Name: Raorane Sakshi
CDAC AUGUST 2024
CDAC MUMBAI
Module 1: Concepts of Operating System
Assignment 2

Part A

What will the following commands do?

1. echo "Hello, World!"
   -This command prints "Hello, World!" to the terminal.

2. name="Productive"
   -This assigns the string "Productive" to the variable 'name'.

3. touch file.txt
   -This command creates an empty file named 'file.txt'.

4. ls -a
   -This command lists all files and directories in the current directory including the files name starts with '.'.

5. rm file.txt
   -This command removes or deletes the file named file.txt.

6. cp file1.txt file2.txt
   -This command copies the contents of file1.txt to file2.txt. If the file2.txt is not there it'll create a new file named file2.txt'

7. mv file.txt /path/to/directory/
   -This command moves file.txt to the specified directory /path/to/directory/ if there is already a file present named file1.txt it'll override the present file.

8. chmod 755 script.sh
   -This command changes the permissions of script.sh to 755, which means the owner can read, write, and execute the file, while the group and others can only read and execute it.
   i. Owner: Read, write, and execute (7)
   ii. Group: Read and execute (5)
   iii. Others: Read and execute (5)

9. grep "pattern" file.txt

-This command searches for the string "pattern" within file.txt and prints lines that match the pattern.

10. kill PID
    -This command sends a signal to the process with the specified Process ID (PID) to terminate or end that PID.

11. mkdir mydir && cd mydir && touch file.txt && echo "Hello, World!" > file.txt && cat file.txt
    -This sequence of commands:
    i. mkdir mydir - Creates a directory called mydir.
    ii. cd mydir - Changes the current directory to mydir.
    iii. touch file.txt - Creates an empty file named file.txt.
    iv. echo "Hello, World!" > file.txt - Writes "Hello, World!" to file.txt.
    v. cat file.txt - Displays the contents of file.txt to the terminal.

12. ls -l | grep ".txt"
    -This command lists all files in the current directory in long format (detailed information like permissions, owner, size,date,time etc.) and then filters the output to display only files with .txt in their names.

13. cat file1.txt file2.txt | sort | uniq
    -This command concatenates the contents of file1.txt and file2.txt, sorts the combined lines, and then filters out any duplicate lines, displaying only unique lines.

14. ls -l | grep "^d"
    -This command lists all files and directories in long format, then filters the output to show only directories that start with "d".

15. grep -r "pattern" /path/to/directory/
    -This command searches recursively for the string "pattern" in all files within /path/to/directory/ and its subdirectories.

16. cat file1.txt file2.txt | sort | uniq –d
    -This command concatenates the contents of file1.txt and file2.txt, sorts the combined lines, and then displays only duplicate lines i.e. lines that appear more than once.

17. chmod 644 file.txt
    -This command changes the permissions of file.txt to 644, meaning the owner can read and write the file, while the group and others can only read it.
    i. Owner: Read and write (6)
    ii. Group: Read only (4)
    iii. Others: Read only (4)

18. cp -r source_directory destination_directory
    -This command copies the source_directory and all its contents including subdirectories to the destination_directory. The -r option means recursive, allowing it to copy directories.

19. find /path/to/search -name "*.txt"
    -This command searches for files ending with .txt within the /path/to/search directory and its subdirectories.

20. chmod u+x file.txt
    -This command adds execute permissions to the owner of the file.txt.

21. echo $PATH
    -This command displays the current value of the PATH environment variable, which contains a list of directories the shell searches for executable files.

Part B

Identify True or False:

1. ls is used to list files and directories in a directory.
    -True

2. mv is used to move files and directories.
    -True

3. cd is used to copy files and directories.
    -False. cd is used to change the directories.

4. pwd stands for "print working directory" and displays the current directory.
    - True

5. grep is used to search for patterns in files.
    -True

6. chmod 755 file.txt gives read, write, and execute permissions to the owner, and read and execute permissions to group and others.
    -True

7. mkdir -p directory1/directory2 creates nested directories, creating directory2 inside directory1 if directory1 does not exist.
    -True

8. rm -rf file.txt deletes a file forcefully without confirmation.
    -True

Identify the Incorrect Commands:

1. chmodx is used to change file permissions.
    - chmod

2. cpy is used to copy files and directories.
    - cp

3. mkfile is used to create a new file.
    - touch OR nano

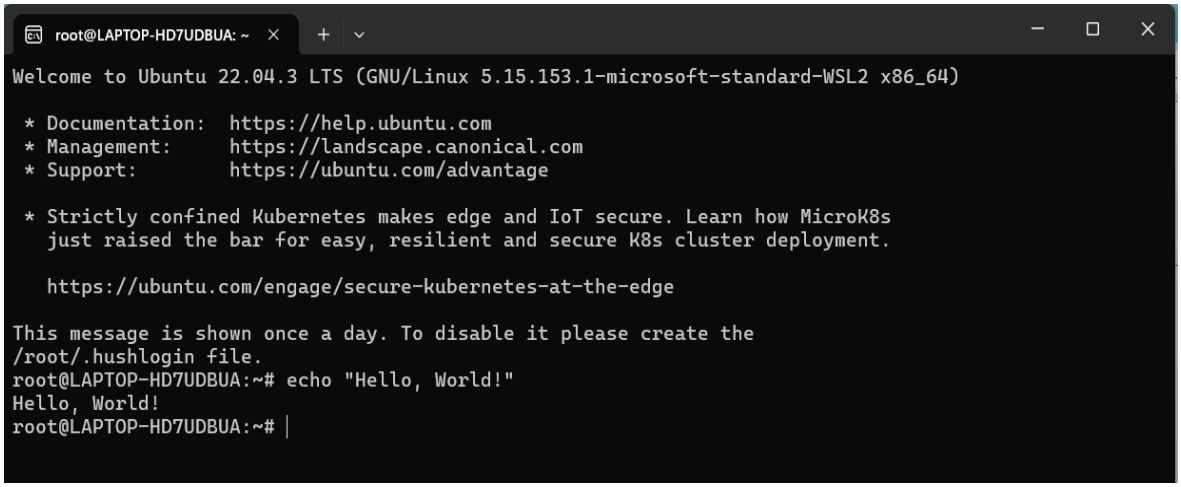
4. catx is used to concatenate files.
    - cat

