CSE344 – System Programming
REPORT - HW 2

Furkan ÖZEV
161044036

BEFORE FORK:

The program to be developed will consist of two processes: the main process P1 and the subprocess P2 that will run simultaneously.

P1 must admit two command line parameters as paths to regular files:

```
./program -i inputPath -o outputPath
```

I used the getopt() library method for parsing command line arguments.
If the command line arguments are missing/invalid your program must print usage information and exit.

InputPath outputPath files are opened before the process forked. Then temproray file created via mkstemp. For other process, this temporary file reopen again. So there are 2 different file descriptor or cursor.

Also before the process forked, SIGUSR1, SIGUSR2, SIGTERM signals will be handler in the newly created handler function. This is catcher() in code.

AFTER FORK:

Then fork program. After fork I checked return value of fork. If return value is negative, forking is failed then print error message end exit program.
Otherwise there will be 2 new processes. These processes are called P1 and P2.

So, If return value is equal 0, this indicates that is child process, and call p2 function.

Otherwise, parent process call p1 function.
In order to avoid a problem when sending a signal from transaction P1 to transaction P2, P1 waits for a SIGUSR2 signal from P2 indicating that the transaction has started. So P1 can start after P2 starts. After that call p1() function to to fulfill its function.
In order not to kill the parent process before the child process is finished, the parent process waits for the child process to finish using the waitpid() function.

PROCESS 1:

Process 1 or Parent Process will read the contents of the file denoted by inputPath.
The input file tries to read 20 bytes each time until it ends.
Every couple of unsigned bytes it reads, will be interpreted as a 2D coordinate (x,y).

To convert unsigned bytes, I use casting like that **(int)((unsigned char) buf[i])**

The part where the calculations are made according to the assignment is the critical section.
Therefore, when SIGINT signal is received in this region, it is requested not to interrupt.
So it created new mask for signal, and it changed SIGINT handler.
If SIGINT signal comes, it will be handler in the function called catcher.

After that calculations can start. It’ll apply the least squares method to the corresponding 2D coordinates and calculate the line equation (ax+b) that fits them.

Calculation Steps:

1- For each (x,y) point calculate x^2 and x.y
2- Sum all x, y, x^2 and xy
3- Then calculate a with formula: (N is number of points)
  a = (N.Σ(xy) – Σx.Σy) / (N.Σ(x^2) – (Σx)^2)
4- Calculate b:
  b = (Σy – a.Σx) / N

After calculation, these line equation will be added end of line.
As the calculation is finished, it must exit the critical section. Therefore, Signal mask and handler are set to default.

In order to avoid conflicts between P1 and P2, P1 locks the temporary file before starting the writing process.
Then the line added to the line equation starts writing to the temp file. While doing this process, it is checked in the loop to avoid interrupt division.
After writing is finished, the lock is removed.

Then the SIGUSR2 signal is sent to P2.
Indicates that there is new input for the child process.
If the child process is in suspend state and is waiting for this signal, it will continue to work when this signal arrives.

If the number of bytes in the input file is less than 20, no new line can be created, so the loop ends.

Then it print on screen how many bytes it has read as input, how many line equations it has estimated, and which signals where sent to P1 while it was in a critical section.

After finishing processing the contents of the input file, P1 will terminate gracefully by closing open file (input file and temporary file).

End of process, SIGUSR1 signal is sent to the child process to indicate that the Parent process is finished. Thus, the child process understands that no new input will come. In addition, when this signal is received, the child process understands that the parent process is done with the inputPath file and deletes it.

Then the parent process waits for the child process to finish.

PROCESS 2:

By putting P1 at suspend for SIGUSR2 at the beginning of the program, we provided early forks by P2, P1.
Since P2 has started, it must send the SIGUSR2 signal to P1. P2 will do this at the beginning of the process. But if the SIGUSR1 signal is sent before process 1 enters the suspend state, it cannot handle. to prevent this situation, also continuous signal is sent in loop.

It will read the contents of the temporary file created by P1, line by line. For this, it will start from the top, it looks for the first line available. It does this by searching for the '\ n' character. It always start read from top, because the line whose job is finished is deleted. Then, it delete the new line character at the end of the string.

