LAB Manual

Name of Student : Aditi Bansal	PRN:23070122013
Semester : IV	**Year AY 24-25**
Subject Title: Operating Systems Lab	
EXPERIMENT No : 2	Assignment No: 3
TITLE : Basic Shell commands	DoP : 28-01-2025

Aim : Demonstrate the use of basic Shell commands

Learning Outcome: *1. To understand the shell command*

2. To Demonstrate the shell command

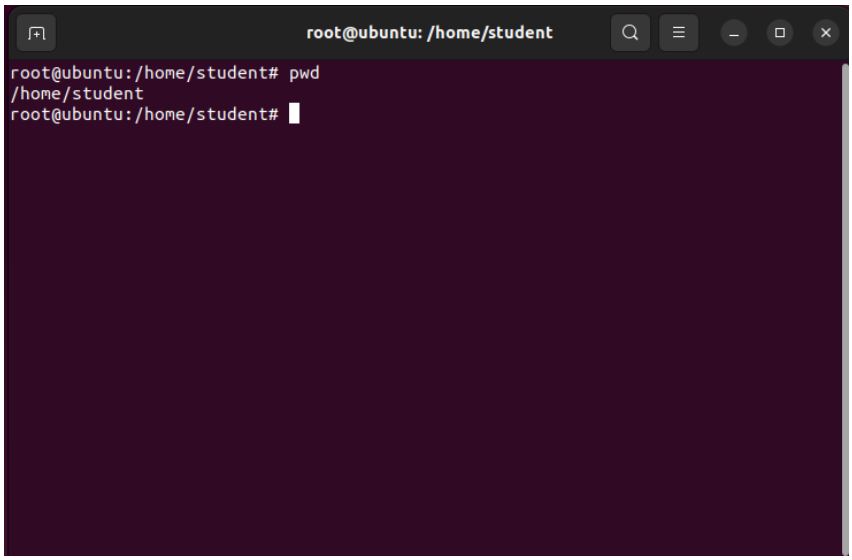
Hardware/Software* : *(It should be handwritten)

Theory: (It should be handwritten)

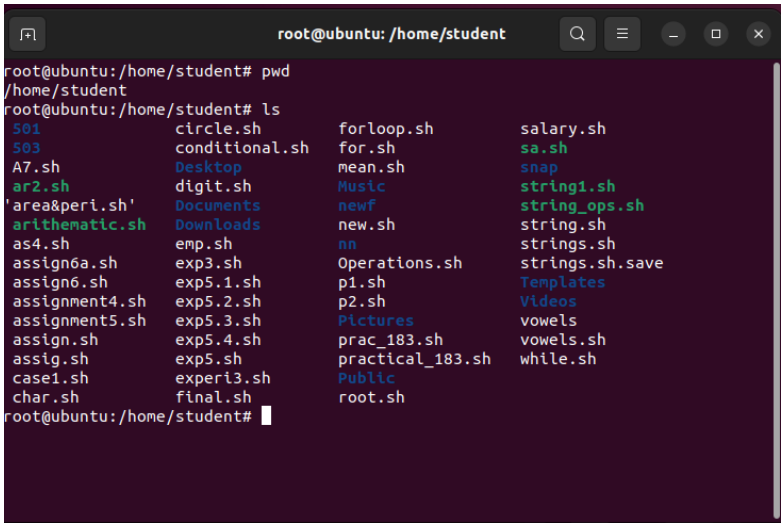
pwd, ls, cd, Touch, mkdir, cat, Rm, Bc, Tty, Who, Man, Date, History, ps, time, Head, Tail, cp, sort, wc, chmod

Output: snapshots of commands demonstration

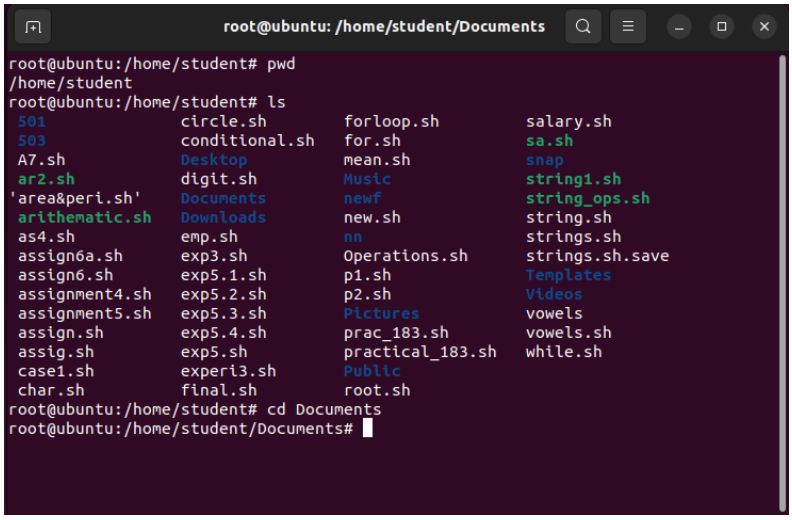
1) Pwd: print working directory



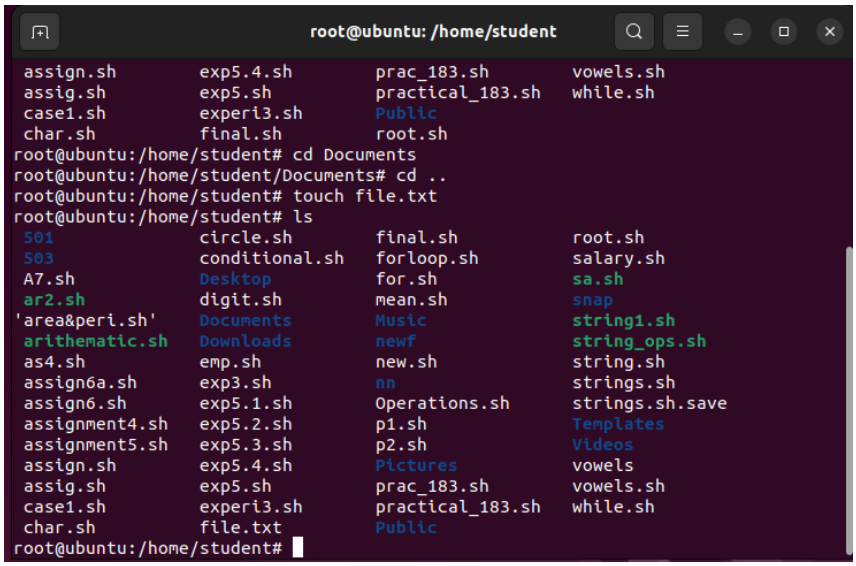
2) Ls: list the content of the directory



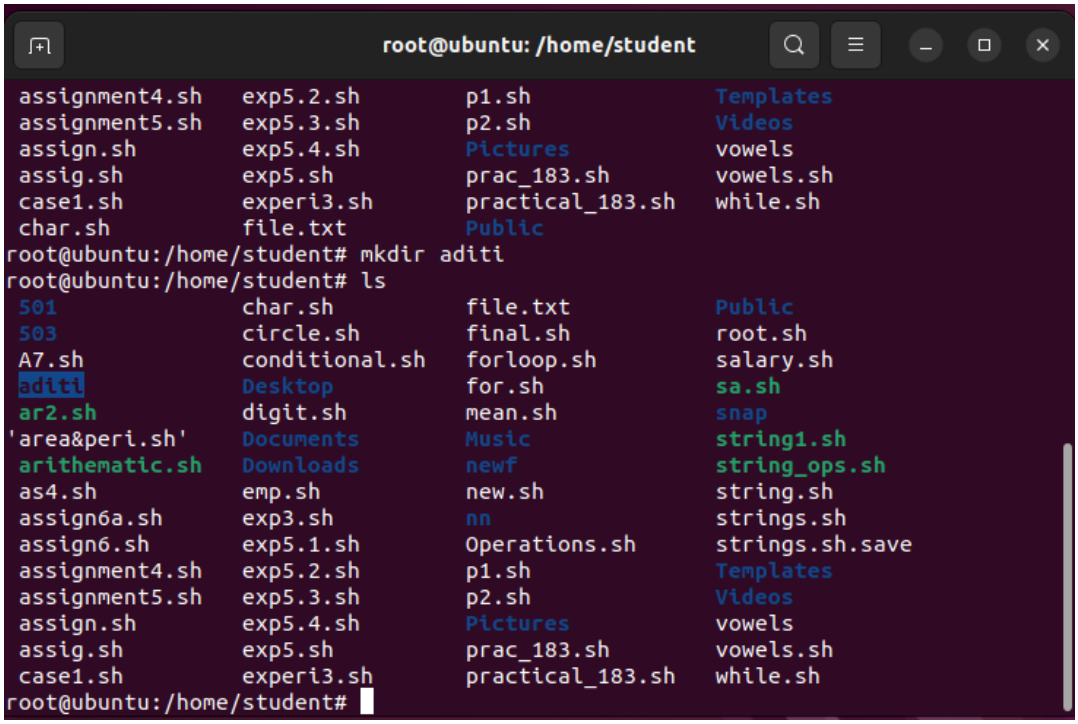
3) Cd: change the current working directory



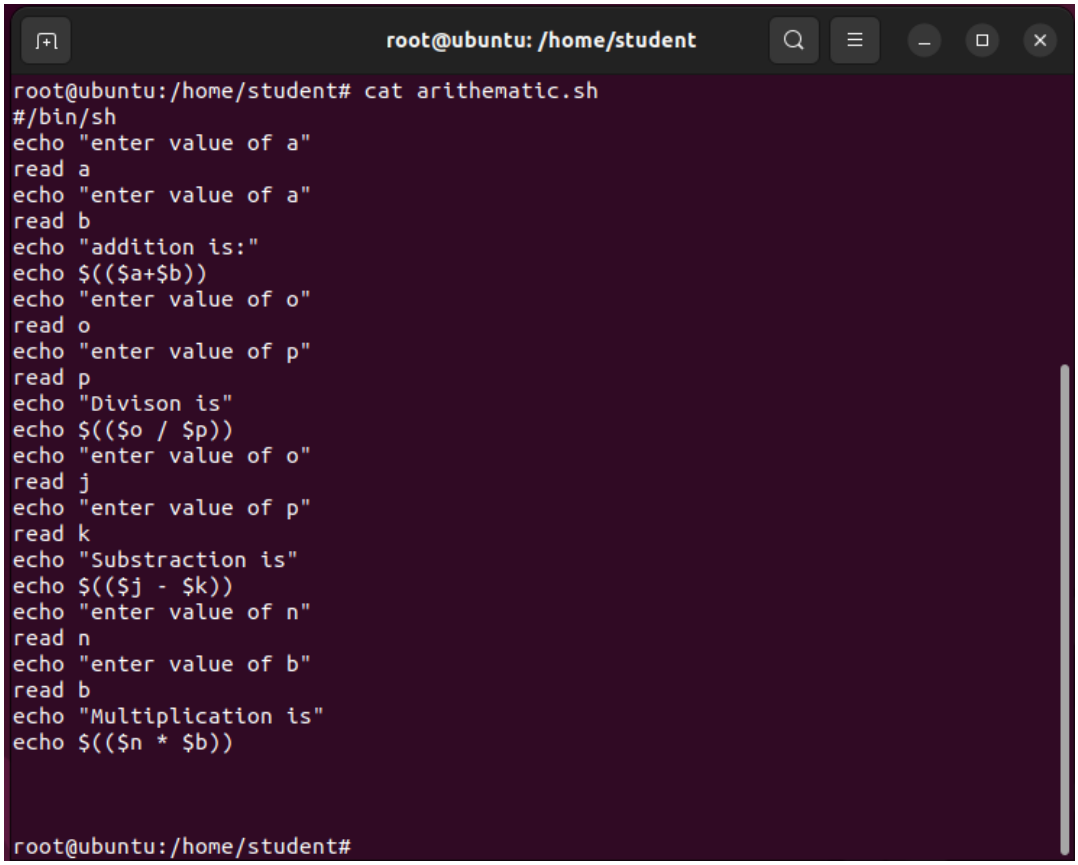
4) Touch: creates empty files and updates timestamps of existing files.