5. rn is used to rename files.
    - mv

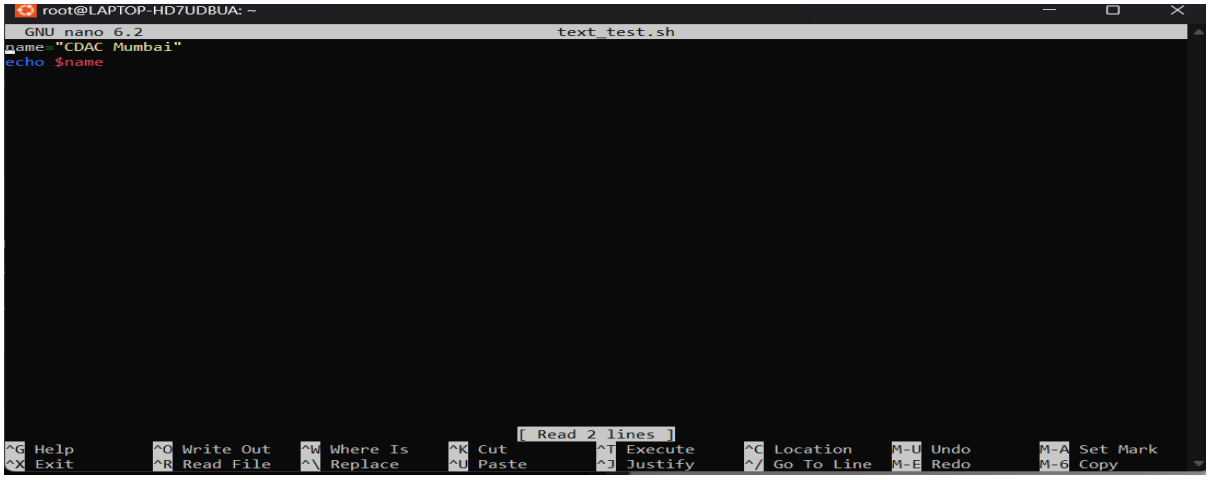
Part C

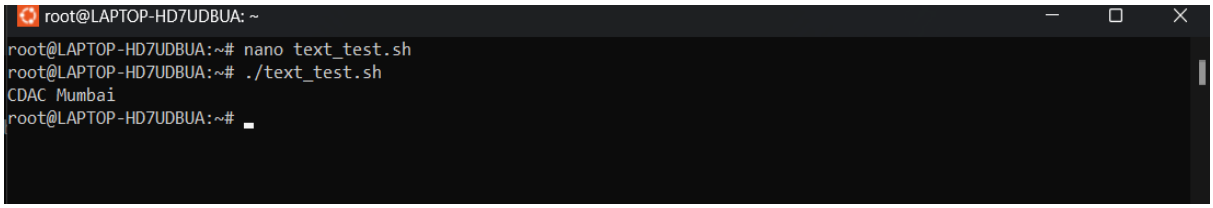
Question 1: Write a shell script that prints "Hello, World!" to the terminal.

- Echo “Hello, World!”

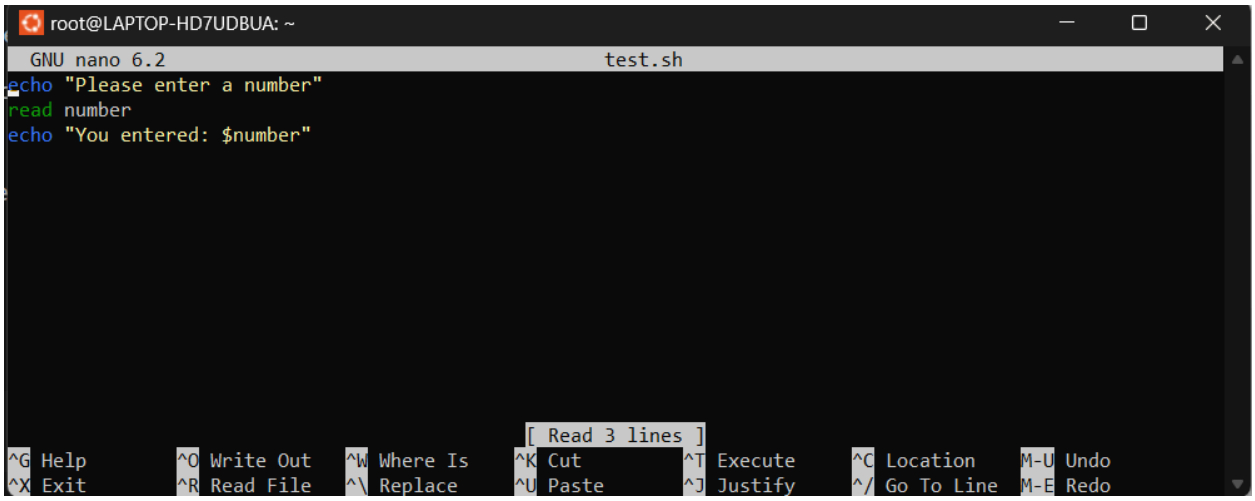


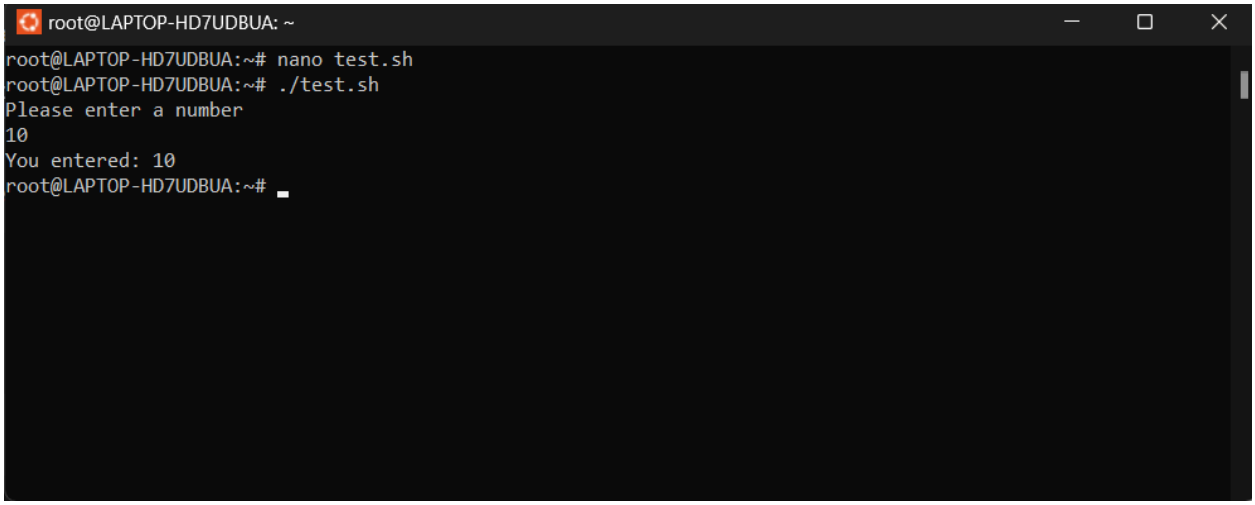
Question 2: Declare a variable named "name" and assign the value "CDAC Mumbai" to it. Print the value of the variable.
Nano text_test.sh
name=”CDAC Mumbai”
echo $name
Chmod +x text_test.sh
./text_test.sh