Check if there is at least 1 line of input in the temporary file and the parent process is over, it means that there will be no new input. So it exits the loop.

Check if there is at least 1 line of input in the temporary file and the parent process is not over, it means new input can be arrive, So suspend SIGUSR1 or SIGUSR2 signal. If one of these signals arrived, then it will turn begin of loop. Then if this signal was SIGUSR1 then it will break loop because temp file is done and there is no more input, Otherwise, new input is come it will read in loop.

The values to be used in the calculation are kept in the array. If the array is too small for it, realloc is done and enlarged.

The part where the calculations are made according to the assignment is the critical section.
Therefore, when SIGINT signal is received in this region, it is requested not to interrupt.
So it created new mask for signal, and it changed SIGINT handler.
If SIGINT signal comes, it will be handler in the function called catcher.

After that calculations can start. it will calculate the mean absolute error (MAE),mean squared error (MSE) and root mean squared error (RMSE) between the coordinates and the estimated line.

Calculation Steps:

1- Get all coordinates. (x1,y1),…,(x10,y10)
2- Get line equation. ax+b
3- Each x coordinate is substituted in line equation.
4- Let's say this value is Q.
5- The Q value calculated for each coordinate is subtracted from the y coordinate.
6- The sum of the absolute values of these results gives the MAE.
  MAE = Σ(abs(Q[i]-y[i]))
7- And the square power of the values of these results gives the MSE.
  MSE = Σ(Q[i]-y[i])
8- **RMSE** is square root of MSE.

While performing calculations, all mathematical operations were realized with accuracy down to 3 decimal points as desired in the homework.

After calculations, for each line these result keep in arrays. Then this metrics will be wrote end of file.

As the calculation is finished, it must exit the critical section. Therefore, Signal mask and handler are set to default.

In order to avoid conflicts between P2 and any process, P2 locks the output file before starting the writing process.
Then the line added to the metrics starts writing to the output file. While doing this process, it is checked in the loop to avoid interrupt division.
After writing is finished, the lock is removed.

Then, it will then remove the line it read from the file.
For this reason, it get the size of temp file.
The bytes after the line to be deleted are copied.
The file is resized with the truncate () function.
Then the bytes copied are written to the file from the beginning of the file.
Such line will be deleted. After writing is finished, the lock is removed.

The same processes are repeated as long as there is an input.

If the input is not empty, mean and deviations calculations are made.

Mean:

1- Each metric type is summed up in itself and divided by the number of metric.
2- Thus, the means of each metric is found.
  Mean = Sum of all data for specific metric / Number of data

Mean Deviation:

1 - Each metric type is summed up in terms of absolute differences from the mean and divided by the number of metrics.

2- Thus, the mean deviations of each metric is found.

Mean Deviation = Σ(abs(mean – Metric[i])) / Number of data

Standart Deviation:

1 - Each metric type is summed up in terms of square power of differences from the mean and divided by the number of metrics. Then get the square root of these result.

2- Thus, the mean deviations of each metric is found.

X = Σ(mean – Metric[i])^2) / Number of data

Standart Deviation = $\sqrt{}$(X)

If P2 making sure that no more input will arrive,
It will print on screen for each error metric, its mean, mean deviation and standard deviation. It will free memory that are allocated.

After finishing processing the contents of the temporary file, P2 will terminate gracefully by closing open file (temporary file, output file).
Temporary file and input file can be deleted since P2 process is also completed.

SIGNAL HANDLER OR CATCHER:

SIGUSR1:

If the SIGUSR1 signal comes, the value of the global variable used for P2 becomes 0.
If this variable equal 0, parent process done, Otherwise parent process is working.
Then the input file is deleted.

* P1 sends SIGUSR1 to P2 to indicate that P1 is done.

SIGINT:

If there is a SIGINT signal in the critical section, 1 is assigned to the flag.
This flag will be use print signals.

SIGTERM:

In either P1 or P2, your processes must catch it and clean up after them before exiting gracefully, by closing open files, and removing the input and temporary files from disk.
Parent after these operations send kill signal to child.
Child after these operations send kill signal to parent.

