EXP NO: **REG NO:**

DATE: **NAME:**

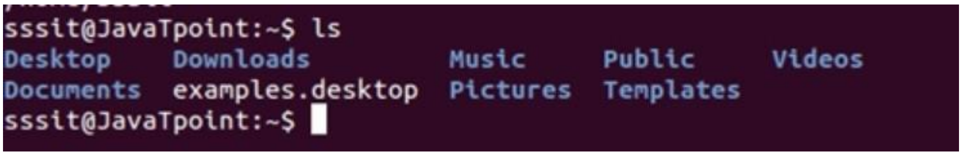
STUDY ON BASIC LINUX COMMANDS

AIM:
Run the following shell commands from the Terminal:

1. ls
2. cat
3. ps
4. cp
5. echo
6. cmp
7. pwd
8. rm
9. mv
10. touch
11. chmod
12. clear
13. man
14. more
15. less
16. grep
17. head
18. tail
19. sort
20. whoami

1. ls: Lists files and directories in the current directory.

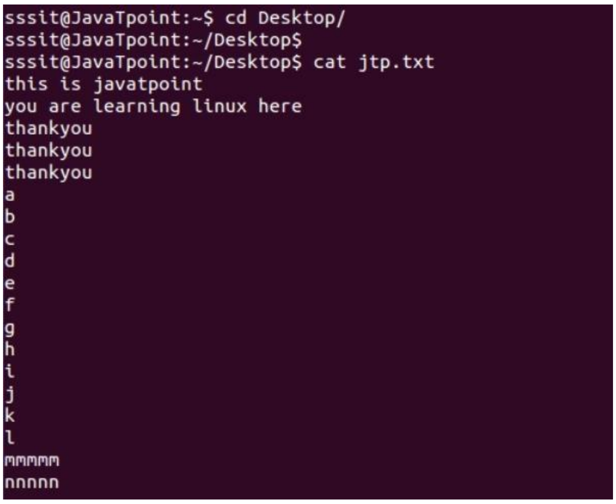
ls



```
sssit@JavaTpoint:~$ ls
Desktop    Downloads         Music     Public     Videos
Documents  examples.desktop  Pictures  Templates
sssit@JavaTpoint:~$ 
```

2. cat: Concatenates and displays the content of files.

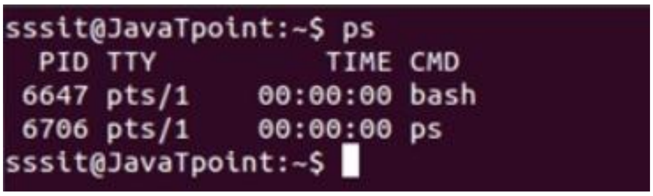
cat filename.txt



```
sssit@JavaTpoint:~$ cd Desktop/
sssit@JavaTpoint:~/Desktop$
sssit@JavaTpoint:~/Desktop$ cat jtp.txt
this is javatpoint
you are learning linux here
thankyou
thankyou
thankyou
a
b
c
d
e
f
g
h
i
j
k
l
mmmmm
nnnnn
```

3. ps: Displays a list of currently running processes.

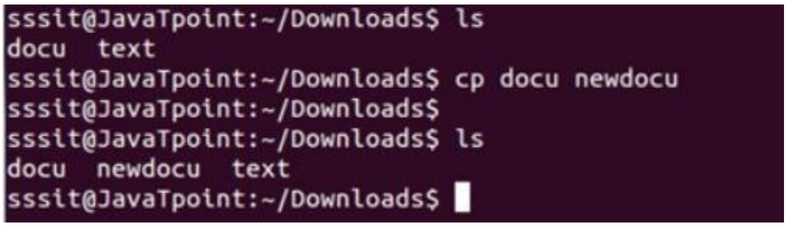
ps



```
sssit@JavaTpoint:~$ ps
  PID TTY          TIME CMD
 6647 pts/1    00:00:00 bash
 6706 pts/1    00:00:00 ps
sssit@JavaTpoint:~$ 
```

4. cp: Copies files or directories.

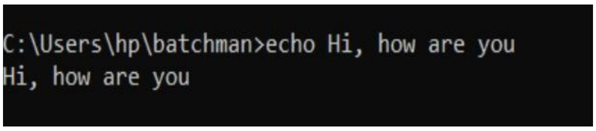
cp source.txt destination.txt



```
sssit@JavaTpoint:~/Downloads$ ls
docu  text
sssit@JavaTpoint:~/Downloads$ cp docu newdocu
sssit@JavaTpoint:~/Downloads$
sssit@JavaTpoint:~/Downloads$ ls
docu  newdocu  text
sssit@JavaTpoint:~/Downloads$ 
```

5. echo: Prints text to the terminal.