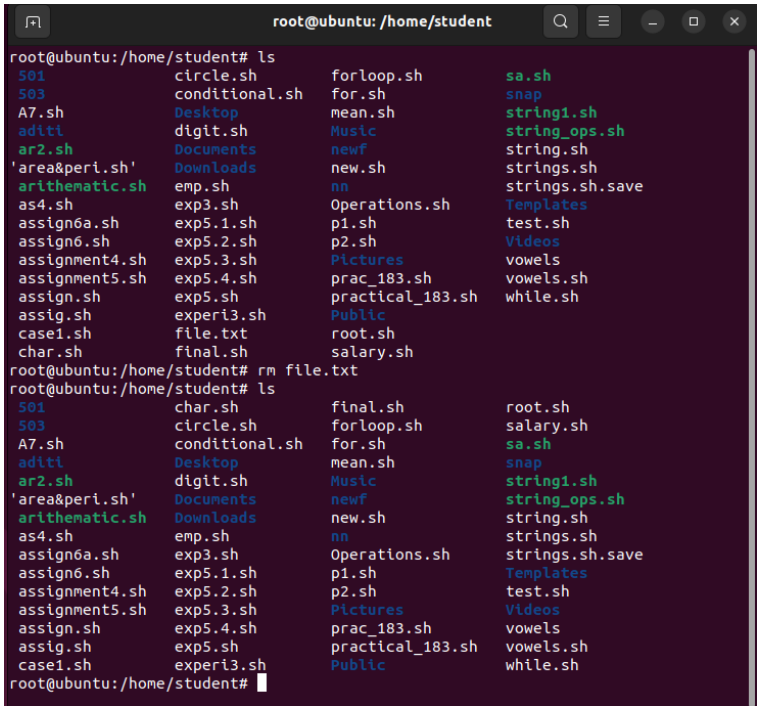
5) Mkdir: create new directory



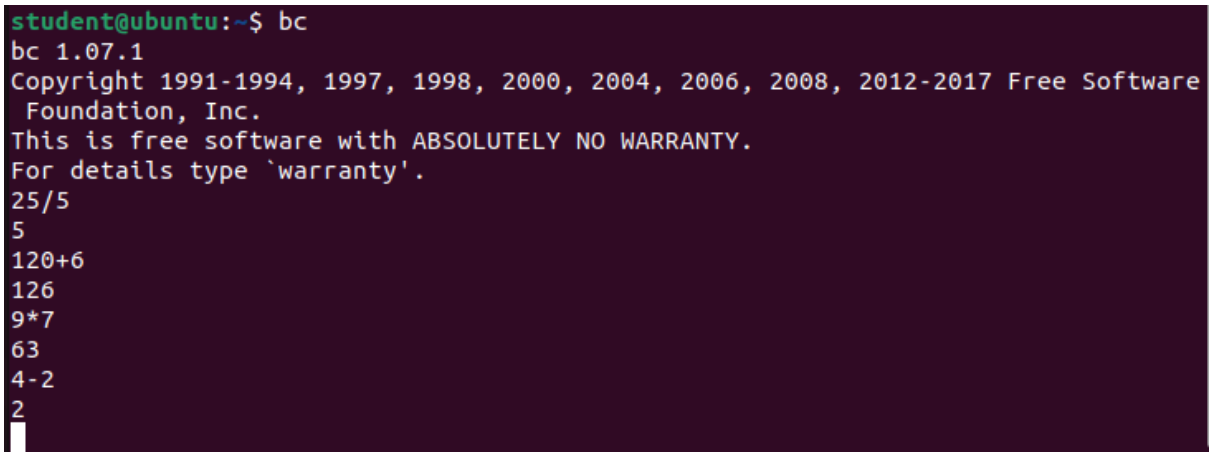
6) Cat: concatenate, basically reads, display and concatenates text files



7) Rm: remove or delete files and directories

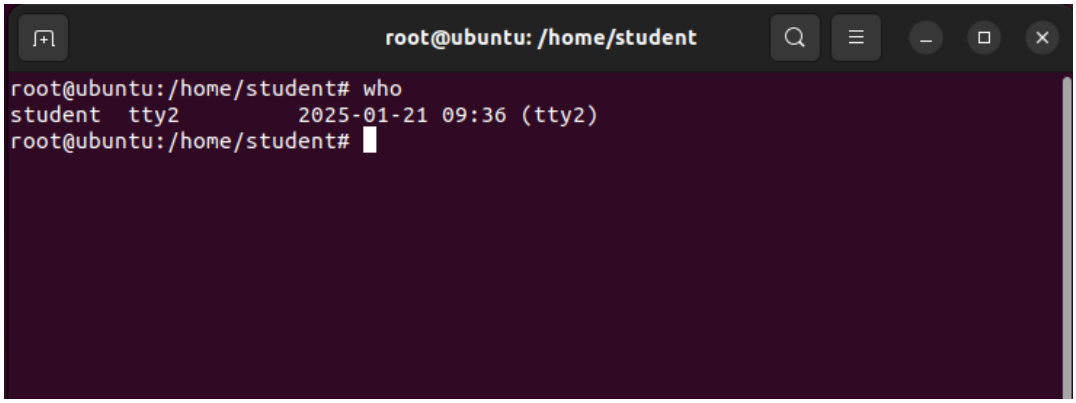


8) Bc: works as a basic calculator

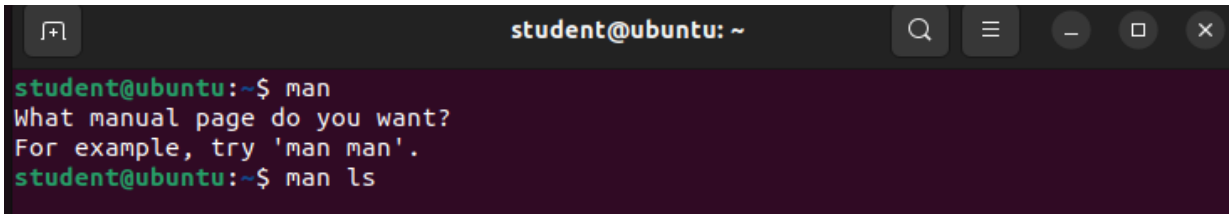


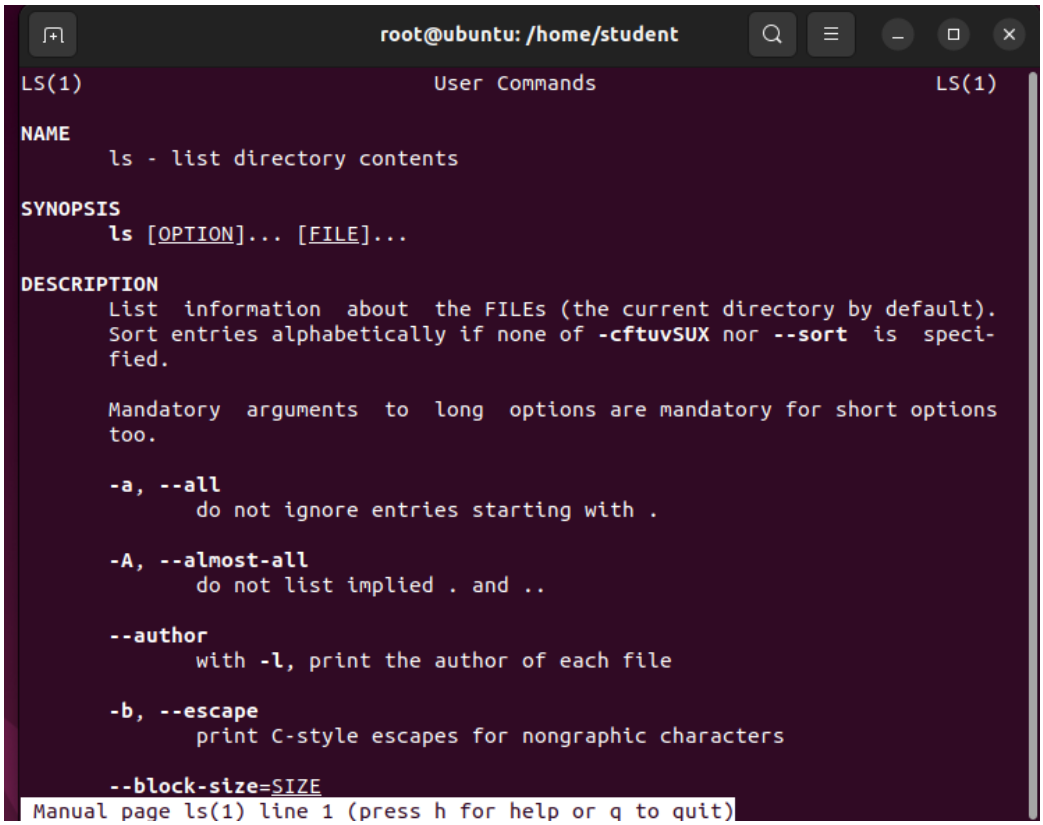
9) Tty: teletypewriter, allows users to interact with the system by sending and receiving text based commands, print the file name of the terminal connected to the standard input

10) Who: display information about users who are currently logged into the system