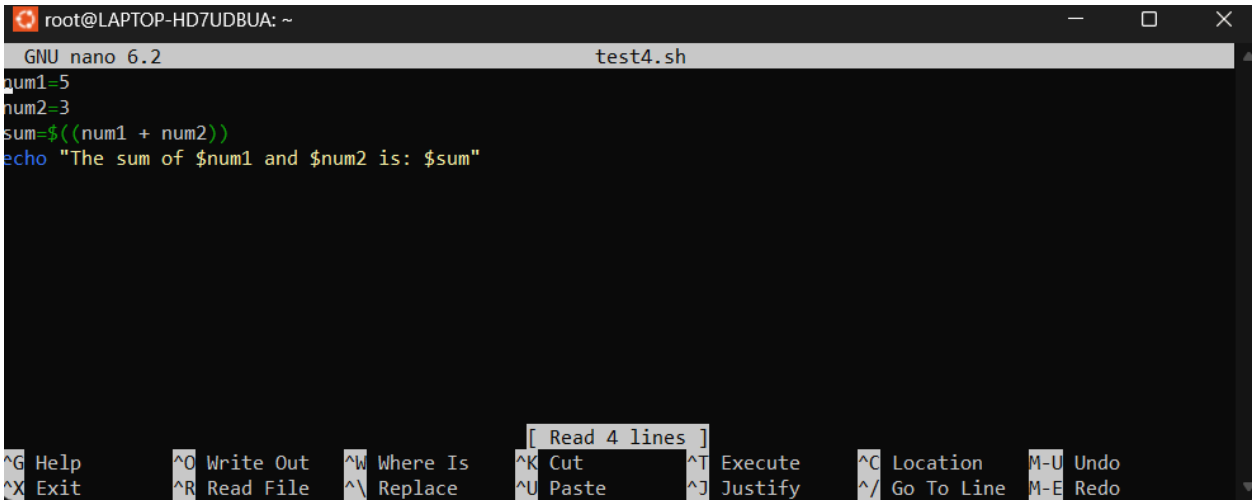


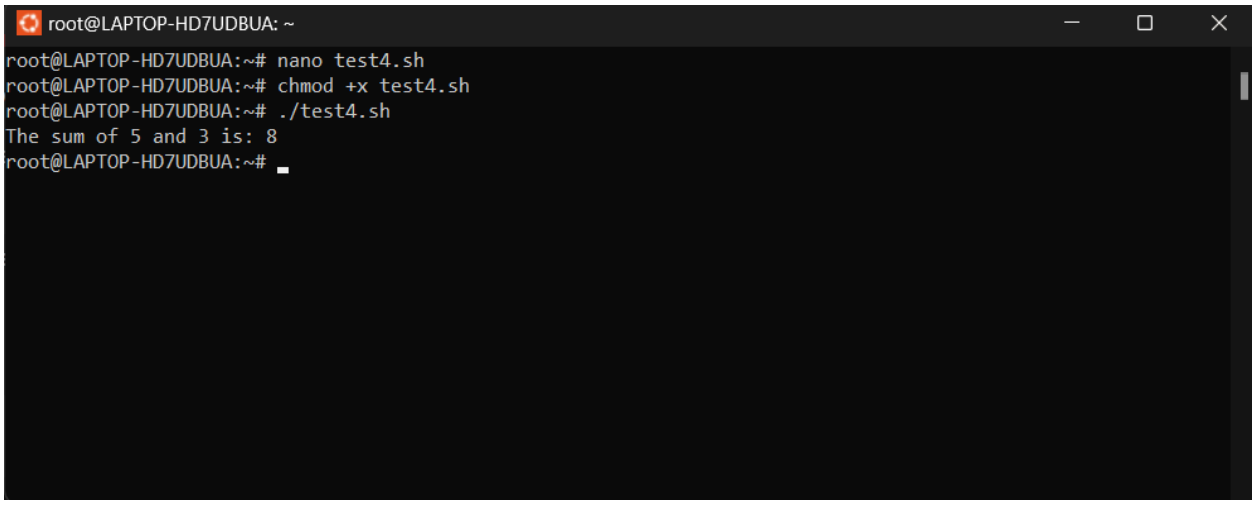
Question 3: Write a shell script that takes a number as input from the user and prints it.
Nano test.sh
echo “Please enter a number”
read number
Echo “You entered $number”
./test.sh





Question 4: Write a shell script that performs addition of two numbers (e.g., 5 and 3) and prints the result.
Nano test4.sh
num1=5
num2=3
sum=$((num1 + num2))
Echo “The sum of $num1 and $num2 is: $sum”
Chmod +x test4.sh
./test4.sh





Question 5: Write a shell script that takes a number as input and prints "Even" if it is even, otherwise prints "Odd".

nano test5.sh

echo “PLease enter a number ”

read number

if((nuumber % 2 == 0)); then

echo “Even”