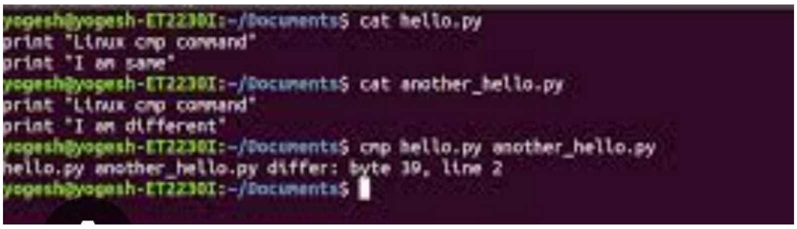
echo "Hello, World!"



```
C:\Users\hp\batchman>echo Hi, how are you
Hi, how are you
```

6. cmp: Compares two files byte by byte.

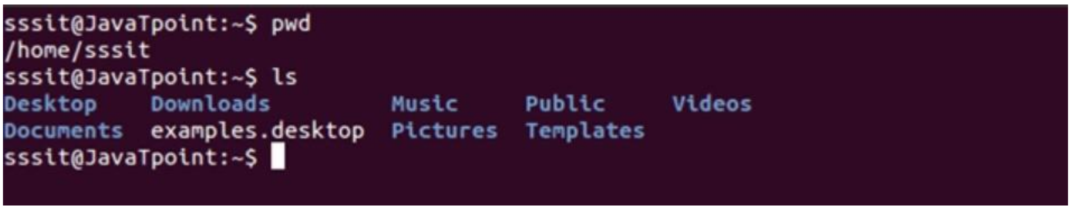
cmp file1.txt file2.txt



```
yogesh@yogesh-ET2230I:~/Documents$ cat hello.py
print "Linux cmp command"
print "I am same"
yogesh@yogesh-ET2230I:~/Documents$ cat another_hello.py
print "Linux cmp command"
print "I am different"
yogesh@yogesh-ET2230I:~/Documents$ cmp hello.py another_hello.py
hello.py another_hello.py differ: byte 39, line 2
yogesh@yogesh-ET2230I:~/Documents$ 
```

7. pwd: Displays the current working directory.

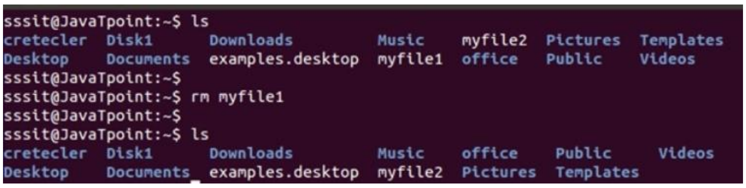
pwd



```
sssit@JavaTpoint:~$ pwd
/home/sssit
sssit@JavaTpoint:~$ ls
Desktop    Downloads          Music     Public     Videos
Documents  examples.desktop  Pictures  Templates
sssit@JavaTpoint:~$ 
```

8. rm: Removes files or directories.

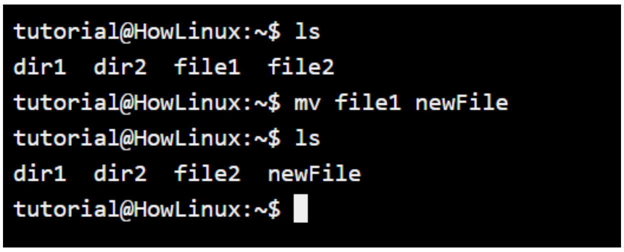
rm file.txt



```
sssit@JavaTpoint:~$ ls
cretecler  Disk1      Downloads         Music    myfile2  Pictures  Templates
Desktop    Documents  examples.desktop  myfile1  office   Public    Videos
sssit@JavaTpoint:~$
sssit@JavaTpoint:~$ rm myfile1
sssit@JavaTpoint:~$
sssit@JavaTpoint:~$ ls
cretecler  Disk1      Downloads         Music    office    Public     Videos
Desktop    Documents  examples.desktop  myfile2  Pictures  Templates
```

9. mv: Moves or renames files or directories.