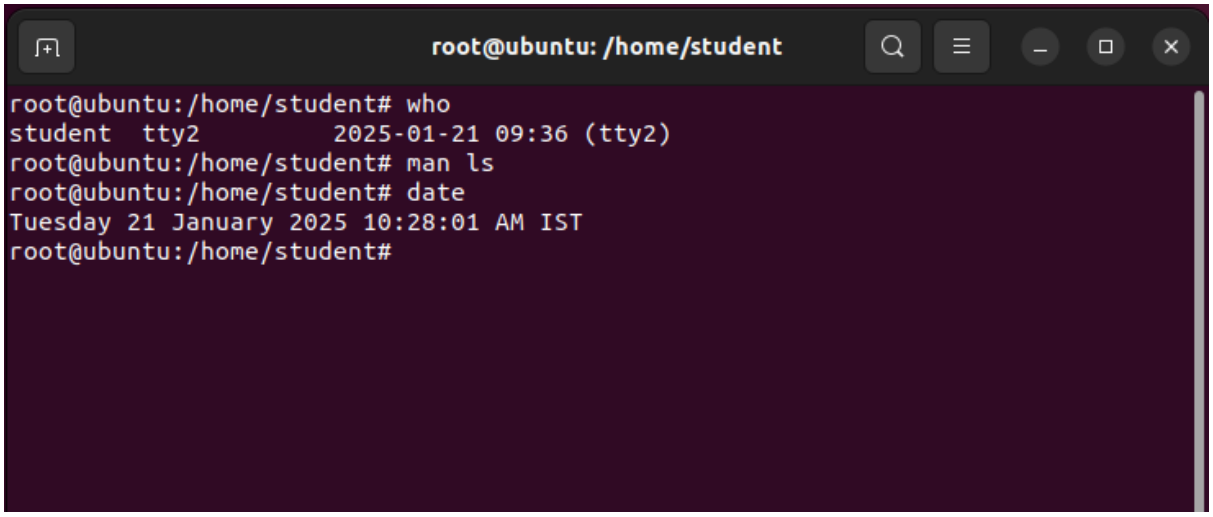


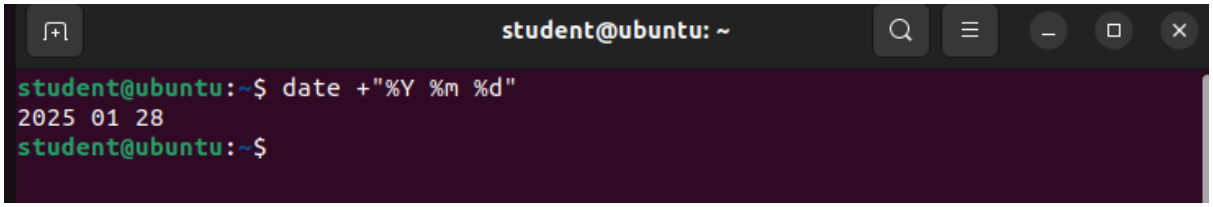
11) Man : displays manual pages which are documentation for commands

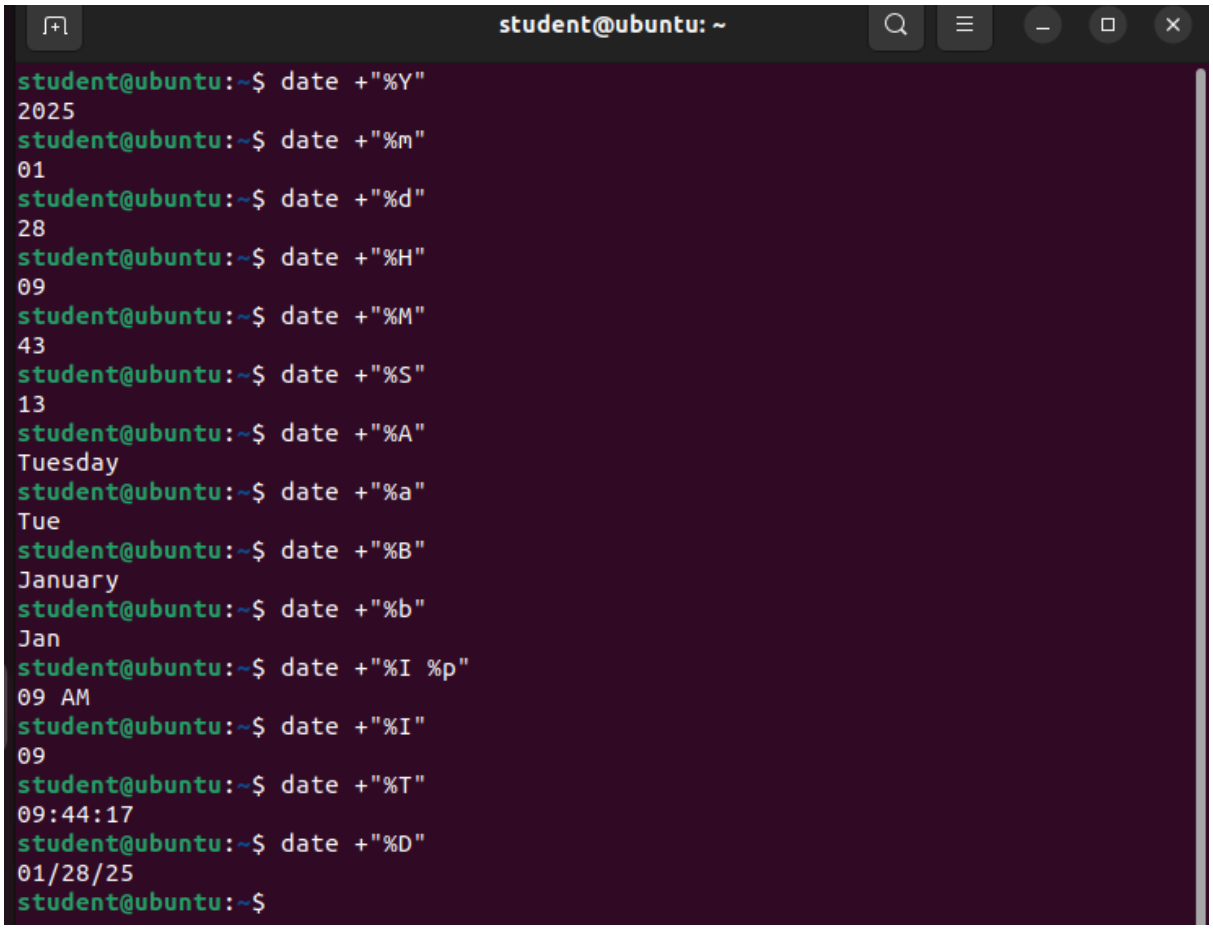




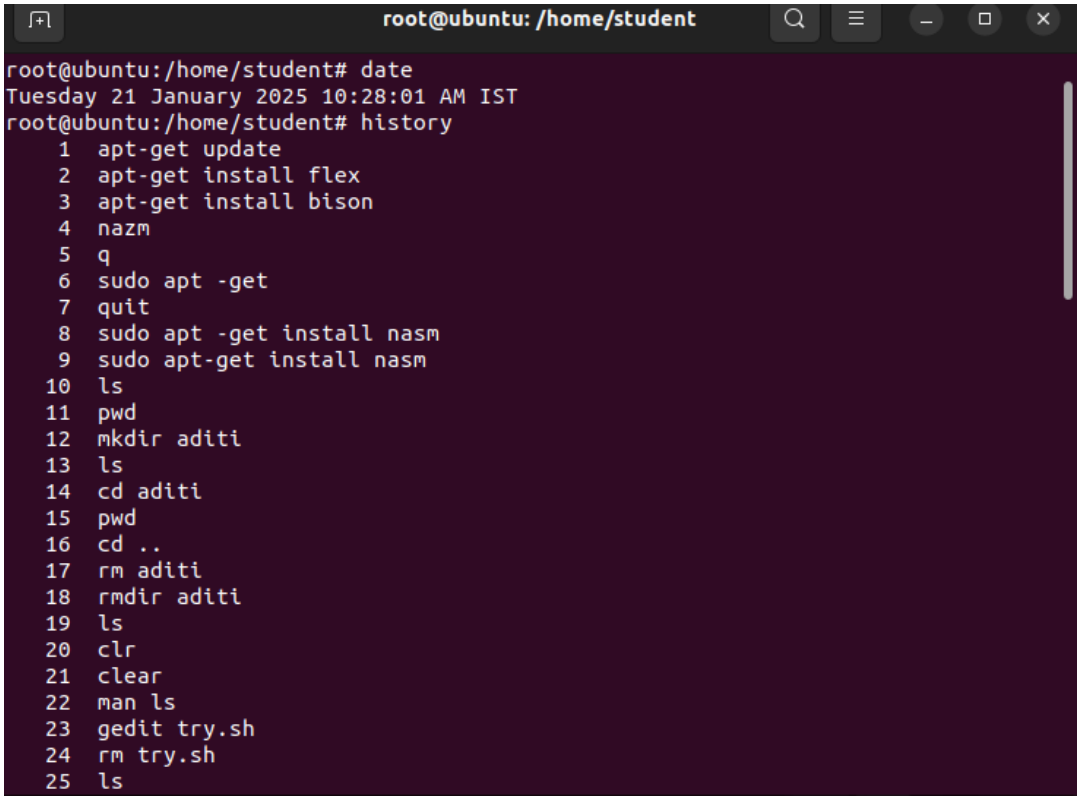
12) Date: displays and sets the system's date and time





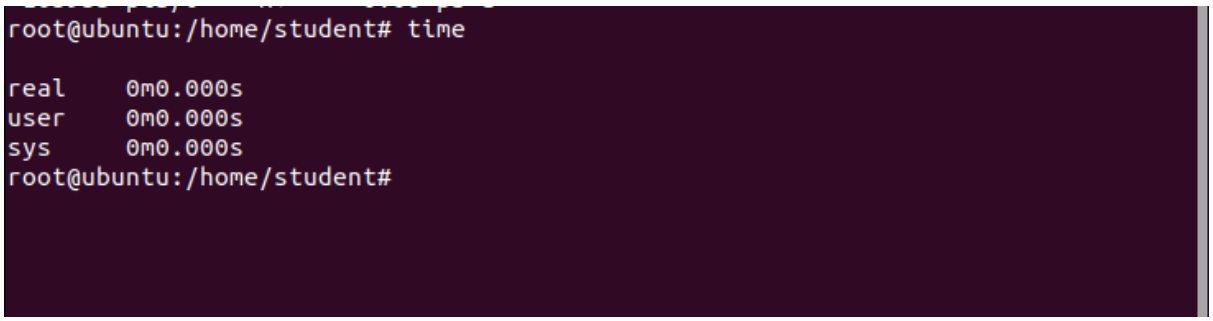


13) History: allows a list of previously executed commands

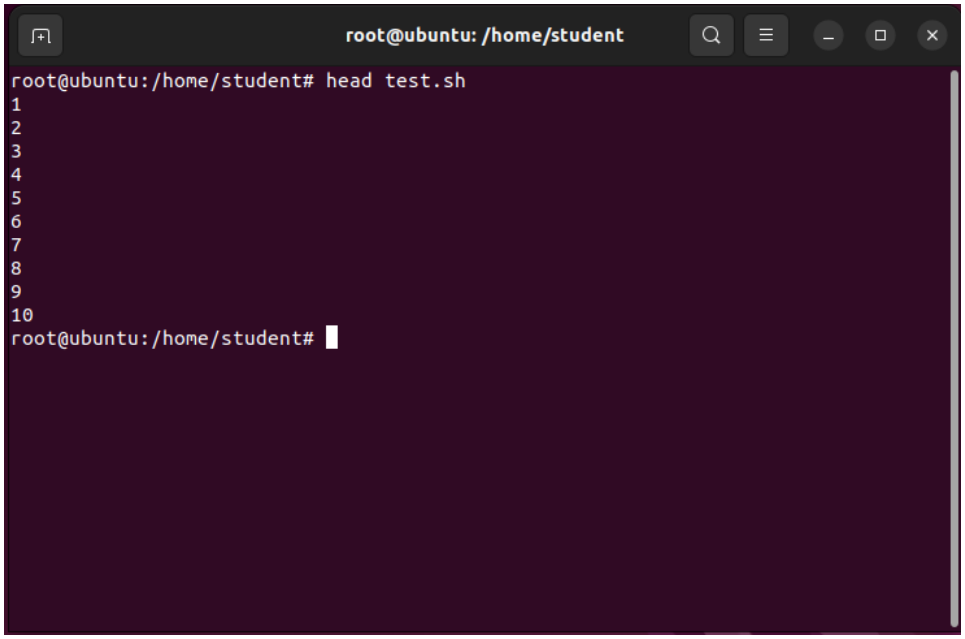


14) Ps: Used to display a snapshot of currently running processes and their associated information