SIGUSR2:

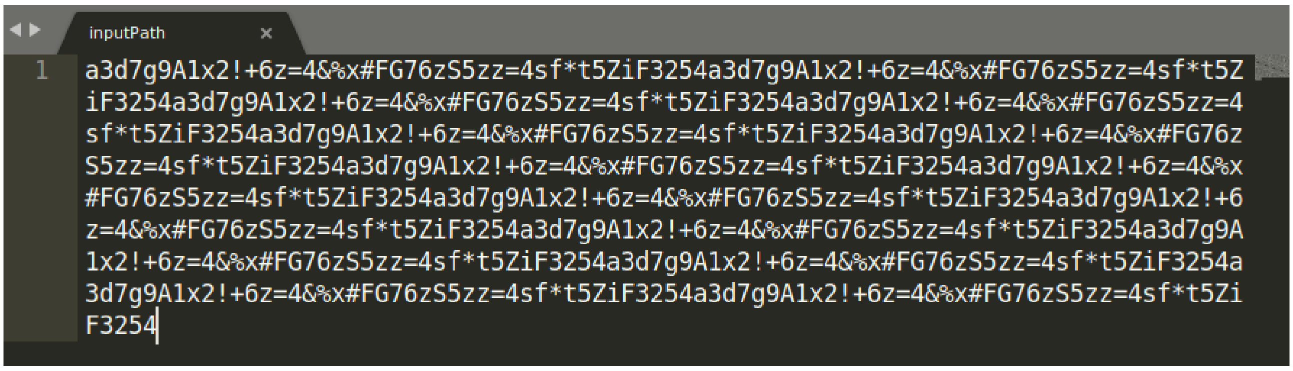
It is just a signal. There will be nothing to do.

* P1 sends the SIGUSR2 signal to P2 to indicate that the new input has arrived and must leave suspend.

* P2 sends the SIGUSR2 signal to P1, to indicate that the P2 process has started and must leave suspend.

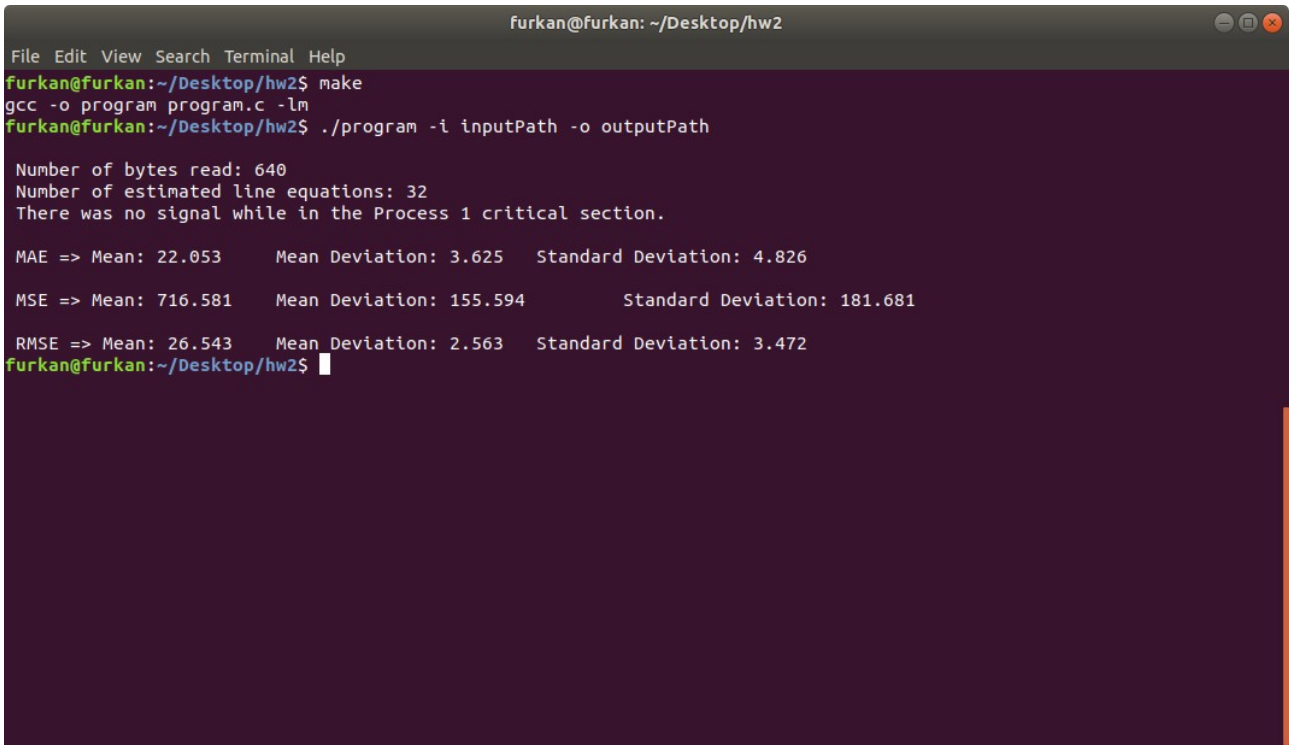
RUN EXAMPLE - 1:

INPUT FILE – A: (645 bytes)



```
a3d7g9A1x2!+6z=4&%x#FG76zS5zz=4sf*t5ZiF3254a3d7g9A1x2!+6z=4&%x#FG76zS5zz=4sf*t5Z
iF3254a3d7g9A1x2!+6z=4&%x#FG76zS5zz=4sf*t5ZiF3254a3d7g9A1x2!+6z=4&%x#FG76zS5zz=4
sf*t5ZiF3254a3d7g9A1x2!+6z=4&%x#FG76zS5zz=4sf*t5ZiF3254a3d7g9A1x2!+6z=4&%x#FG76z
S5zz=4sf*t5ZiF3254a3d7g9A1x2!+6z=4&%x#FG76zS5zz=4sf*t5ZiF3254a3d7g9A1x2!+6z=4&%x
#FG76zS5zz=4sf*t5ZiF3254a3d7g9A1x2!+6z=4&%x#FG76zS5zz=4sf*t5ZiF3254a3d7g9A1x2!+6
z=4&%x#FG76zS5zz=4sf*t5ZiF3254a3d7g9A1x2!+6z=4&%x#FG76zS5zz=4sf*t5ZiF3254a3d7g9A
1x2!+6z=4&%x#FG76zS5zz=4sf*t5ZiF3254a3d7g9A1x2!+6z=4&%x#FG76zS5zz=4sf*t5ZiF3254a
3d7g9A1x2!+6z=4&%x#FG76zS5zz=4sf*t5ZiF3254a3d7g9A1x2!+6z=4&%x#FG76zS5zz=4sf*t5Zi
F3254
```