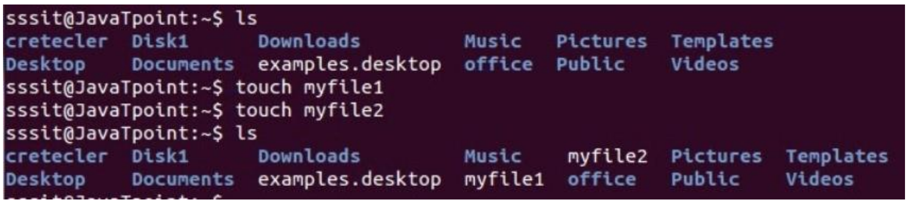
mv oldname.txt newname.txt



```
tutorial@HowLinux:~$ ls
dir1  dir2  file1  file2
tutorial@HowLinux:~$ mv file1 newFile
tutorial@HowLinux:~$ ls
dir1  dir2  file2  newFile
tutorial@HowLinux:~$
```

10. touch: Creates an empty file or updates the timestamp of an existing file.

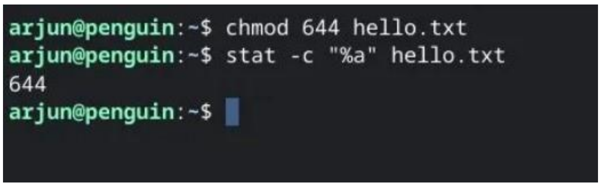
touch newfile.txt



```
sssit@JavaTpoint:~$ ls
cretecler  Disk1      Downloads         Music    Pictures  Templates
Desktop    Documents  examples.desktop  office   Public    Videos
sssit@JavaTpoint:~$ touch myfile1
sssit@JavaTpoint:~$ touch myfile2
sssit@JavaTpoint:~$ ls
cretecler  Disk1      Downloads         Music    myfile2  Pictures  Templates
Desktop    Documents  examples.desktop  myfile1  office   Public    Videos
```

11. chmod: Changes the file permissions.

chmod 755 filename.sh



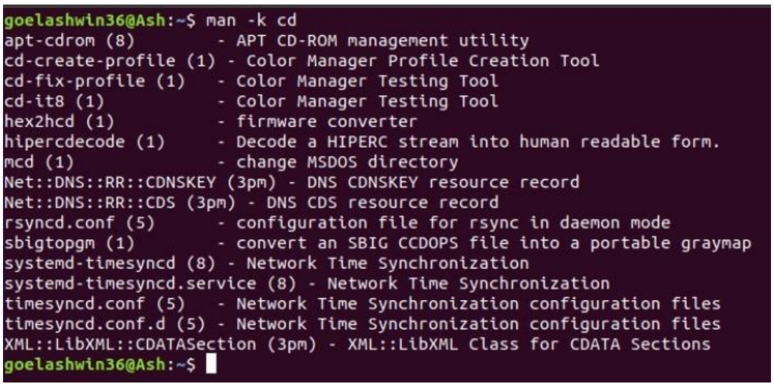
```
arjun@penguin:~$ chmod 644 hello.txt
arjun@penguin:~$ stat -c "%a" hello.txt
644
arjun@penguin:~$
```

12. clear: Clears the terminal screen.
Clear

```
lakshaygarg@ubuntu:~$
```

13. man: Displays the manual for a command.

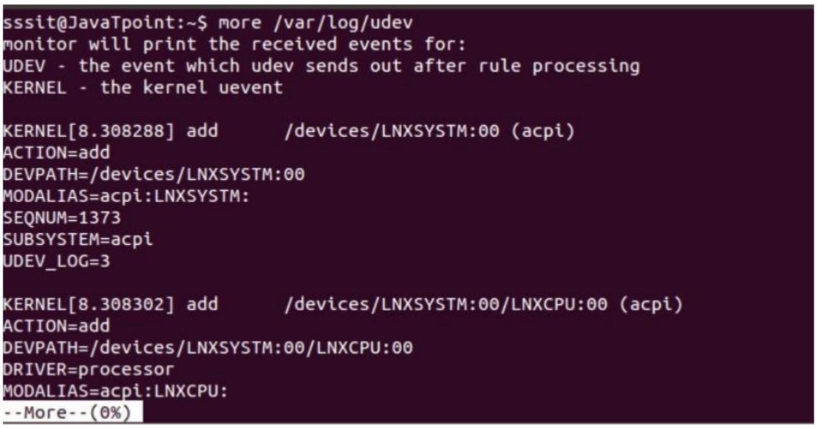
man ls



```
goelashwin36@Ash:~$ man -k cd
apt-cdrom (8)        - APT CD-ROM management utility
cd-create-profile (1) - Color Manager Profile Creation Tool
cd-fix-profile (1)   - Color Manager Testing Tool
cd-it8 (1)           - Color Manager Testing Tool
hex2hcd (1)          - firmware converter
hipercdecode (1)     - Decode a HIPERC stream into human readable form.
mcd (1)              - change MSDOS directory
Net::DNS::RR::CDNSKEY (3pm) - DNS CDNSKEY resource record
Net::DNS::RR::CDS (3pm) - DNS CDS resource record
rsyncd.conf (5)      - configuration file for rsync in daemon mode
sbigtopgm (1)        - convert an SBIG CCDOPS file into a portable graymap
systemd-timesyncd (8) - Network Time Synchronization
systemd-timesyncd.service (8) - Network Time Synchronization
timesyncd.conf (5)   - Network Time Synchronization configuration files
timesyncd.conf.d (5) - Network Time Synchronization configuration files
XML::LibXML::CDATASection (3pm) - XML::LibXML Class for CDATA Sections
goelashwin36@Ash:~$
```