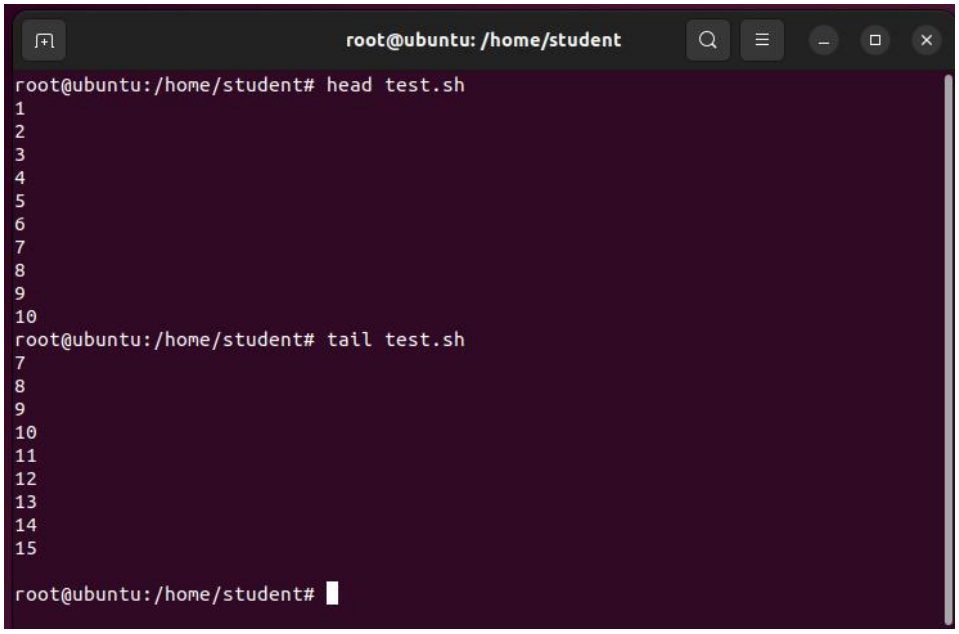
15) Time: used to measure the execution time



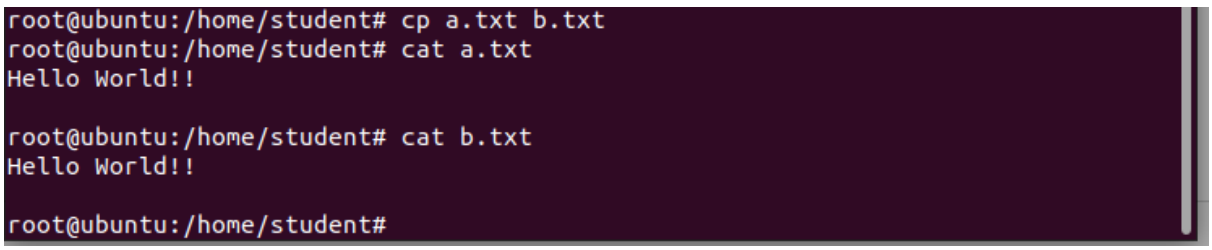
16) Head: output the first part of the file



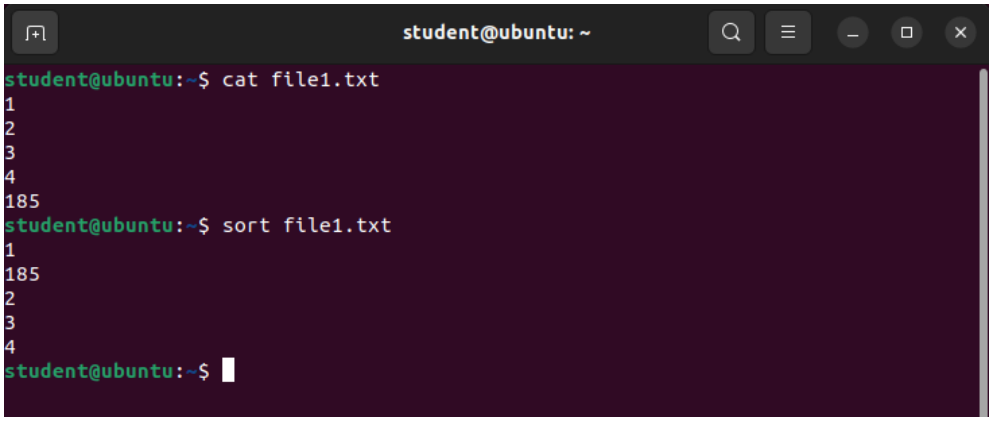
17) Tail: output the last part of the file



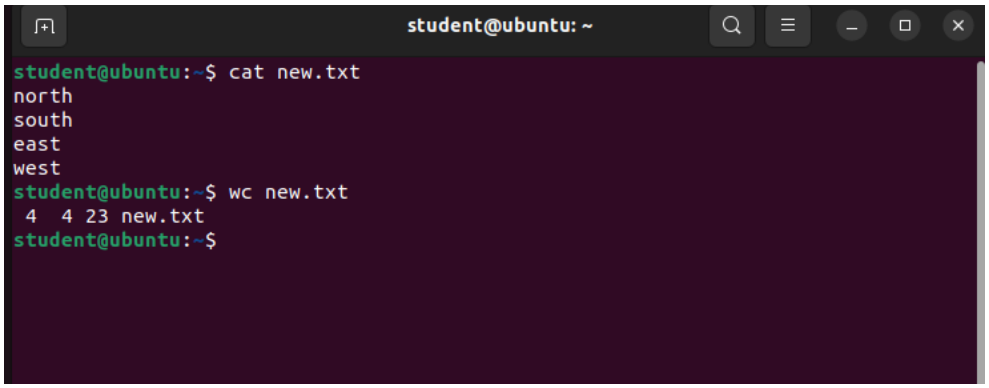
18) Cp: used to copy files and directories from one location to another



19) Sort: sort a file, arrange the records in a particular order



20) Wc: count the number of lines, words, and characters (bytes) in a file



21) Chmod: used to change the permissions of files and directories, controlling who can read, write, and execute them

```
student@ubuntu:~$ chmod -x new.txt
student@ubuntu:~$ ls -l
total 264
drwxrwxr-x 2 student student 4096 Apr  8  2024  501
drwxrwxr-x 2 student student 4096 Mar  1  2024  503
-rw-rw-r-- 1 student student  228 Mar  5  2024  A7.sh
drwxr-xr-x 2 root    root    4096 Jan 21 10:11  aditi
-rwxrwxr-x 1 student student  137 Jan 24  2024  ar2.sh
-rw-rw-r-- 1 student student  145 Feb  2  2024 'area&peri.sh'
-rwxrwxr-x 1 student student  416 Jan 24  2024  arithematic.sh
-rw-rw-r-- 1 student student  823 Feb 13  2024  as4.sh
-rw-rw-r-- 1 student student  237 Apr  1  2024  assign6a.sh
-rw-rw-r-- 1 student student  233 Apr  1  2024  assign6.sh
-rw-rw-r-- 1 student student  280 Jan 31  2024  assignment4.sh
-rw-rw-r-- 1 student student  177 Feb 14  2024  assignment5.sh
-rw-rw-r-- 1 student student  244 Jan 31  2024  assign.sh
-rw-rw-r-- 1 student student  214 Jan 31  2024  assig.sh
-rw-r--r-- 1 root    root     15 Jan 21 10:32  a.txt
-rw-r--r-- 1 root    root     15 Jan 21 10:33  b.txt
-rw-rw-r-- 1 student student  211 Feb 27  2024  case1.sh
-rw-rw-r-- 1 student student  221 Apr 30  2024  char.sh
-rw-rw-r-- 1 student student  145 Feb  2  2024  circle.sh
-rw-rw-r-- 1 student student  122 Feb 14  2024  conditional.sh
drwxr-xr-x 3 student student 4096 May  2  2024  Desktop
-rw-rw-r-- 1 student student  386 Mar 18  2024  digit.sh
drwxr-xr-x 6 student student 4096 Apr 30  2024  Documents
drwxr-xr-x 2 student student 4096 Apr  1  2024  Downloads
-rw-rw-r-- 1 student student  117 Feb  5  2024  emp.sh
-rw-r--r-- 1 root    root     33 Jan 21 10:20  example.txt
-rw-rw-r-- 1 student student  595 Feb 19  2024  exp3.sh
-rw-rw-r-- 1 student student  288 Feb 26  2024  exp5.1.sh
```

Conclusion: (It should be handwritten)

Linux commands provide powerful tools to manage files, folders, and system information very effectively. From navigation (pwd, cd, ls) to file operations (touch, mkdir, rm, cp), and system uses (ps, who, time), they form the base for learning the Linux environment. Commands like chmod and man help with permissions and learning command usage, while sort, cat, wc, head, and tail are useful for processing and analyzing data.

LAB Manual

Name of Student : Aditi Bansal	PRN:23070122013
Semester : IV	**Year AY 24-25**
Subject Title: Operating Systems Lab	
EXPERIMENT No : 3	Assignment No: 4
TITLE : Basic Shell script for File operation	DoP :11-02-2025

Aim : Implement and demonstrate a shell script on file operations

Learning Outcome: *1. To understand the shell script for file operation*

2. To Demonstrate the shell script for file operations

***Hardware/Software*:**

Problem Statement:

- Create file
- Enter the content of the file
- Display the content of the file
- Display the number of lines, words and characters in the file
- Sort the content of the file
- Transform the content of the files from lowercase to uppercase
- Search in the file
- Cut position
- Rename the file
- Delete the file

Procedure:

The steps to implement a basic shell script for file operations are:
1. Create a new file.
2. Edit the file using Text Editor and add the commands required for performing the desired file operations.
3. Save and close the file. The file extension should be .sh
4. Using the terminal, edit the permissions of the file to be executable using the **chmod** command
5. Run the script in the terminal using ./filename.sh
6. Based on the operations, provide the necessary inputs for the file operations.