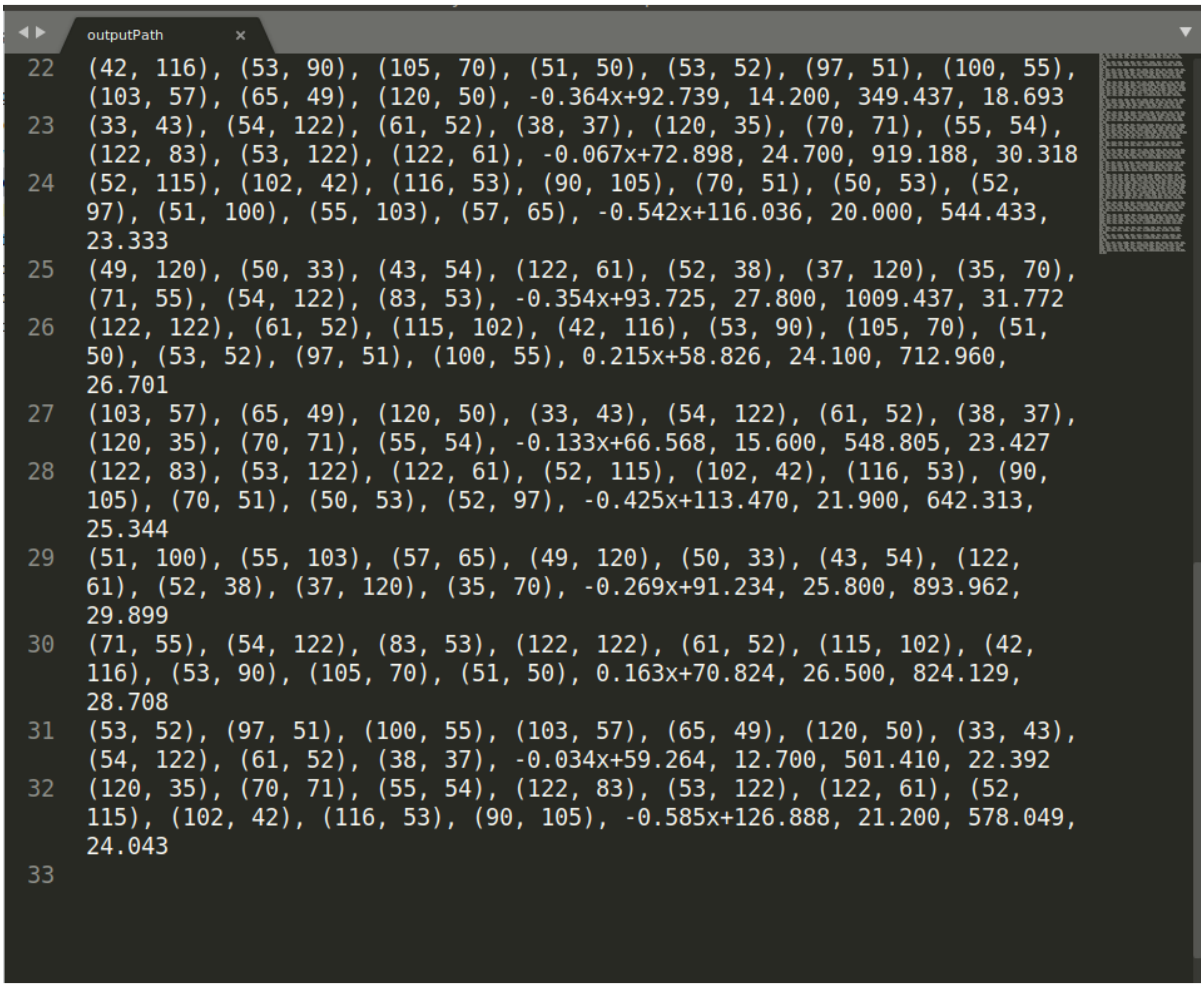
COMPILE & RUN:



furkan@furkan: ~/Desktop/hw2
File Edit View Search Terminal Help

```
furkan@furkan:~/Desktop/hw2$ make
gcc -o program program.c -lm
furkan@furkan:~/Desktop/hw2$ ./program -i inputPath -o outputPath

 Number of bytes read: 640
 Number of estimated line equations: 32
 There was no signal while in the Process 1 critical section.

 MAE => Mean: 22.053     Mean Deviation: 3.625   Standard Deviation: 4.826

 MSE => Mean: 716.581    Mean Deviation: 155.594         Standard Deviation: 181.681

 RMSE => Mean: 26.543    Mean Deviation: 2.563   Standard Deviation: 3.472
furkan@furkan:~/Desktop/hw2$ 
```

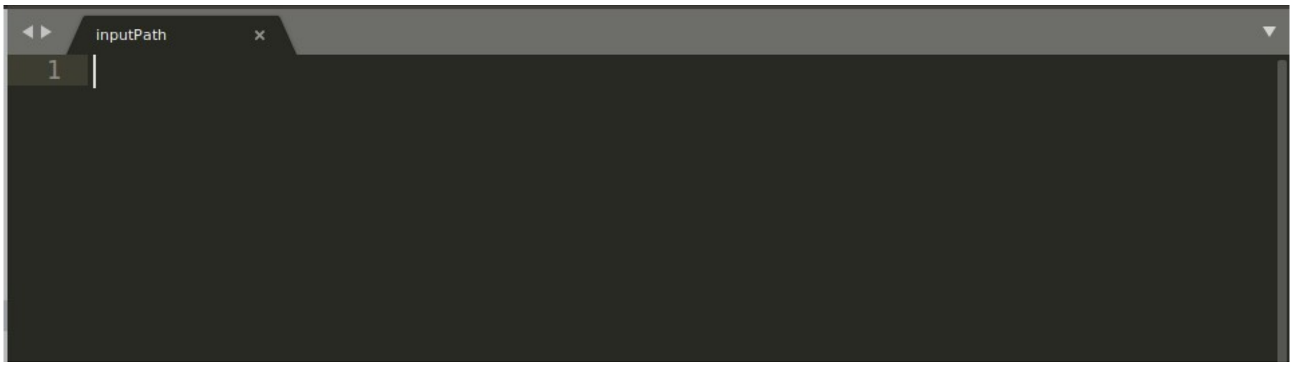
OUTPUT : (32 Equations)