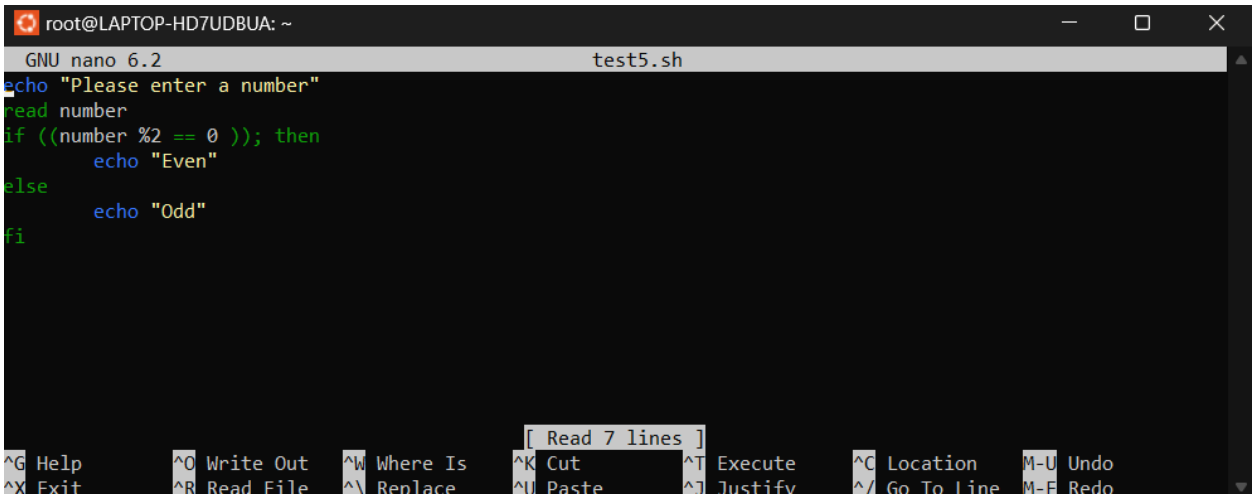
else

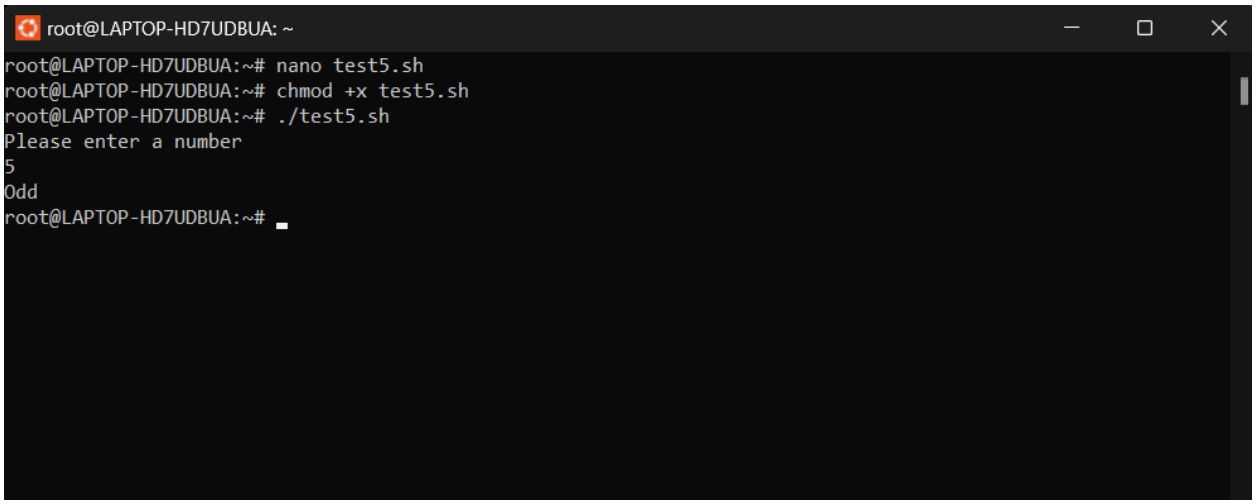
Echo “Odd”

fi

chmod +x test5.sh

./test5.sh



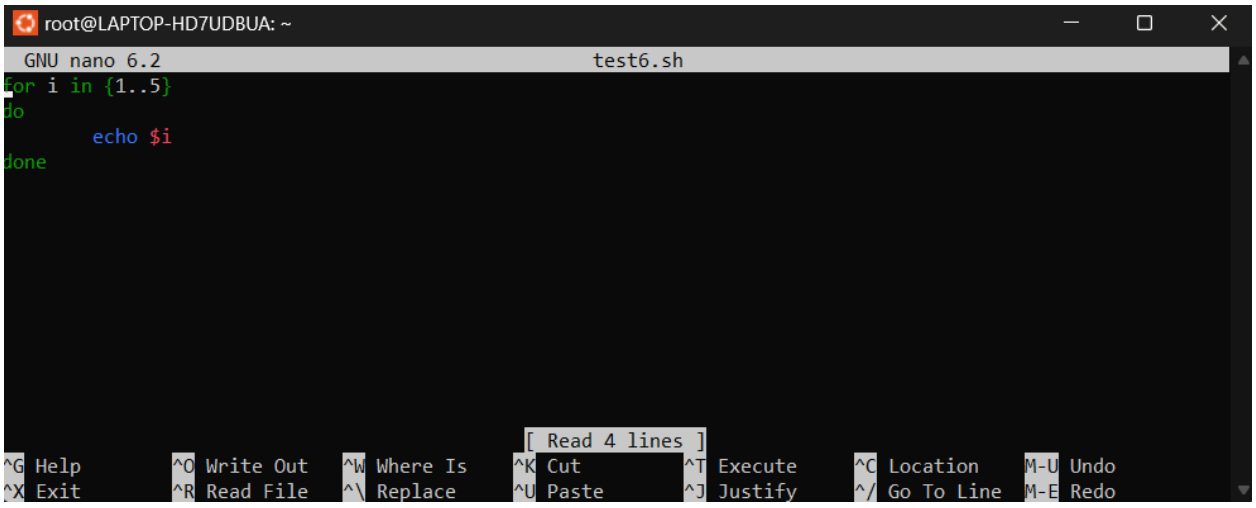


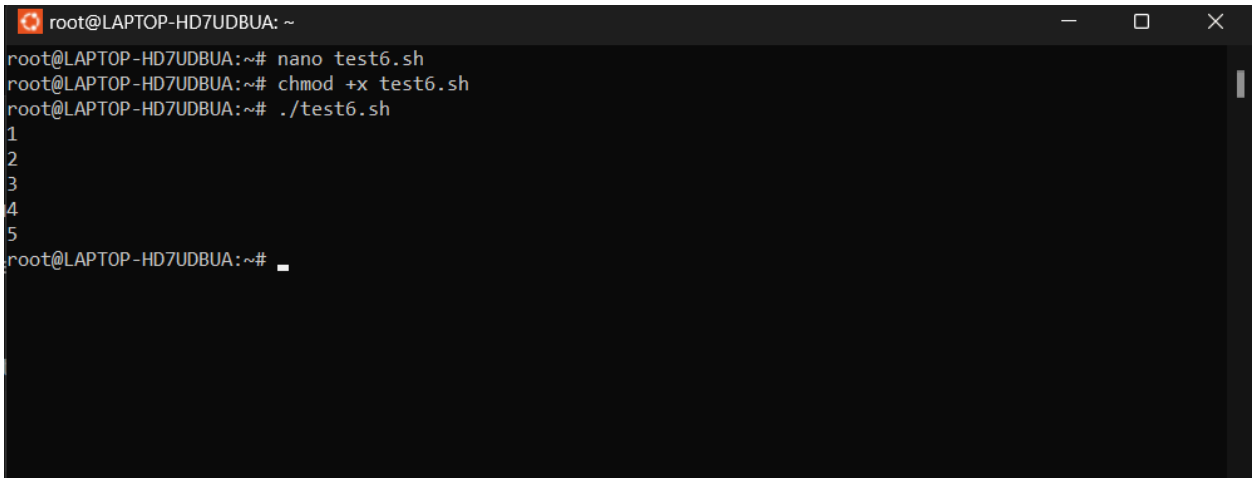
Question 6: Write a shell script that uses a for loop to print numbers from 1 to 5.