Shell Script:

```
#! /bin/sh
echo "Enter filename:"
read file
touch $file
echo "File $file created successfully"
echo " "

echo "Enter the content of the file:"
cat>$file
echo " "

echo "Displaying file content and word count:"
cat $file
wc $file
echo " "

echo "Displaying the content in uppercase:"
cat $file | tr [:lower:] [:upper:]
echo " "

echo "Enter number of lines to be displayed:"
read no_of_lines
echo "The content of the first $no_of_lines lines is "
head -n $no_of_lines $file
echo " "

echo "Enter the word to be searched:"
read word
grep $word $file
echo " "

echo "Enter the cut position:"
read cut_position
cut -c $cut_position $file
echo " "

echo "Enter the new filename:"
read new
mv -f $file $new
echo "File renamed successfully from $file to $new"
echo " "

echo "Enter the file to be removed:"
read file_name
rm $file_name
echo "File $file_name removed successfully"
```

Output: snapshots of the demonstration

```
student@ubuntu:~$ bash ans1.sh
Enter filename:
new_file.txt
File new_file.txt created successfully
```

```
Enter the content of the file:
Name:Aditi
PRN:23070122013
Date: 11-02-2025
Experiment 3 Assignment 4
Learning Ubuntu Linux
Implementing and demonstrating shell script for file operations
```

```
  6  19 157 new_file.txt
```

```
Displaying the content in uppercase:
NAME:ADITI
PRN:23070122013
DATE: 11-02-2025
EXPERIMENT 3 ASSIGNMENT 4
LEARNING UBUNTU LINUX
IMPLEMENTING AND DEMONSTRATING SHELL SCRIPT FOR FILE OPERATIONS
```

```
Enter number of lines to be displayed:
3
The content of the first 3 lines is
Name:Aditi
PRN:23070122013
Date: 11-02-2025
```

```
Enter the word to be searched:
shell
Implementing and demonstrating shell script for file operations
```

```
Enter the cut position:
3
m
N
t
p
a
p
```

```
Enter the new filename:
renamed_file.txt
File renamed successfully from new_file.txt to renamed_file.txt
```

```
Enter the file to be removed:
ques1.sh
File ques1.sh removed successfully
```

```
student@ubuntu:~$ ls
 501                conditional.sh   Music              sa.sh
 503                Desktop          newf               scriptsample.sh
 A7.sh              digit.sh         new_file           scripttest.sh
 aditi              Documents        newfolder          snap
 ans1.sh            Downloads        NewFolder1         state.txt
 ar2.sh             emp.sh           new.sh             string1.sh
'area&peri.sh'      example.txt      new.txt            string_ops.sh
 arithematic.sh     exp3.sh          nn                 string.sh
 as4.sh             exp5.1.sh        Operations.sh      strings.sh
 assign6a.sh        exp5.2.sh        p1.sh              strings.sh.save
 assign6.sh         exp5.3.sh        p2.sh              Templates
 Assignment3        exp5.4.sh        Pictures           test1.txt
 assignment4.sh     exp5.sh          prac_183.sh        test2.sh
 assignment5.sh     experi3.sh       practical_183.sh   test.sh
 assign.sh          file1.txt        Public             test.txt
 assig.sh           file2.txt        ques2.sh           Videos
 a.txt              file_exp3.txt    ques3.sh           vowels
 b.txt              final.sh         ques4.sh           vowels.sh
 case1.sh           forloop.sh       renamed_file.txt   while.sh
 char.sh            for.sh           root.sh
 circle.sh          mean.sh          salary.sh
```

LAB Manual

Name of Student : Aditi Bansal	**PRN:23070122013**
Semester : IV	**Year AY 23-24**
Subject Title: Operating Systems Lab	
EXPERIMENT No : 4	**Assignment No: 5**
TITLE : Arithmetic Operations	**DoP : 04-02-2025**

Aim: Implement and demonstrate arithmetic operations

Learning Outcomes: *1. To understand the arithmetic operation*

2. To Demonstrate the arithmetic operations using Linux command.

***Hardware/Software*: (to be handwritten)**

Problem Definition:

A program should be written where all the following conditions should be satisfied.

1. Perform basic operations like Addition, Subtraction, Multiplication, and Division. Accept input from a user.

2. Calculate the area and perimeter of the circle. Accept radius from a user. (Assume pi=3.14)

3. Calculate employees' gross salary, where DA is 1.65% of the basic salary, and HRA is 0.30% of the basic salary. Accept the value of basic salary from a user.

4. Calculate the mean salary given by a company if the basic salary of employees A, B, C, and D are Rs.1200, 1400, 1350, and 1800 respectively.

Theory: handwritten

Arithmetic operations are the most basic in any kind of programming language. Linux or Unix operating system provides the bc command and expr command for doing arithmetic calculations. expr stands for "expression" and allows for the evaluation of values and returns the result to standard output. It is particularly useful in scripts for handling both numerical and string data efficiently. In short, it helps in:

- Basic operations like addition, subtraction, multiplication, division, and modulus on integers.
- Evaluating regular expressions, string operations like substring, length of strings etc.

bc command is used for command line calculator. It is similar to basic calculator by using which we can do basic mathematical calculations.

Algorithm : softcopy format

1.

Step1: Start
Step2: Read 2 numbers from the user
Step3: Perform operations- addition, subtraction, multiplication and division using commands:
`expr $a + $b`
`expr $a - $b`
`expr $a \* $b`
`expr $a / $b`
Step4: Print the outputs
Step5: Stop

2.

Step1: Start
Step2: Read the value of radius from user
Step3: Initialize pi as 3.14
Step4: Calculate area and perimeter of circle suing commands
$(echo "$pi * $radius * $radius" | bc -l)
$(echo "2 * $pi * $radius" | bc -l)
Step5: Print the output
Step6: Stop

3.

Step1: Start
Step2: Read basic salary from the user and store in variable 'basic'.
Step3: Calculate DA using command
$(echo "scale=2 ; $basic *165/100" | bc -l)
Step4: Calculate HRA using command
$(echo "scale=2 ; $basic *30/100" | bc -l)
Step5: Calculate gross salary i.e. sum of basic, DA and HRA
Step6: Print gross salary
Step7: Stop

4.

Step1: Start
Step2: Initialize A, B, C, D with the provided salaries
Step3: Calculate mean of the salaries using (A+B+C+D)/4 using command
$(echo "scale=2 ; ($A+$B+$C+$D) / 4" | bc -l)
Step4: Print the mean salary
Step5: Stop

Program: Softcopy format

```
#! /bin/sh
echo "Enter your first number:"
read a
echo "Enter you second number:"
read b
echo "Sum= `expr $a + $b`"
echo "Difference= `expr $a - $b`"
echo "Product= `expr $a \* $b`"
echo "Quotient= `expr $a / $b`"
```

ques2.sh

```
#! /bin/sh
echo "Enter the radius of the circle:"
read radius
pi=3.14
echo "Area of the circle = $(echo "$pi * $radius * $radius"| bc -l)"
echo "Perimeter of the circle = $(echo "2 * $pi * $radius" |bc -l)"
```

```
#! /bin/sh
echo "Enter the basic salary:"
read basic
da=$(echo "scale=2 ; $basic * 165 / 100" | bc -l)
hra=$(echo "scale=2 ; $basic * 30 / 100" | bc -l)
gross=$(echo "scale=2 ; $basic + $da + $hra" | bc -l)
echo "Gross salary = $gross"
```

```
#! /bin/sh
A=1200
B=1400
C=1350
D=1800
echo "The salary of A=$A"
echo "The salary of B=$B"
echo "The salary of C=$C"
echo "The salary of D=$D"
mean_salary=$(echo "scale=2 ; ($A+$B+$C+$D) / 4" | bc -l)
echo "Mean salary given by company = $mean_salary"
```

Output: snapshots of the demonstration

```
student@ubuntu:~$ bash ques1.sh
Enter your first number:
6
Enter you second number:
2
Sum= 8
Difference= 4
Product= 12
Quotient= 3
student@ubuntu:~$
```

```
student@ubuntu:~$ bash ques2.sh
Enter the radius of the circle:
7
Area of the circle = 153.86
Perimeter of the circle = 43.96
student@ubuntu:~$
```

```
student@ubuntu:~$ bash ques3.sh
Enter the basic salary:
10000
Gross salary = 29500.00
student@ubuntu:~$
```

```
student@ubuntu:~$ bash ques4.sh
The salary of A=1200
The salary of B=1400
The salary of C=1350
The salary of D=1800
Mean salary given by company = 1437.50
student@ubuntu:~$
```

LAB Manual

Name of Student: Aditi Bansal	**PRN: 23070122013**
Semester: IV	**Year AY 24-25**
Subject Title: Operating Systems Lab	
EXPERIMENT No: 5	**Assignment No : 7**
TITLE: Conditional Statement	**DoP : 18-02-2025**

Aim: Implement shell script to demonstrate conditional statements

Learning Outcomes: *1. To understand the* conditional statements

2. To Demonstrate the shell script to demonstrate conditional statements *using Linux command.*

***Hardware/Software*: handwritten**

Problem Definition:

a) Shell script to print whether the number entered by the user is even or odd

b) Shell script to print the largest of three numbers

c) Shell script to print whether the year entered by the user is leap year or not

d) Shell script to calculate balance of the account based on the following conditions:

i. Accept the account balance from the user

ii. Accept withdrawal amount from the user

iii. If withdrawal amount < 1500 then calculate the tax as 3% of the withdrawal amount

iv. If withdrawal amount >1500 and less than 3000 then calculate the tax as 4% of the withdrawal amount

v. If withdrawal amount > 3000 then calculate the tax as 5% of the withdrawal amount

vi. If balance is less than withdrawal amount then print insufficient balance

vii. Print amount withdrawn along with tax deducted

Theory: Write theory about Conditional statements:

- if statement
- if-else statement
- if..elif..else..fi statement (Else If ladder)
- if..then..else..if..then..fi..fi..(Nested if)

1)

```
if [ expression ]
then
    statement
fi
```

2)

```
if [ condition ]
then
    statements
else
   statements
fi
```

3)

```
if [ expression 1 ]  // if [ $arg1 == $ag2 -a $arg1 != $arg3 ]
then
    Statement(s) to be executed if expression 1 is true
elif [ expression 2 ]
then
    Statement(s) to be executed if expression 2 is true
elif [ expression 3 ]
then
   Statement(s) to be executed if expression 3 is true
else
  Statement(s) to be executed if no expression is true
fi
```

4)

```
if [ expression 1 ]
then
   statement1
   statement2
   .
else
   if [ expression 2]
  then
       statement3
       .
  fi
fi
```