14. more: Views file content page by page.

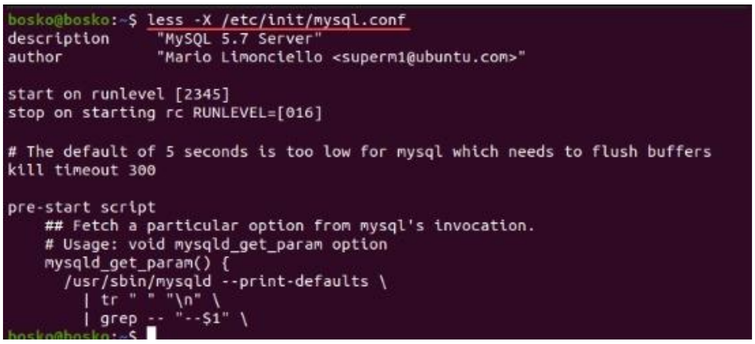
more filename.txt



```
sssit@JavaTpoint:~$ more /var/log/udev
monitor will print the received events for:
UDEV - the event which udev sends out after rule processing
KERNEL - the kernel uevent

KERNEL[8.308288] add      /devices/LNXSYSTM:00 (acpi)
ACTION=add
DEVPATH=/devices/LNXSYSTM:00
MODALIAS=acpi:LNXSYSTM:
SEQNUM=1373
SUBSYSTEM=acpi
UDEV_LOG=3

KERNEL[8.308302] add      /devices/LNXSYSTM:00/LNXCPU:00 (acpi)
ACTION=add
DEVPATH=/devices/LNXSYSTM:00/LNXCPU:00
DRIVER=processor
MODALIAS=acpi:LNXCPU:
--More--(0%)
```

15. less: Similar to more, but with more viewing options.

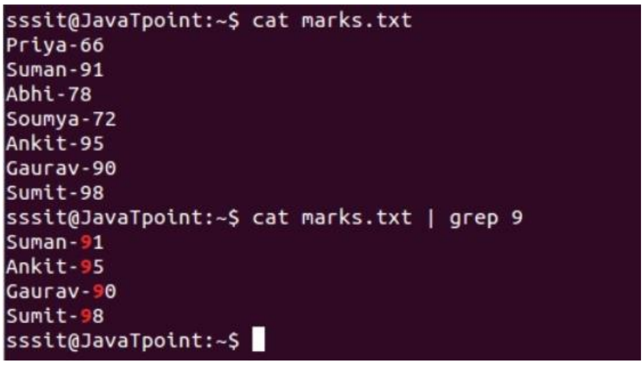
less filename.txt



```
bosko@bosko:~$ less -X /etc/init/mysql.conf
description     "MySQL 5.7 Server"
author          "Mario Limonciello <superm1@ubuntu.com>"

start on runlevel [2345]
stop on starting rc RUNLEVEL=[016]

# The default of 5 seconds is too low for mysql which needs to flush buffers
kill timeout 300

pre-start script
    ## Fetch a particular option from mysql's invocation.
    # Usage: void mysqld_get_param option
    mysqld_get_param() {
      /usr/sbin/mysqld --print-defaults \
        | tr " " "\n" \
        | grep -- "--$1" \
bosko@bosko:~$
```

16. grep: Searches for patterns within files.

