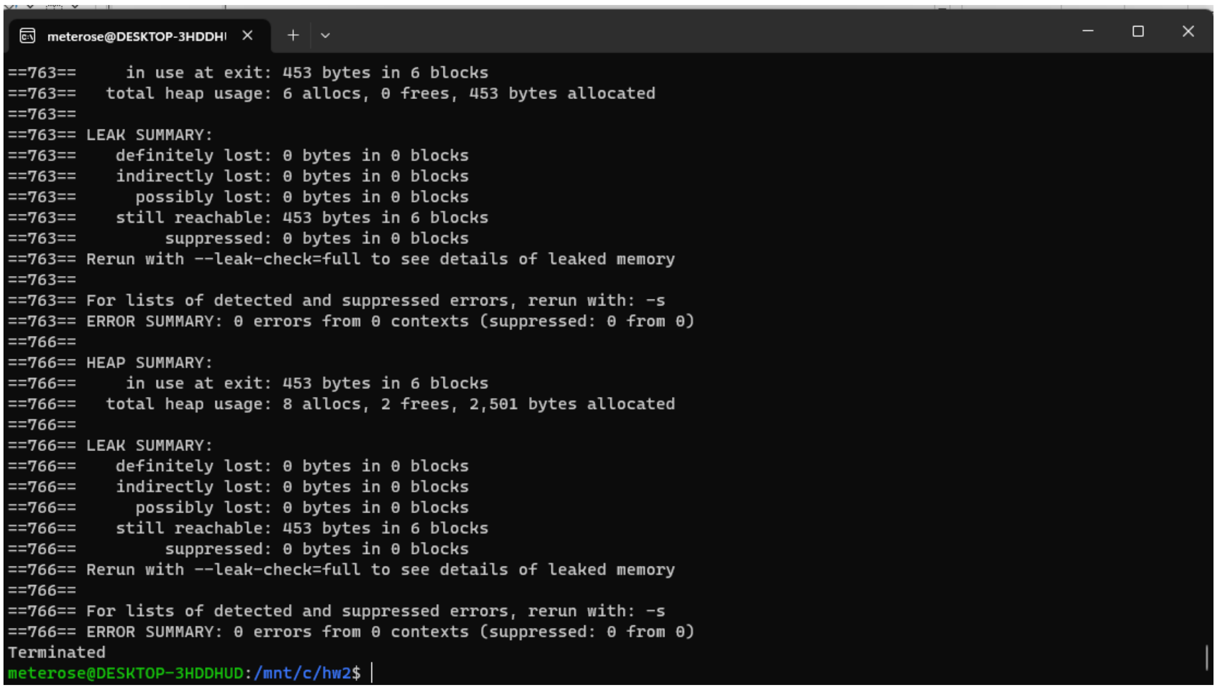
Following you can see the outputs for each signal



```
myTerminal:$^CSIGINT
SIGINT
^ZSIGTSTP
SIGTSTP
:q
Terminated
meterose@DESKTOP-3HDDHUD:/mnt/c/hw2$
```

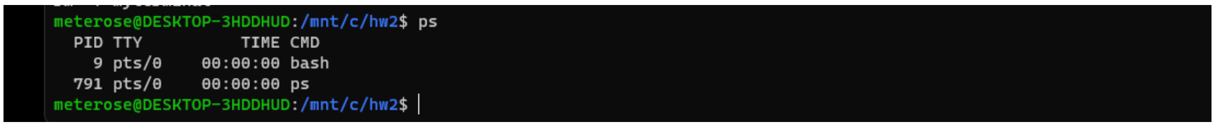
As you can see it prints what is supposed to print.

Valgrind Results



```
==763==     in use at exit: 453 bytes in 6 blocks
==763==   total heap usage: 6 allocs, 0 frees, 453 bytes allocated
==763==
==763== LEAK SUMMARY:
==763==    definitely lost: 0 bytes in 0 blocks
==763==    indirectly lost: 0 bytes in 0 blocks
==763==      possibly lost: 0 bytes in 0 blocks
==763==    still reachable: 453 bytes in 6 blocks
==763==         suppressed: 0 bytes in 0 blocks
==763== Rerun with --leak-check=full to see details of leaked memory
==763==
==763== For lists of detected and suppressed errors, rerun with: -s
==763== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
==766==
==766== HEAP SUMMARY:
==766==     in use at exit: 453 bytes in 6 blocks
==766==   total heap usage: 8 allocs, 2 frees, 2,501 bytes allocated
==766==
==766== LEAK SUMMARY:
==766==    definitely lost: 0 bytes in 0 blocks
==766==    indirectly lost: 0 bytes in 0 blocks
==766==      possibly lost: 0 bytes in 0 blocks
==766==    still reachable: 453 bytes in 6 blocks
==766==         suppressed: 0 bytes in 0 blocks
==766== Rerun with --leak-check=full to see details of leaked memory
==766==
==766== For lists of detected and suppressed errors, rerun with: -s
==766== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
Terminated
meterose@DESKTOP-3HDDHUD:/mnt/c/hw2$
```

Zombies check



```
meterose@DESKTOP-3HDDHUD:/mnt/c/hw2$ ps
  PID TTY          TIME CMD
    9 pts/0    00:00:00 bash
  791 pts/0    00:00:00 ps
meterose@DESKTOP-3HDDHUD:/mnt/c/hw2$
```

Mehmet Mete Şamlıoğlu

200104004093