Program: softcopy

```
#! /bin/sh
echo "Enter a number:"
read num
if [ `expr $num % 2 ` == 0 ]
then
    echo "$num is even"
else
    echo "$num is odd"
fi
```

```
#! /bin/sh
echo "Enter three numbers:"
read a
read b
read c
if [ "$a" -gt "$b" ] && [ "$a" -gt "$c" ]
then
        echo "$a is largest"
elif [ "$b" -gt "$a" ] && [ "$b" -gt "$c" ]
then
        echo "$b is largest"
else
        echo "$c is largest"
fi
```

```
#! /bin/sh
echo "Enter year to check whether its leap year or not:"
read year
if [ $((year % 4)) == 0 ]
then
        if [ $((year % 400)) == 0 ] || [ $((year % 100)) != 0 ]
        then
                echo "$year is a leap year"
        else
                echo "$year is not a leap year"
        fi
else
        echo "$year is not a leap year"
fi
```

```
#! /bin/sh
echo "Enter account balance:"
read bal
echo "Enter withdrawal amount:"
read withdraw
if [ "$withdraw" -gt "$bal" ]
then
        echo "Insufficient balance."
        exit
fi
if [ "$withdraw" -lt 1500 ]
then
        tax=$(echo "scale=2; $withdraw*0.03" | bc)
elif [ "$withdraw" -ge 1500 ] && [ "$withdraw" -lt 3000 ]
then
        tax=$(echo "scale=2; $withdraw*0.04" | bc)
else
        tax=$(echo "scale=2; $withdraw*0.05" | bc)
fi

amt=$(echo "scale=2; $withdraw-$tax" | bc)
bal=$(echo "scale=2; $bal-$amt" | bc)

echo "Tax Deducted: Rs. $tax"
echo "Amount withdrawn: Rs. $amt"
echo "New balance: Rs. $bal"
```

Steps to execute the program: handwritten

Output: snapshots of the demonstration softcopy

```
student@ubuntu:~$ bash even_odd.sh
Enter a number:
3
3 is odd
student@ubuntu:~$ bash even_odd.sh
Enter a number:
8
8 is even
```

```
student@ubuntu:~$ bash largest_3.sh
Enter three numbers:
3
4
5
5 is largest
student@ubuntu:~$ bash largest_3.sh
Enter three numbers:
1
2
0
2 is largest
student@ubuntu:~$ bash largest_3.sh
Enter three numbers:
5
4
3
5 is largest
```

```
student@ubuntu:~$ bash leap_year.sh
Enter year to check whether its leap year or not:
2025
2025 is not a leap year
student@ubuntu:~$ bash leap_year.sh
Enter year to check whether its leap year or not:
2024
2024 is a leap year
```

```
student@ubuntu:~$ bash tax.sh
Enter account balance:
12000
Enter withdrawal amount:
1800
Tax Deducted: Rs. 72.00
Amount withdrawn: Rs. 1728.00
New balance: Rs. 10272.00
student@ubuntu:~$ bash tax.sh
Enter account balance:
12000
Enter withdrawal amount:
3500
Tax Deducted: Rs. 175.00
Amount withdrawn: Rs. 3325.00
New balance: Rs. 8675.00
student@ubuntu:~$
```

LAB Manual

Name of Student: Aditi Bansal	**PRN: 23070122013**
Semester: IV	**Year AY 24-25**
Subject Title: Operating Systems Lab	
EXPERIMENT No: 6	**Assignment No : 7**
TITLE: Loop Statement	**DoP : 25-02-2025**

Aim: Implement shell script to demonstrate Loop statements

Learning Outcomes: *1. To understand the* control statements

2. To Demonstrate the shell script to demonstrate control statements *using Linux command.*

***Hardware/Software*: Handwritten**

Problem Definition: Implementing for Loop

1. Shell Script using for loop to print the pattern: (ask user to enter value of n)

```
*
**
***
*****
****
***
**
*
```

2. Print pattern:

```
        *
      * * *
    * * * * *
  * * * * * * *
* * * * * * * * *
```

3. shell script to print 8*8 chess board pattern using alternating colours “\033[47m” for light grey and “\033[40m” for black.

4. shell script using for loop which considers Day 1 as Mon(weekday) and accordingly considers Day 7 as Sun(weekend) and print the wage Rs 350 for weekday and Rs 550 for weekend.

Theory: like Control Structure For loop, while loop handwritten

Syntax:

for loop:

```
#Start of for loop
for a in list
do
        …
        …
        <Statements>
        …
done
```

while loop:

```
while [condition]
do
        …
        …
        <statements>
        …
done
```

Program: Softcopy

```
#! /bin/sh
echo "Enter number of columns:"
read n
for ((i=0;i<n;i++))
do
        for ((j=0;j<i+1;j++))
        do
                echo -n "*"
        done
        echo
done
for ((i=n-1;i>=0;i--))
do
        for ((j=i;j>0;j--))
        do
                echo -n "*"
        done
        echo
done

```

```
#! /bin/sh
echo "Enter number of rows:"
read n
for ((i=0;i<n;i++))
do
        for ((j=n-i-1;j>=0;j--))
        do
                echo -n " "
        done
        for ((k=0;k<i+1;k++))
        do
                echo -n "* "
        done
        echo
done


```

```
#! /bin/sh
for ((i=0;i<8;i++))
do
        for ((j=0;j<8;j++))
        do
                if (( (i+j) % 2 == 0)); then
                        echo -n -e "\033[47m  \033[0m"
                else
                        echo -n -e "\033[40m  \033[0m"
                fi
        done
        echo
done

```

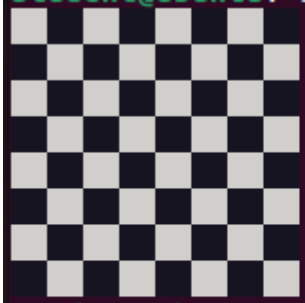
```
#! /bin/sh
for day in {1,2,3,4,5,6,7}
do
        if [ $day -eq 6 ] || [ $day -eq 7 ]; then
                wage=550
        else
                wage=350
        fi
        echo "Day $day : Rs. $wage"
done
```

Output: snapshots of the demonstration sofycopy

```
student@ubuntu:~$ bash loop.sh
Enter number of columns:
5
*
**
***
****
*****
****
***
**
*

student@ubuntu:~$ bash loop.sh
Enter number of columns:
7
*
**
***
****
*****
******
*******
******
*****
****
***
**
*

student@ubuntu:~$
```

```
student@ubuntu:~$ bash loop2.sh
Enter number of rows:
6
      *
     * *
    * * *
   * * * *
  * * * * *
 * * * * * *
student@ubuntu:~$
```

```
student@ubuntu:~$ bash loop3.sh
```



```
student@ubuntu:~$
student@ubuntu:~$
student@ubuntu:~$
```

```
student@ubuntu:~$ bash loop4.sh
Day 1 : Rs. 350
Day 2 : Rs. 350
Day 3 : Rs. 350
Day 4 : Rs. 350
Day 5 : Rs. 350
Day 6 : Rs. 550
Day 7 : Rs. 550
```

LAB Manual

Name of Student: Aditi Bansal	**PRN: 23070122013**
Semester: IV	**Year AY 24-25**
Subject Title: Operating Systems Lab	
EXPERIMENT No: 7	**Assignment No : 7**
TITLE: Switch Statement	**DoP : 04-03-2025**

Aim: Implement shell script to demonstrate **case…esac** statements

Learning Outcomes: *1. To understand the* **case…esac** statements

2. To Demonstrate the shell script to demonstrate **case…esac** statements *using Linux command.*

***Hardware/Software*: Handwritten**

Theory:

case variable in

pattern1)

 # commands to execute if variable matches pattern1

;;

pattern2)

 # commands to execute if variable matches pattern2

;;

pattern3)

 # commands to execute if variable matches pattern3

;;

**)*

 # default commands to execute if variable doesn't match any pattern

;;

esac

Program:

student@ubuntu: ~

Open *character.sh ~/

```
#!/bin/sh
echo "eneter the character"
read c
case $c in
['a','e','i','o','u','A','I','O','U','E'])
echo "vowel"
;;
*)ss
echo "consonant"
;;
esac
```

Open

```
#!/bin/sh
echo "enter the digit"
read digit
case $digit in
[0-9])
echo "$digit is single digit"
;;
[0][0-9])
echo "$digit is  digit"
;;
[1-9][0-9])
echo "$digit is double digit"
;;
[1-9][0-9][0-9])
echo "$digit is 3 digit number"
;;
*)
esac
```

```
#!/bin/sh
echo "enter file name"
read file
case $file in
*.dir)
echo "its a directory"
;;
*.sh)
echo "its a shell file"
;;
*.doc)
echo "it a word document"
esac
```

Open

```
#!/bin/bash
echo "enter the character"
read c
case $c in
['a'-'z'])
echo "lower case"
;;
['A'-'Z'])
echo "upper case"
;;
[0-9])
echo "digit"
;;
*)
echo "special symbol"
;;
esac
```

```
#!/bin/sh
echo "enter vehicle name"
read v
case $v in
car|bike|scooter])
echo "personal vehicle"
;;
truck|tractor|crane])
echo "industrial vehicle"
;;
auto|bus])
echo "public vehicle"
;;
*)
echo "unknown category"
;;
esac
```

```
#!/bin/sh
echo "enter fruit name"
read v
case $v in
apple|mango|banana])
echo "common fruit"
;;
strawberry|blueberry])
echo "berries"
;;
coconut|walnut)
echo "dry fruit"
;;
*)
echo "unknown category"
;;
esac
```

```
#!/bin/sh
echo "enter the month"
read month
case $month in
december|january|february|12|1|2)
echo "winter"
;;
march|may|april|3|4|5)
echo "spring"
;;
june|july|august|6|7|8)
echo "summer"
;;
september|october|november|9|10|11)
echo "autumn"
;;
*)
;;
esac
```

Open

```
#!/bin/sh

echo "Enter a day of the week:"
read day

case $day in
    Monday|Tuesday|Wednesday|Thursday|Friday)
        echo "It's a weekday."
        ;;
    Saturday|Sunday)
        echo "It's the weekend!"
        ;;
    *)
        echo "Invalid input. Please enter a valid day."
        ;;
esac
```

Output:

```
student@ubuntu:~$ bash character.sh;
eneter the character
a
vowel
student@ubuntu:~$ bash character.sh;
eneter the character
q
consonant
```

```
student@ubuntu:~$ bash digit.sh
enter the digit
123
123 is 3 digit number
student@ubuntu:~$ bash digit.sh
enter the digit
1
1 is single digit
student@ubuntu:~$
```

```
student@ubuntu:~$ gedit fletype.sh
student@ubuntu:~$ bash fletype.sh;
enter file name
anu.doc
it a word document
student@ubuntu:~$
```

```
enter the character
1
digit
student@ubuntu:~$ bash file.sh
enter the character
a
lower case
student@ubuntu:~$ bash file.sh
enter the character
A
upper case
student@ubuntu:~$
```

```
student@ubuntu:~$ bash vehicle.sh;
enter vehicle name
car
personal vehicle
student@ubuntu:~$ gedit vehicle.sh;
```

```
student@ubuntu:~$ bash fruit.sh
enter fruit name
apple
common fruit
student@ubuntu:~$
```

```
student@ubuntu:~$ bash month.sh;
enter the month
april
spring
student@ubuntu:~$ bash month.sh;
enter the month
4
spring
student@ubuntu:~$ gedit file.sh
```

```
student@ubuntu:~$ bash week.sh;
Enter a day of the week:
Saturday
It's the weekend!
student@ubuntu:~$
```

LAB Manual

Name of Student: Aditi Bansal	**PRN: 23070122013**
Semester: IV	**Year AY 24-25**
Subject Title: Operating Systems Lab	
EXPERIMENT No: 8	**Assignment No : 1**
TITLE: Scheduling Algorithm	**DoP : 25-03-2025**

Aim: Implement C program demonstrate **FCFS algorithm**

Learning Outcomes: *1. To understand the* **scheduling algorithm**

2. To Demonstrate the FCFS algorithm

***Hardware/Software*: Handwritten**

Problem Statement :

Process	Brust (execution time)
P1	24
P2	3
P3	7
P4	13
P5	21

Theory:Handwritten

Algorithm: Soft copy

Step 1: Initialize arrays for burst_time[], waiting_time[], turnaround_time[], arrival_time[], and processes[]. Also, initialize variables for total_waiting_time and total_turnaround_time.