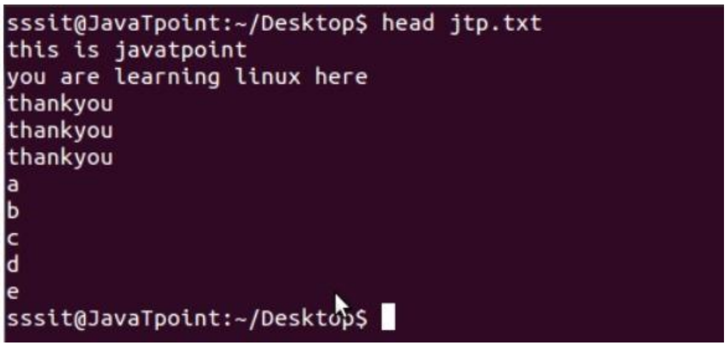
grep "search_term" filename.txt



```
sssit@JavaTpoint:~$ cat marks.txt
Priya-66
Suman-91
Abhi-78
Soumya-72
Ankit-95
Gaurav-90
Sumit-98
sssit@JavaTpoint:~$ cat marks.txt | grep 9
Suman-91
Ankit-95
Gaurav-90
Sumit-98
sssit@JavaTpoint:~$
```

17. head: Displays the first few lines of a file.

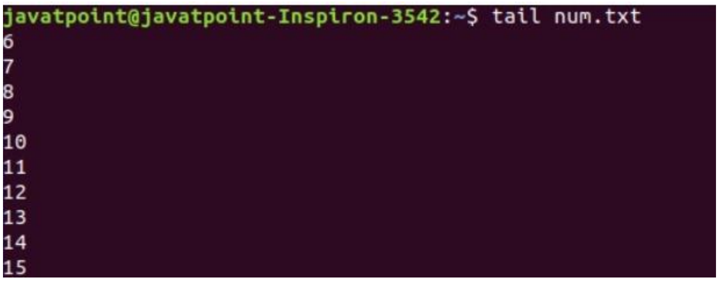
head filename.txt



```
sssit@JavaTpoint:~/Desktop$ head jtp.txt
this is javatpoint
you are learning linux here
thankyou
thankyou
thankyou
a
b
c
d
e
sssit@JavaTpoint:~/Desktop$
```

18. tail: Displays the last few lines of a file.

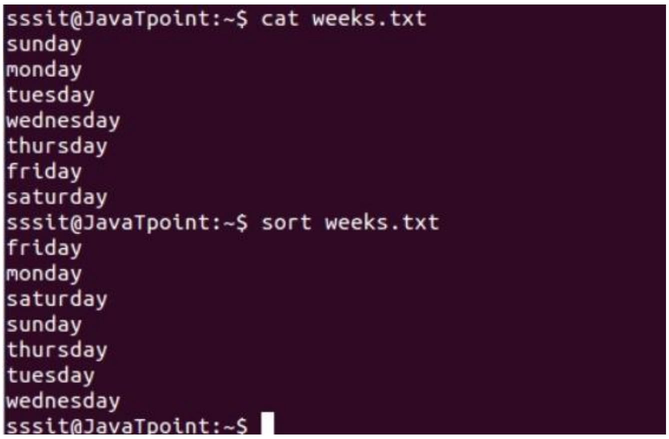
tail filename.txt



```
javatpoint@javatpoint-Inspiron-3542:~$ tail num.txt
6
7
8
9
10
11
12
13
14
15
```

19. sort: Sorts the contents of a file.

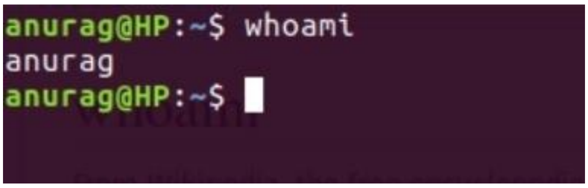
sort filename.txt



```
sssit@JavaTpoint:~$ cat weeks.txt
sunday
monday
tuesday
wednesday
thursday
friday
saturday
sssit@JavaTpoint:~$ sort weeks.txt
friday
monday
saturday
sunday
thursday
tuesday
wednesday
sssit@JavaTpoint:~$
```

20. whoami: Displays the current user's name.

Whoami



```
anurag@HP:~$ whoami
anurag
anurag@HP:~$
```

RESULT:

The following shell commands are implemented in the Terminal.