nano test6.sh

```
for i in {1..5}
do
        echo $i
done
```

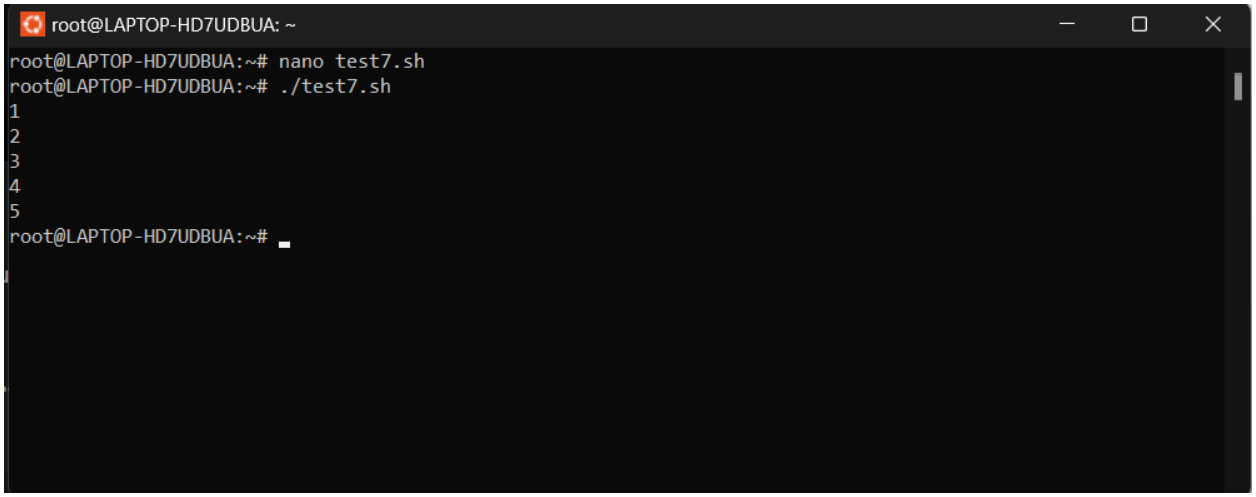
./test6.sh

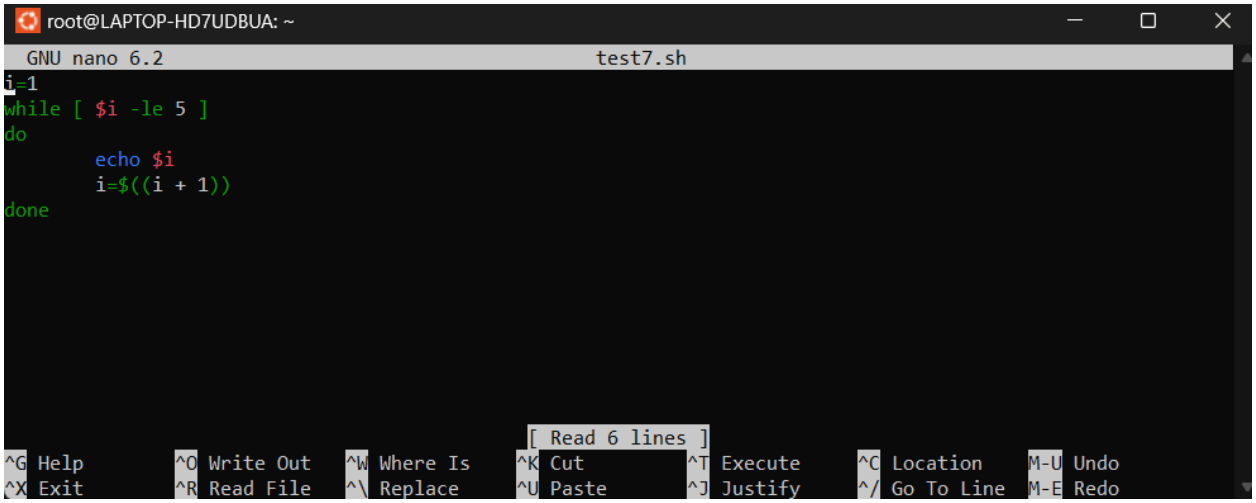




Question 7: Write a shell script that uses a while loop to print numbers from 1 to 5.

```
nano test7.sh
i=1
while[ $i -le 5 ]
do
        echo $i
        i=$((i + 1))
done
./test7.sh
```





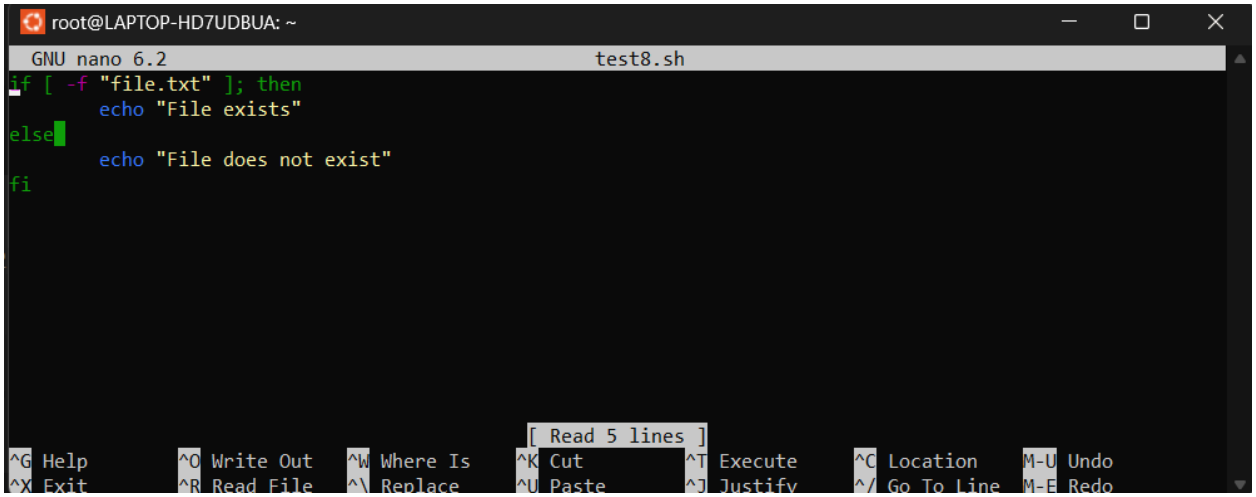
Question 8: Write a shell script that checks if a file named "file.txt" exists in the current directory. If it does, print "File exists", otherwise, print "File does not exist".