Step 2: Calculate the waiting time for each process. If it's the first process, set waiting_time[0] = 0. For subsequent processes, use the formula

waiting_time[i] = waiting_time[i-1] + burst_time[i-1] - arrival_time[i].

Step 3: calculate the turnaround time for each process using

turnaround_time[i] = burst_time[i] + waiting_time[i].

Step 4: calculate the average waiting time and average turnaround time by dividing the total waiting time and total turnaround time by the number of processes.

Step 5: the main function takes input for the processes, their burst times, and arrival times, and calls the above functions to compute the times and averages.

Step 6: Stop

Program:Softcopy

```cpp
#include<iostream>
using namespace std;

int burst_time[100], n, waiting_time[100], turnaround_time[100], total_waiting_time, total_turnaround_time, avg_turnaround_time, avg_waiting_time, processes[100], arrival_time[100];

void WaitingTime(int processes[], int n, int burst_time[], int arrival_time[], int waiting_time[])
{
    waiting_time[0] = 0;
    for (int i = 1; i < n; i++) {
        waiting_time[i] = burst_time[i - 1] + waiting_time[i - 1] - arrival_time[i];
        if (waiting_time[i] < 0) {
            waiting_time[i] = 0;
        }
    }
}

void TurnAroundTime(int processes[], int n, int burst_time[], int waiting_time[], int turnaround_time[])
{
    for (int i = 0; i < n; i++) {
        turnaround_time[i] = burst_time[i] + waiting_time[i];
    }
}

void findavgTime(int processes[], int n, int burst_time[], int arrival_time[])
{
    int waiting_time[n], turnaround_time[n], total_waiting_time = 0, total_turnaround_time = 0;
    WaitingTime(processes, n, burst_time, arrival_time, waiting_time);
    TurnAroundTime(processes, n, burst_time, waiting_time, turnaround_time);

    cout << "Processes\tBurst time\tArrival Time\tWaiting Time\tTurnaround Time\n";
    for (int i = 0; i < n; i++) {
        total_waiting_time += waiting_time[i];
        total_turnaround_time += turnaround_time[i];
        cout << "   " << i + 1 << "\t\t" << burst_time[i] << "\t\t" << arrival_time[i]
             << "\t\t" << waiting_time[i] << "\t\t" << turnaround_time[i] << endl;
    }

    avg_turnaround_time = (float)(total_turnaround_time) / n;
    avg_waiting_time = (float)(total_waiting_time) / n;
    cout << "Avg Turnaround Time: " << avg_turnaround_time << endl;
    cout << "Avg Waiting Time: " << avg_waiting_time << endl;
}
```

```cpp
int main()
{
    cout << "Enter number of processes: ";
    cin >> n;

    cout << "Enter the name of the processes: ";
    for (int i = 0; i < n; i++)
        cin >> processes[i];

    cout << "Enter burst time for each process: ";
    for (int i = 0; i < n; i++)
        cin >> burst_time[i];

    cout << "Enter arrival time for each process: ";
    for (int i = 0; i < n; i++) {
        cin >> arrival_time[i];
        if (arrival_time[i] < 0) {
            arrival_time[i] = 0;
        }
    }

    findavgTime(processes, n, burst_time, arrival_time);
}
```

Output: snapshots of the demonstration

```
Enter number of processes: 3
Enter the name of the processes: 1 2 3
Enter burst time for each process: 24 3 3
Enter arrival time for each process: 0 0 0
Processes       Burst time      Arrival Time    Waiting Time    Turnaround Time
   1            24              0               0               24
   2            3               0               24              27
   3            3               0               27              30
Avg Turnaround Time: 27
Avg Waiting Time: 17

--------------------------------
Process exited after 11.18 seconds with return value 0
Press any key to continue . . .
```

```
Enter number of processes: 4
Enter the name of the processes: 1 2 3 4
Enter burst time for each process: 1 100 1 100
Enter arrival time for each process: 0 1 2 3
Processes       Burst time      Arrival Time    Waiting Time    Turnaround Time
   1            1               0               0               1
   2            100             1               0               100
   3            1               2               98              99
   4            100             3               96              196
Avg Turnaround Time: 99
Avg Waiting Time: 48.5

--------------------------------
Process exited after 11.56 seconds with return value 0
Press any key to continue . . .
```

Name of Student: Aditi Bansal	**PRN:23070122013**
Semester: IV	**Year AY 24-25**
Subject Title: Operating Systems Lab	
EXPERIMENT No: 9	**Assignment No : 1**
TITLE: Scheduling Algorithm	**DoP : 01-04-2025**

Aim: Implement C program demonstrate **Priority algorithm**

Learning Outcomes: *1. To understand the* **scheduling algorithm**

2. To Demonstrate the Priority algorithm

***Hardware/Software*: Handwritten**

Theory: Handwritten

Algorithm: Softcopy

- Input the number of processes (n), then for each process, input Arrival Time (AT), Burst Time (BT), and Priority, initializing Remaining Time = BT and isCompleted = false.
- Initialize variables: currentTime = 0, completed = 0, prev = -1, totalTAT = 0, and totalWT = 0.
- At each time unit, find the process with the highest priority (lowest priority value) that has arrived (AT <= currentTime) and is not completed; if multiple processes have the same priority, select the one with the earliest arrival time.
- Execute the selected process for 1 time unit; if it starts for the first time, set Start Time (ST) = currentTime, then decrement remainingTime and increment currentTime.
- If a process completes (remainingTime == 0), set End Time (ET) = currentTime, calculate Turnaround Time (TAT) = ET - AT, Waiting Time (WT) = TAT - BT, mark it as completed, and increment the completed count.
- Repeat steps 3-5 until all processes are completed, then display the process details in a table, compute Average Turnaround Time (TAT) and Average Waiting Time (WT), and print the results.

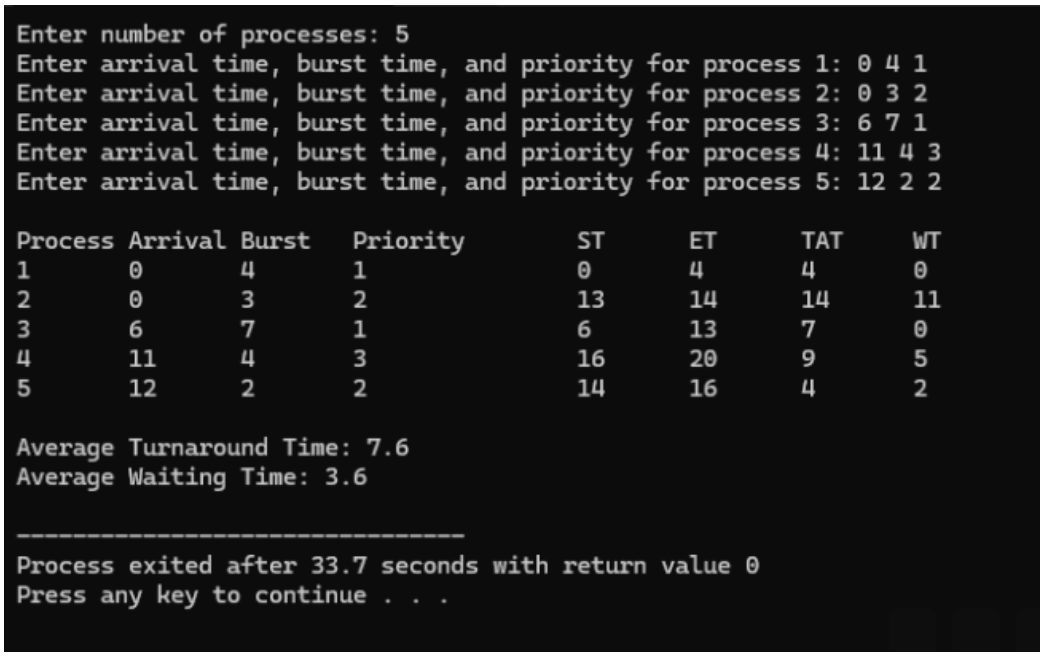
Program:

```cpp
#include <iostream>
#include <climits>
using namespace std;

class Process {
public:
    int id, AT, BT, priority;
    int ST, ET, remainingTime;
    int TAT, WT;
    bool isCompleted;

    Process() {
        id = AT = BT = priority = 0;
        ST = ET = remainingTime = 0;
        TAT = WT = 0;
        isCompleted = false;
    }
};

int main() {
    int n;
    cout << "Enter number of processes: ";
    cin >> n;

    Process p[n];

    for (int i = 0; i < n; i++) {
        p[i].id = i + 1;
        cout << "Enter arrival time, burst time, and priority for process " << i + 1 << ": ";
        cin >> p[i].AT >> p[i].BT >> p[i].priority;
        p[i].remainingTime = p[i].BT;
    }

    int completed = 0, currentTime = 0, prev = -1;
    float totalTAT = 0, totalWT = 0;
```

```cpp
while (completed != n) {
    int idx = -1, highestPriority = INT_MAX;

    for (int i = 0; i < n; i++) {
        if (p[i].AT <= currentTime && !p[i].isCompleted) {
            if (p[i].priority < highestPriority || (p[i].priority == highestPriority && p[i].AT < p[idx].AT)) {
                highestPriority = p[i].priority;
                idx = i;
            }
        }
    }

    if (idx != -1) {
        if (prev != idx) {
            p[idx].ST = currentTime;
        }

        p[idx].remainingTime--;
        currentTime++;

        if (p[idx].remainingTime == 0) {
            p[idx].ET = currentTime;
            p[idx].TAT = p[idx].ET - p[idx].AT;
            p[idx].WT = p[idx].TAT - p[idx].BT;
            p[idx].isCompleted = true;
            totalTAT += p[idx].TAT;
            totalWT += p[idx].WT;
            completed++;
        }
        prev = idx;
    } else {
        currentTime++;
    }
}
```

```cpp
    cout << "\nProcess\tArrival\tBurst\tPriority\tST\tET\tTAT\tWT\n";
    for (int i = 0; i < n; i++) {
        cout << p[i].id << "\t" << p[i].AT << "\t" << p[i].BT << "\t"
             << p[i].priority << "\t\t" << p[i].ST << "\t" << p[i].ET << "\t"
             << p[i].TAT << "\t" << p[i].WT << "\n";
    }

    cout << "\nAverage Turnaround Time: " << totalTAT / n << "\n";
    cout << "Average Waiting Time: " << totalWT / n << "\n";

    return 0;
}
```