EXP NO: **REG NO:**

DATE: **NAME:**

SHELL PROGRAMMING

AIM:

To create a shell script for the following problem description and implement them.

a) Greatest of three numbers.

Aim:
To write a shell program to find the greatest of three number.

Algorithm Description:
Given any three number as input the program will return the largest among the three number as output.

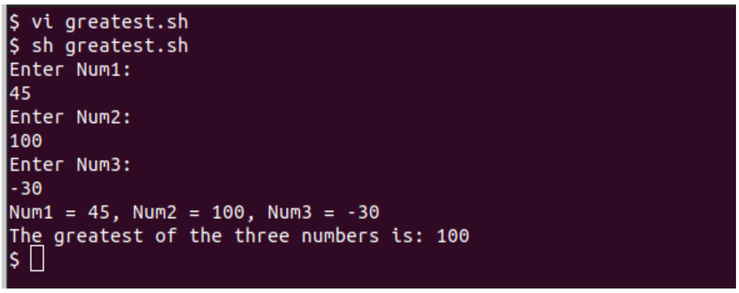
Algorithm:

```
1. Start.
2. Read three numbers (`num1`, `num2`, `num3`).
3. Compare the numbers:
  - If `num1` is the greatest, set `greatest = num1`.
  - Else if `num2` is the greatest, set `greatest = num2`.
  - Else, set `greatest = num3`.
4. Print `greatest`.
5. End.
```

Program:

```
echo "Enter
Num1:"read num1
echo "Enter
Num2:"read num2
echo "Enter
Num3:"read num3
echo "Num1 = $num1, Num2 = $num2, Num3 = $num3"

if [ $num1 -gt $num2 ] && [ $num1 -gt
$num3 ]then
  greatest=$num1
elif [ $num2 -gt $num1 ] && [ $num2 -gt
$num3 ]then
  greatest=$num
2else
  greatest=$num3
fi
echo "The greatest of the three numbers is: $greatest"
```

Test case:



```
$ vi greatest.sh
$ sh greatest.sh
Enter Num1:
45
Enter Num2:
100
Enter Num3:
-30
Num1 = 45, Num2 = 100, Num3 = -30
The greatest of the three numbers is: 100
$ 
```

b) Sum of N numbers.

Aim:
To write a shell program to find the sum of N numbers.

Algorithm Description:
Given a number N as input, the program will accept N numbers and return the sum of these numbers asoutput.

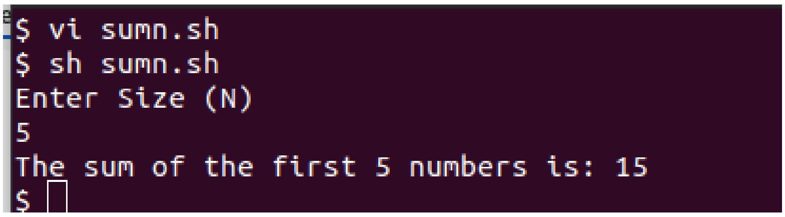
Algorithm:

```
1. Start.
2. Read the size (N) of the set of numbers.
3. Initialize i to 1 and sum to 0.
4. Display a prompt to enter the numbers.
5. While i is less than or equal to N:
       ▪ Read a number (num).
       ▪ Add num to sum.
       ▪ Increment i by 1.
6. Print the sum of the N numbers.
7. Stop.
```

Program:

```
echo "Enter Size
(N)"read N
i=1
sum=
0

echo "Enter
Numbers"while [ $i
-le $N ]
do
  read num        # Get number
  sum=$((sum + num)) # sum +=
  numi=$((i + 1))
done

echo "The sum of the $N numbers is: $sum"
```

Test case:



```
$ vi sumn.sh
$ sh sumn.sh
Enter Size (N)
5
The sum of the first 5 numbers is: 15
$ 
```

c) Factorial of number.

Aim:
To write a shell program to find the factorial of a given number.

Algorithm Description:
Given a number as input, the program calculates the factorial of that number using a loop.