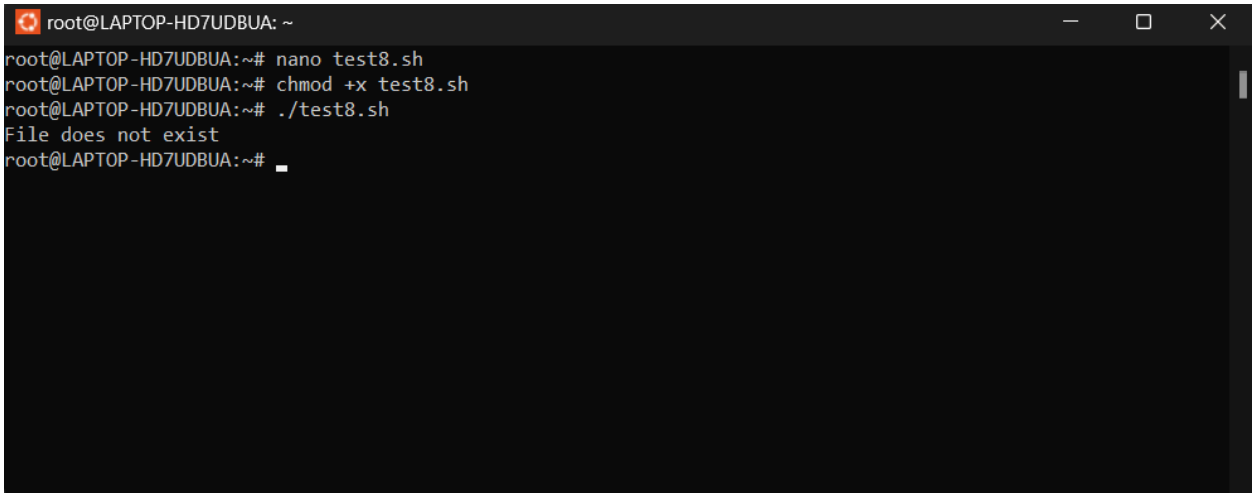
nano test8.sh

```
If[ -f “file.txt” ]; then
        echo “File exists”
else
        echo “File does not exist”
fi
```

chmod +x test.sh

./test8.sh

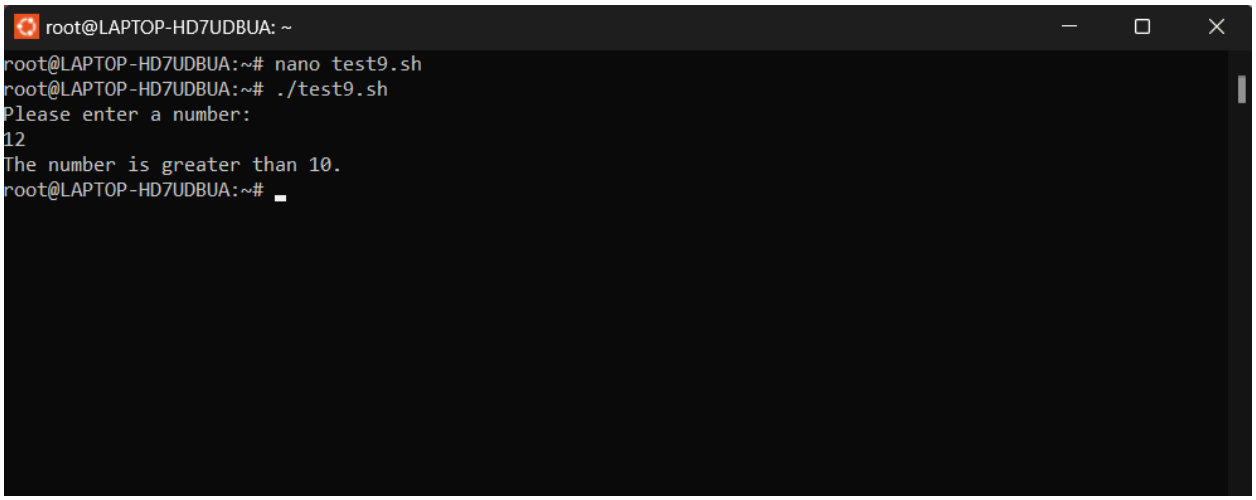


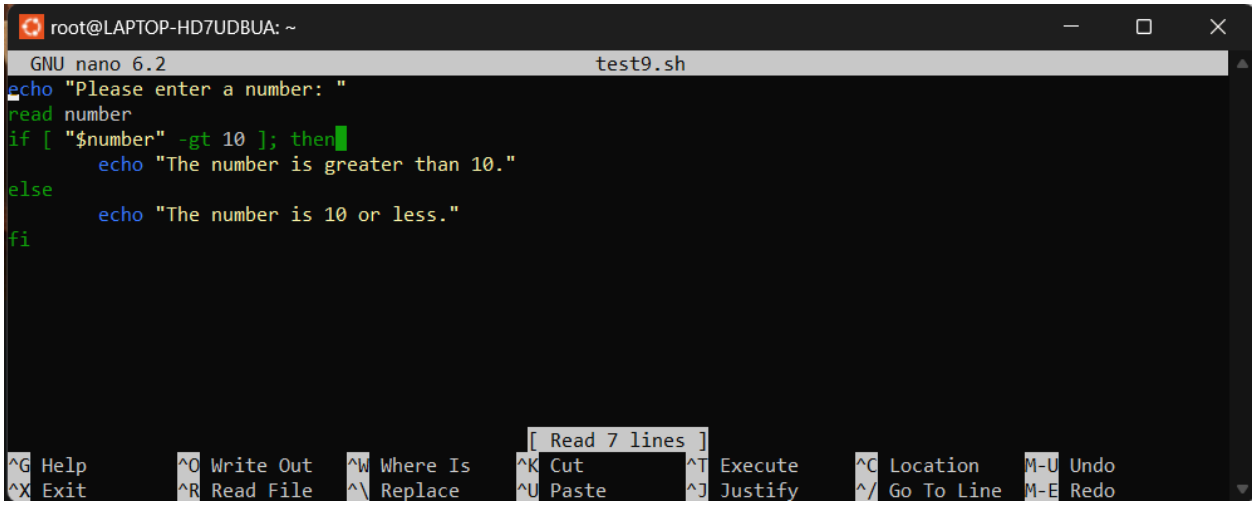


Question 9: Write a shell script that uses the if statement to check if a number is greater than 10 and prints a message accordingly.