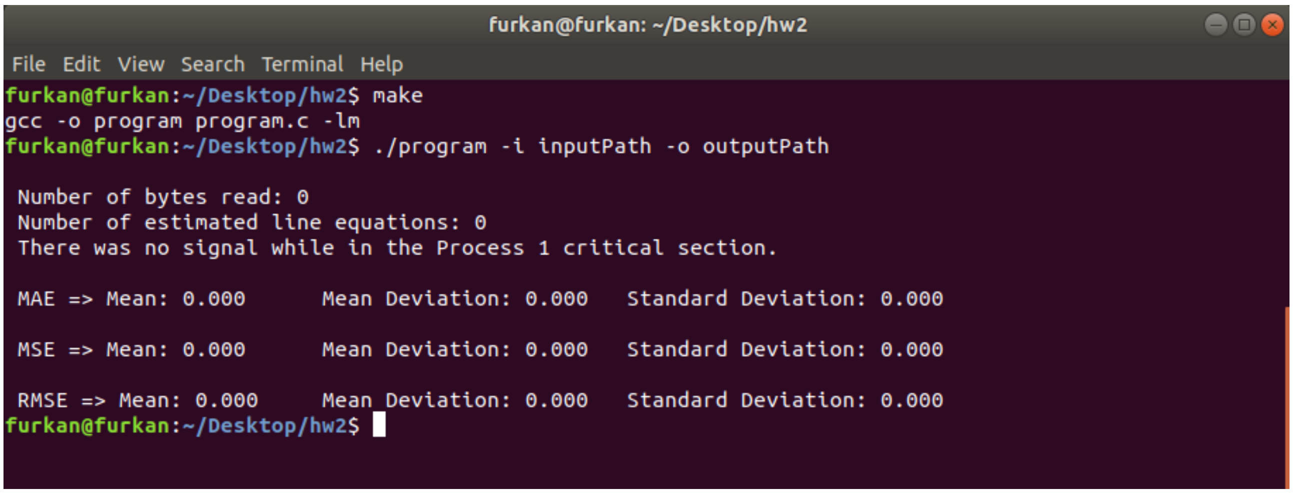
```
(42, 116), (53, 90), (105, 70), (51, 50), (53, 52), (97, 51), (100, 55),
(103, 57), (65, 49), (120, 50), -0.364x+92.739, 14.200, 349.437, 18.693
(33, 43), (54, 122), (61, 52), (38, 37), (120, 35), (70, 71), (55, 54),
(122, 83), (53, 122), (122, 61), -0.067x+72.898, 24.700, 919.188, 30.318
(52, 115), (102, 42), (116, 53), (90, 105), (70, 51), (50, 53), (52,
97), (51, 100), (55, 103), (57, 65), -0.542x+116.036, 20.000, 544.433,
23.333
(49, 120), (50, 33), (43, 54), (122, 61), (52, 38), (37, 120), (35, 70),
(71, 55), (54, 122), (83, 53), -0.354x+93.725, 27.800, 1009.437, 31.772
(122, 122), (61, 52), (115, 102), (42, 116), (53, 90), (105, 70), (51,
50), (53, 52), (97, 51), (100, 55), 0.215x+58.826, 24.100, 712.960,
26.701
(103, 57), (65, 49), (120, 50), (33, 43), (54, 122), (61, 52), (38, 37),
(120, 35), (70, 71), (55, 54), -0.133x+66.568, 15.600, 548.805, 23.427
(122, 83), (53, 122), (122, 61), (52, 115), (102, 42), (116, 53), (90,
105), (70, 51), (50, 53), (52, 97), -0.425x+113.470, 21.900, 642.313,
25.344
(51, 100), (55, 103), (57, 65), (49, 120), (50, 33), (43, 54), (122,
61), (52, 38), (37, 120), (35, 70), -0.269x+91.234, 25.800, 893.962,
29.899
(71, 55), (54, 122), (83, 53), (122, 122), (61, 52), (115, 102), (42,
116), (53, 90), (105, 70), (51, 50), 0.163x+70.824, 26.500, 824.129,
28.708
(53, 52), (97, 51), (100, 55), (103, 57), (65, 49), (120, 50), (33, 43),
(54, 122), (61, 52), (38, 37), -0.034x+59.264, 12.700, 501.410, 22.392
(120, 35), (70, 71), (55, 54), (122, 83), (53, 122), (122, 61), (52,
115), (102, 42), (116, 53), (90, 105), -0.585x+126.888, 21.200, 578.049,
24.043
```