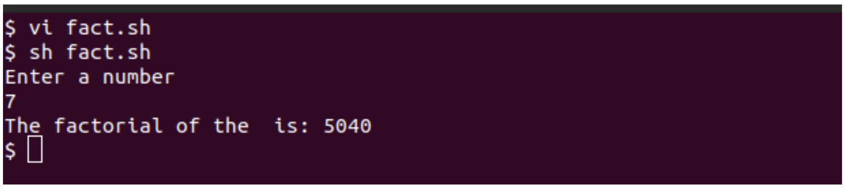
Algorithm:

```
1. Start.
2. Read the number (`num`).
3. Initialize `fact` to 1.
4. While `num` is greater than 1:
  1. Multiply `fact` by `num`.
  2. Decrement `num` by 1.
5. Print the factorial (`fact`).
6. End.
```

Program:

```
echo "Enter a
number"read num
fact=1

while [ $num -gt
1 ]do
  fact=$((fact * num)) #fact = fact *
numnum=$((num - 1)) #num = num
- 1 done

echo "The factorial of the $N is: $fact"
```

Test case:



```
$ vi fact.sh
$ sh fact.sh
Enter a number
7
The factorial of the  is: 5040
$ 
```

d) Fibonacci series of number.

Aim:

To write a shell program to generate the Fibonacci series up to N terms.

Algorithm Description:

The program reads a number N as input and prints the Fibonacci series up to N terms.

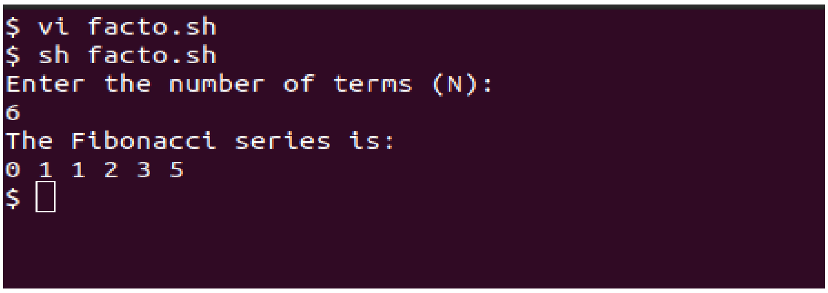
Algorithm:

```
1. Start.
2. Read `N`.
3. Set `a = 0`, `b = 1`.
4. Set `i = 0`.
5. While `i < N`:
   - Print `a`.
   - Set `fn = a + b`.
   - Set `a = b`, `b = fn`.
   - Increment `i`.
6. End.
```

Program:

```
echo "Enter the number of terms
(N):"read N

a=0
b=
1
echo "The Fibonacci series is:"

i=0
while [ $i -lt $N
]do
   echo -n "$a
   " fn=$((a  +
   b))a=$b
   b=$fn
   i=$((i     +
   1))
don
e
ech
o
```

Test case:



```
$ vi facto.sh
$ sh facto.sh
Enter the number of terms (N):
6
The Fibonacci series is:
0 1 1 2 3 5
$ 
```

e) Armstrong number.

Aim:
To write a shell program to check if a given number is an Armstrong number.

Algorithm Description:
Given a number as input, the program calculates the sum of the cubes of its digits. If this sum equalsthe original number, it is an Armstrong number.

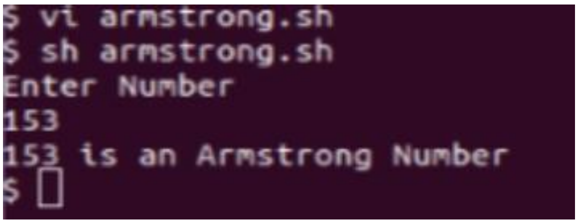
Algorithm:

```
1. Start.
2. Read `num`.
3. Initialize `sum = 0` and `item = num`.
4. While `item` is not equal to 0:
   1. Find the remainder: `rem = item % 10`.
   2. Calculate the cube: `cube = rem * rem * rem`.
   3. Add `cube` to `sum`.
   4. Update `item = item / 10`.
5. If `sum` equals `num`:
   1. Print that `num` is an Armstrong number.
6. Else:
   1. Print that `num` is not an Armstrong number.
7. End.
```

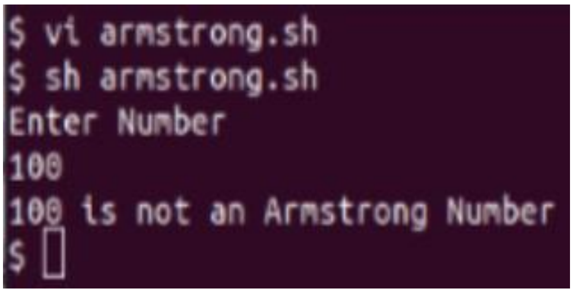
Program:

```
echo "Enter
Number"read num
sum=0
item=$nu
m
while [ $item -ne
0 ]do
  rem=$(expr $item % 10)
  cube=$(expr $rem \* $rem \*
  $rem)sum=$(expr $sum +
  $cube) item=$(expr $item / 10)
done
if [ $sum -eq
$num ]then
  echo "$num is an Armstrong
Number"else
  echo "$num is not an Armstrong
Number"fi
```

Test case:



```
$ vi armstrong.sh
$ sh armstrong.sh
Enter Number
153
153 is an Armstrong Number
$ 
```



```
$ vi armstrong.sh
$ sh armstrong.sh
Enter Number
100
100 is not an Armstrong Number
$ 
```

f) **Reverse a String.**

Aim:
To create a shell script that reverses a given string input by the user.