nano test9.sh

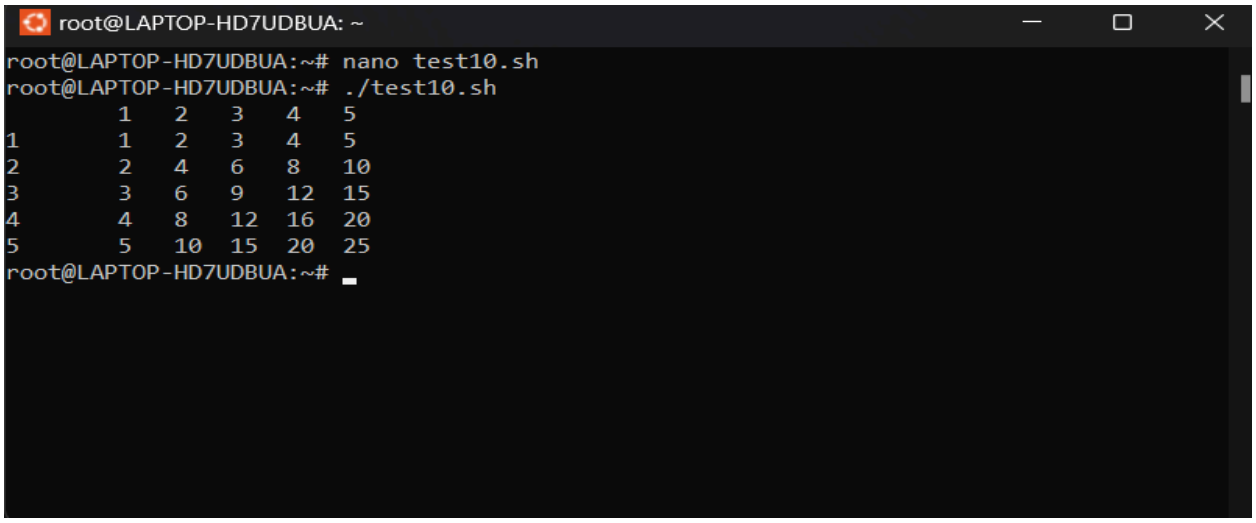
```
echo “Please enter a number”
read number
If [ “$number” -gt 10]; then
        echo “The number is greater than 10.”
else
        echo “The number is 10 or less.”
fi
```

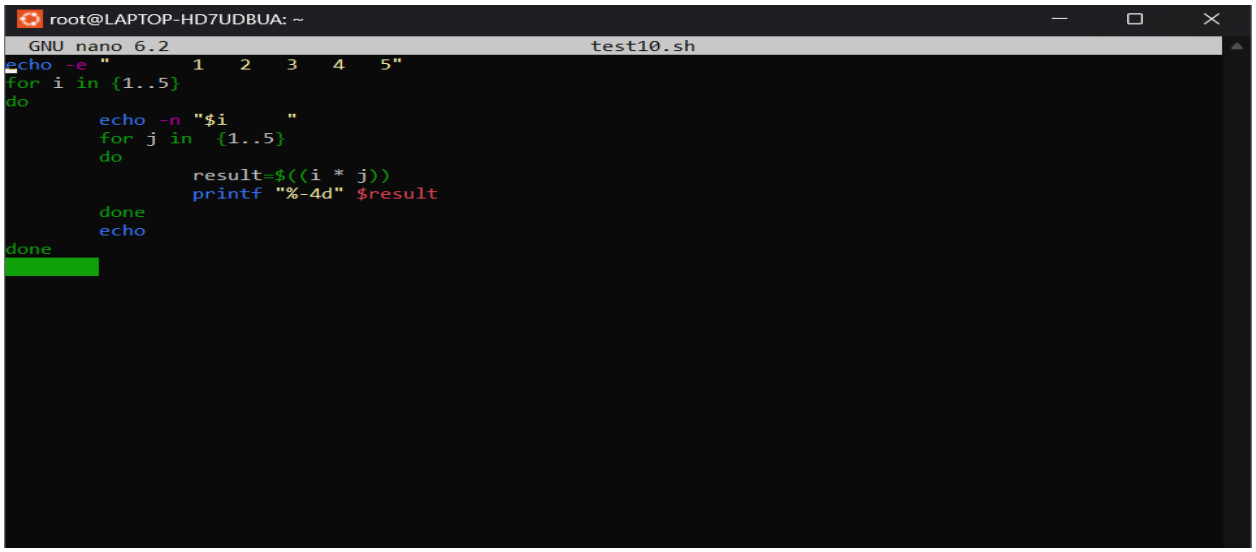




Question 10: Write a shell script that uses nested for loops to print a multiplication table for numbers from 1 to 5. The output should be formatted nicely, with each row representing a number and each column representing the multiplication result for that number.

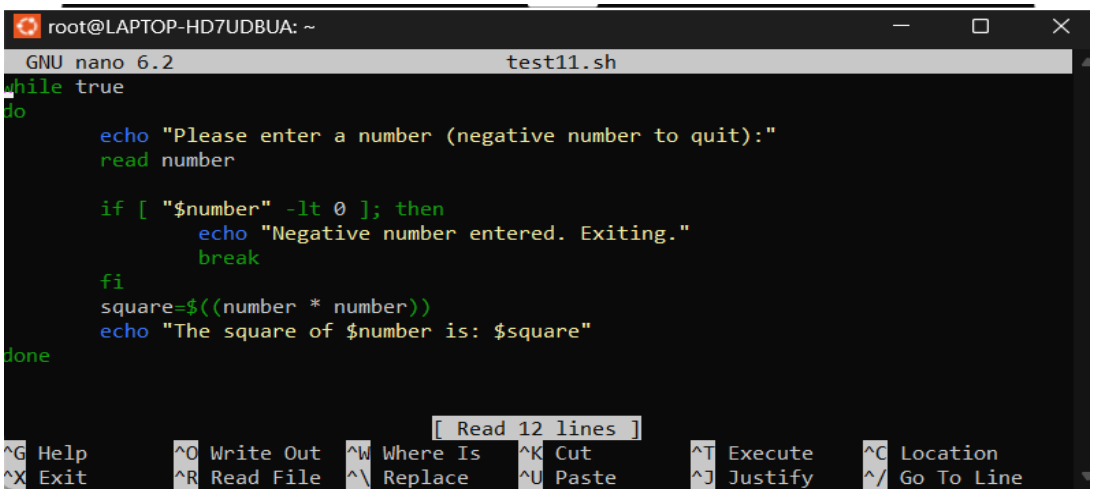
```
echo -e " 1\t2\t3\t4\t5"
for i in {1..5}
do
        echo -n "$i "
        for j in {1..5}
        do
                result=$((i * j))
                 printf "%-4d" $result
                done
                echo
 done
```