Output: snapshots of the demonstration



Name of Student: Aditi Bansal	**PRN: 23070122013**
Semester: IV	**Year AY 24-25**
Subject Title: Operating Systems Lab	
EXPERIMENT No: 12	**Assignment No : 6**
TITLE: Banker's algorithm	**DoP : 09-04-2025**

Aim: Implement C program demonstrate **Banker's algorithm**

Learning Outcomes: *1. To understand the* **deadlock concept**

2. To Demonstrate the **Banker's** algorithm

***Hardware/Software*: Handwritten**

Problem Statement

- Answer the following questions using Banker's algorithm
- A) What is the content of the matrix need?
- B) Is the system is in safe state
- If a request from process P1 arrives for(1,0,2)can the request be granted immediately?

PID	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
P0	0	1	0	7	5	3	3	3	2
P1	2	0	0	3	2	2			
P2	3	0	2	9	0	2			
P3	2	1	1	2	2	2			
P4	0	0	2	4	3	3			

Theory: Handwritten

Algorithm: Softcopy

Step 1 : Input system data

Take input for the number of processes and resources. Then input the Allocation matrix, Request (or Need) matrix, and the Available resource vector.

Step 2: Initialize variables

Copy the Available vector into a Work array. Set all values in the Finish array to false. Initialize an empty Safe Sequence list.

Step 3: Find an eligible process

Search for a process that is not finished and whose request can be satisfied by the current Work resources.

Step 4: Simulate execution

If such a process is found, add its allocated resources back to Work, mark it as finished, and add it to the Safe Sequence.

Step 5: Repeat search

Repeat Steps 3 and 4 until all processes are finished or no eligible process is found.

Step 6: Check system date

If all processes are finished, the system is in a safe state and the Safe Sequence is printed. Otherwise, it's an unsafe state.

Step 7: End

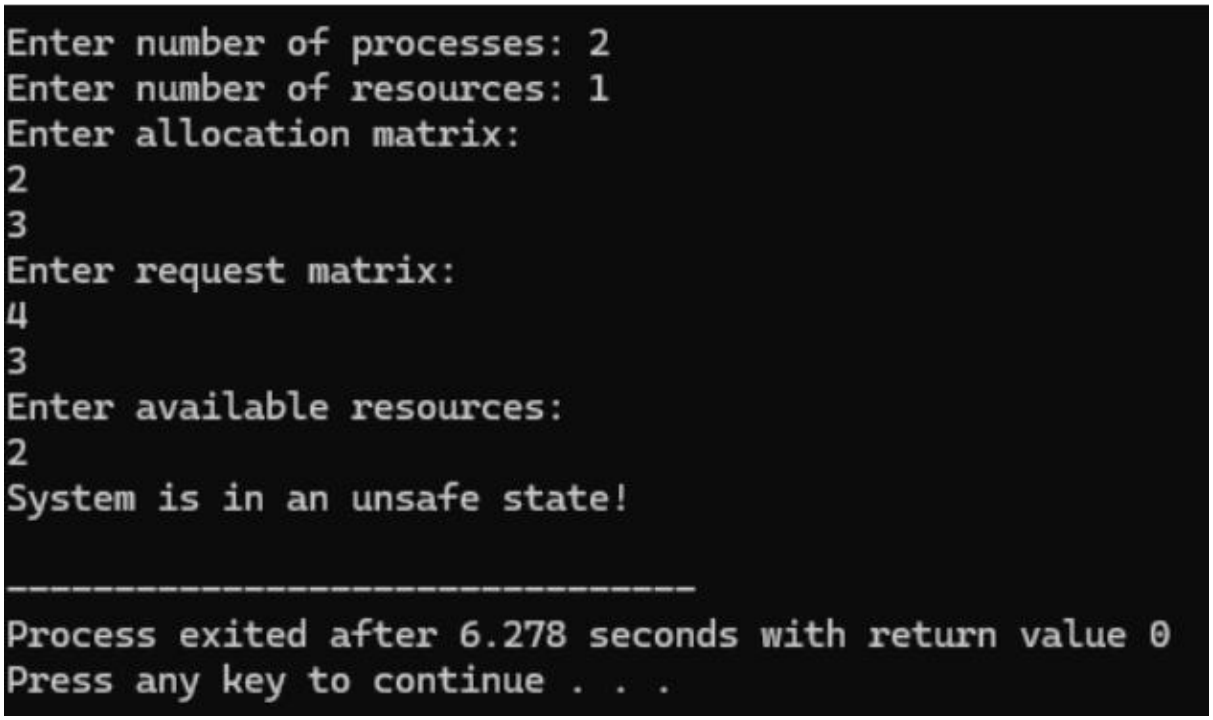
Program: Softcopy

```
    }
    cout << "System is in a safe state. Safe sequence is: ";
    for (int i = 0; i < numProcesses; i++) {
        cout << "P" << safeSequence[i] << " ";
    }
    cout << endl;

    return true;
}
int main() {
    int numProcesses, numResources;
    cout << "Enter number of processes: ";
    cin >> numProcesses;
    cout << "Enter number of resources: ";
    cin >> numResources;
    int allocation[10][10], request[10][10], available[10];
    cout << "Enter allocation matrix: \n";
    for (int i = 0; i < numProcesses; i++) {
        for (int j = 0; j < numResources; j++) {
            cin >> allocation[i][j];
        }
    }
    cout << "Enter request matrix: \n";
    for (int i = 0; i < numProcesses; i++) {
        for (int j = 0; j < numResources; j++) {
            cin >> request[i][j];
        }
    }
    cout << "Enter available resources: \n";
    for (int i = 0; i < numResources; i++) {
        cin >> available[i];
    }
    isSafeState(allocation, request, available, numProcesses, numResources);
    return 0;
}
```

bankers.cpp

```
#include <iostream>
#include <algorithm>

using namespace std;
bool isSafeState(int allocation[][10], int request[][10], int available[],
                 int numProcesses, int numResources) {
    int work[10];
    bool finish[10] = {false};
    int safeSequence[10];
    for (int i = 0; i < numResources; i++) {
        work[i] = available[i];
    }
    int count = 0;
    while (count < numProcesses) {
        bool found = false;
        for (int i = 0; i < numProcesses; i++) {
            if (!finish[i]) {
                bool canProceed = true;
                for (int j = 0; j < numResources; j++) {
                    if (request[i][j] > work[j]) {
                        canProceed = false;
                        break;
                    }
                }
                if (canProceed) {
                    for (int j = 0; j < numResources; j++) {
                        work[j] += allocation[i][j];
                    }
                    finish[i] = true;
                    safeSequence[count++] = i;
                    found = true;
                }
            }
        }
        if (!found) {
            cout << "System is in an unsafe state!" << endl;
            return false;
```

Output: snapshots of the demonstration



Name of Student:	**PRN**
Semester: IV	**Year AY 24-25**
Subject Title: Operating Systems Lab	
EXPERIMENT No: 12	**Assignment No : 8**
TITLE: Page Replacement Policy	**DoP :**

Aim: Implement C program demonstrate **Least Recently Used (LRU) Replacement Policy**

Learning Outcomes: *1. To understand the* **Least Recently Used algorithm**

2. To Demonstrate the **Least Recently Used** algorithm

***Hardware/Software*: Handwritten**

Problem Statement : F*ind page fault and page hit using LRU for the given page sequence*

No of frames 3

Consider reference string is 5,0,2,3,0,1,3,4,5,4,2,0,3,4,3

Algorithm:

- Input total number of pages and the page reference string.
- Input number of frames (memory slots).
- Initialize an empty list for frames (memory).
- Initialize a map to store the last used time for each page.
- For each page in the reference string:
- If the page is already in memory → Hit:
  Do nothing (no replacement needed).
- Else → Page Fault:
  If memory is not full, insert the page.
  If memory is full:
  Find the page in memory with the least recent use time.
  Replace it with the current page.
  Increment page fault counter.
- Update the last used time of the current page to the current index.
- After all pages are processed, output the total number of page faults.

Program:

```cpp
#include <iostream>
#include <vector>
#include <unordered_map>
using namespace std;

int main() {
    int n, frames;

    cout << "Enter the number of pages: ";
    cin >> n;

    vector<int> pages(n);
    cout << "Enter the page reference string: ";
    for (int i = 0; i < n; i++) {
        cin >> pages[i];
    }

    cout << "Enter the number of frames: ";
    cin >> frames;

    vector<int> memory;
    unordered_map<int, int> lastUsed;
    int pageFaults = 0;

    cout << "\nPage\tMemory State\n";

    for (int i = 0; i < n; i++) {
        int currentPage = pages[i];
        bool hit = false;

        // Check if page is already in memory
        for (int j = 0; j < memory.size(); j++) {
            if (memory[j] == currentPage) {
                hit = true;
                break;
            }
        }

        if (!hit) {
            if (memory.size() < frames) {
                memory.push_back(currentPage);
            } else {
                // Find LRU page
                int lruIndex = 0, lruTime = i;
                for (int j = 0; j < memory.size(); j++) {
                    if (lastUsed[memory[j]] < lruTime) {
                        lruTime = lastUsed[memory[j]];
                        lruIndex = j;
                    }
                }
                memory[lruIndex] = currentPage;
            }
            pageFaults++;
        }

        lastUsed[currentPage] = i;

        // Print memory state
        cout << currentPage << "\t";
        for (int val : memory)
            cout << val << " ";
        cout << endl;
    }

    cout << "\nTotal Page Faults: " << pageFaults << endl;

    return 0;
}
```

Output: snapshots of the demonstration

```
Enter the number of pages: 5
Enter the page reference string: 1 2 3 2 1
Enter the number of frames: 3

Page    Memory State
1       1
2       1 2
3       1 2 3
2       1 2 3
1       1 2 3

Total Page Faults: 3


...Program finished with exit code 0
Press ENTER to exit console.
```