RUN EXAMPLE - 2:

INPUT FILE – A: (0 bytes)

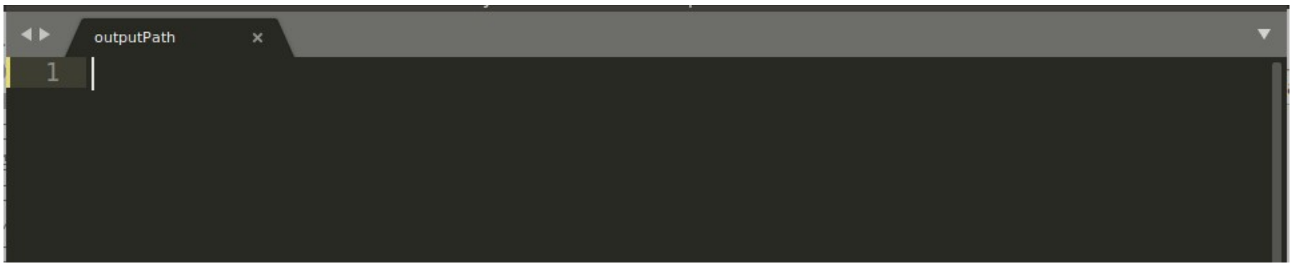


COMPILE & RUN:


furkan@furkan: ~/Desktop/hw2
File Edit View Search Terminal Help

```
furkan@furkan:~/Desktop/hw2$ make
gcc -o program program.c -lm
furkan@furkan:~/Desktop/hw2$ ./program -i inputPath -o outputPath

 Number of bytes read: 0
 Number of estimated line equations: 0
 There was no signal while in the Process 1 critical section.

 MAE => Mean: 0.000       Mean Deviation: 0.000    Standard Deviation: 0.000

 MSE => Mean: 0.000       Mean Deviation: 0.000    Standard Deviation: 0.000

 RMSE => Mean: 0.000      Mean Deviation: 0.000    Standard Deviation: 0.000
furkan@furkan:~/Desktop/hw2$
```

OUTPUT : (0 Equations)



NOTICED:

1- I used the getopt() library method for parsing command line arguments.

2- If the command line arguments are missing/invalid my program will print usage information and exit

3- I locked the files while reading and writing into them

4- I don’t create/use any additional files except temporary file (mkstemp)

5- I tested the signal scenarios in the homework myself.

6- SIGINT is ignored in critical sections, performing its normal function in other sections.

7- If SIGTERM comes, the necessary operations are done and ends in 2 processes.

8- SIGUSR1 and SIGUSR2 are used for communication between processes.

9- I used sigsuspend to make sure P2 waits until there is some input available in the file, if P1 is not yet done with it.

10- While performing calculations, all mathematical operations were realized with accuracy down to 3 decimal points as desired in the homework.

11- Compilation is warning-free

12- I closed all files and free all resources explicitly

13- I used system calls for all file I/O purposes, I didn’t use standard C library functions for file I/O.

14- I did’t use busy waiting or any form of sleep. I just used suspend function.

15- In case of an error, I exit program by printing to stderr a nicely formatted informative errno based message.

16- If you want to test the signals, you can test it by putting sleep () where necessary.