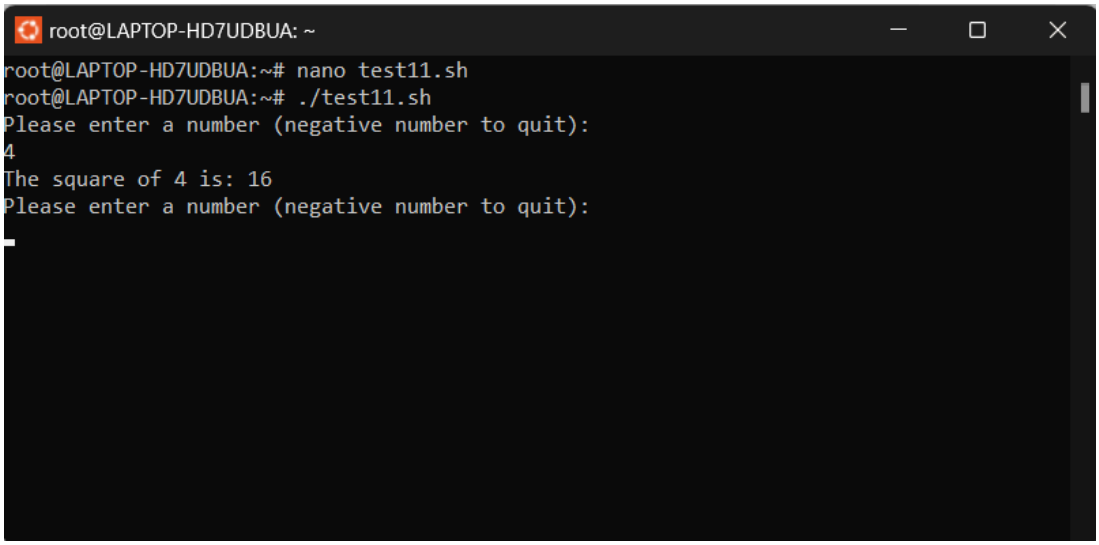




Question 11: Write a shell script that uses a while loop to read numbers from the user until the user enters a negative number. For each positive number entered, print its square. Use the break statement to exit the loop when a negative number is entered.

```
while true
 do
        echo "Please enter a number (negative number to quit):"
            if [ "$number" -lt 0 ]; then
            echo "Negative number entered. Exiting."
            break
        fi
        square=$((number * number))
        echo "The square of $number is: $square"
done
```





Part E

1. Consider the following processes with arrival times and burst times:

Process	Arrival Time	Burst Time
P1	0	5
P2	1	3
P3	2	6

Calculate the average waiting time using First-Come, First-Served (FCFS) scheduling.

FCFS

Process	Arrival Time	Burst Time	Waiting Time	TAT	CT
P1	0	5	0	5-0=5	5
P2	1	3	5-1=4	8-1=7	8
P3	2	6	8-2=6	14-2=12	14

GANTT CHART

	P1	P2	P3	
0		5	8	14

Average Waiting Time = 0 + 4 + 6 / 3

Average Waiting Time = 3.3

2. Consider the following processes with arrival times and burst times:

Process	Arrival Time	Burst Time
P1	0	3
P2	1	5
P3	2	1
P4	3	4

Calculate the average turnaround time using Shortest Job First (SJF) scheduling

SJF

Process	Arrival Time	Burst Time	CT	WT	TAT
P1	0	3	3	0	3
P2	1	5	13	7	12
P3	2	1	4	1	2
P4	3	4	8	1	5

GANTT CHART

P1	P3	P4	P2	
0	3	4	8	13

Average TAT = 3 + 12 + 2 + 5 / 4

Average TAT = 5.5

3. Consider the following processes with arrival times, burst times, and priorities (lower number indicates higher priority):

Process	Arrival Time	Burst Time	Priority
P1	0	6	3
P2	1	4	1
P3	2	7	4
P4	3	2	2

Calculate the average waiting time using Priority Scheduling.

PRIORITY

Process	Arrival Time	Burst Time	Priority	CT	WT	TAT
P1	0	6	3	6	0	6
P2	1	4	1	10	5	9
P3	2	7	4	19	10	17
P4	3	2	2	12	7	9

GANTT CHART

P1	P2	P4	P3	
0	6	10	12	19

Average Waiting Time = 0 + 5 + 10 + 7 / 4

Average Waiting Time = 5.5

4. Consider the following processes with arrival times and burst times, and the time quantum for Round Robin scheduling is 2 units:

Process	Arrival Time	Burst Time
P1	0	4
P2	1	5
P3	2	2
P4	3	3

Calculate the average turnaround time using Round Robin scheduling.

Round Robin

Process	Arrival Time	Burst Time	CT	WT	TAT	
P1	0	4	10	6-0=6	10	
P2	1	5	16	(2+6+2)-1=10	15	
P3	2	2	6	(4)-2=2	4	
P4	3	3	14	(6+4)-3=8	11	

GANTT CHART

P1	P2	P3	P4	P1	P2	P4	P2	
0	2	4	6	8	10	12	14	16

Average TAT = 6 + 9 + 2 + 7 / 4

Average TAT = 6.5

5. Consider a program that uses the fork() system call to create a child process. Initially, the parent process has a variable x with a value of 5. After forking, both the parent and child processes increment the value of x by 1. What will be the final values of x in the parent and child processes after the fork() call?

When a process calls fork() it creates a new child process that is a duplicate of the parent process. This includes the state of all variables, including the variable x. However, the parent and child processes have separate memory spaces, meaning they do not share the same memory for the variable x. Instead, each process has its own copy of x.

Here's how the execution proceeds:

1. Before Fork:
    - The parent process has a variable x with a value of 5.
2. After Fork:
    - The child process is created with a copy of the parent process's memory, so the child also has a variable x with a value of 5.
    - Now there are two independent processes: the parent and the child.
3. Incrementing x:
    - Both the parent and the child increment their own copy of x by 1.
    - In the parent process, x becomes 6.

    - In the **child process**, `x` also becomes `6`.

Final Values:

- **Parent process:** `x = 6`
- **Child process:** `x = 6`

Thus, after the `fork()` call and the increment operation, the value of `x` will be `6` in both the parent and child processes.