Algorithm Description:
The script reads a string from the user, calculates its length, and then constructs the reversed version ofthe string by iterating through it from the end to the start. Finally, it prints the reversed string.

Algorithm:

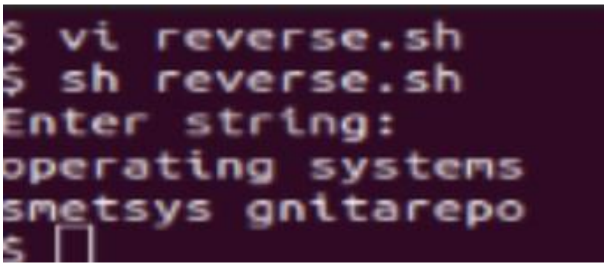
```
1. Start.
2. Read the string input from the user.
3. Initialize `len` to 0 and `temp_string` to the user input.
4. Calculate the length of the string:
   - While `temp_string` is not empty:
   - Remove the first character from `temp_string`.
    - Increment `len` by 1.
5. Initialize `reverse` as an empty string.
6. Set `i` to `len - 1`.
7. Reverse the string:
   - While `i` is greater than or equal to 0:
   - Extract the character at position `i` using `awk`.
   - Append the character to `reverse`.
   - Decrement `i` by 1.
8. Print the reversed string.
9. End.
```

Program:

```
# Reading a string via user
inputecho "Enter string:"
read string

# Getting the length of the given string using a different
methodlen=0
temp_string="$string"
while [ -n "$temp_string" ];
  do
  temp_string=${temp_string
  #?}len=$((len + 1))
done

# Looping for reversing the string
# Initialize i=len-1 for reversing the string and run
till i=0reverse=""
i=$((len - 1))
```

```
while [ $i -ge 0
]do
  # Extract single character from string
  # Using expr to get the character at position i
  char=$(echo "$string" | awk -v i="$((i + 1))" '{print substr($0,
  i, 1)}')reverse="$reverse$char"
  i=$((i -
1))done
echo "$reverse"
```

Test case:



```
$ vi reverse.sh
$ sh reverse.sh
Enter string:
operating systems
smetsys gnitarepo
$ 
```

g) Replacing a character in a

stringAim:

To create a shell script that replaces a character in a given string input by the user.

Algorithm Description:

The script reads a string and two characters from the user: one to be replaced, and another to replace itwith. It iterates over the string, replacing all occurrences of the target character with the new character,and finally prints the modified string.

Algorithm:

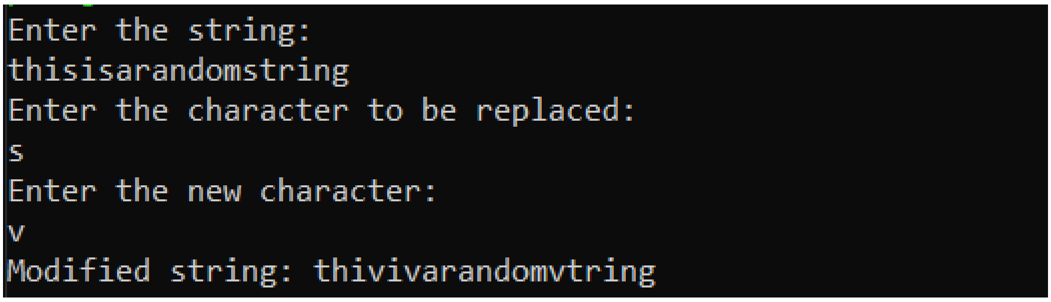
1. Start.
2. Read the string input from the user.
3. Read the character to be replaced from the user.
4. Read the new character (replacement) from the user.
5. Initialize result to an empty string.
6. Iterate through each character of the input string:
    - If the current character matches the target character:
        - Append the new character to result.
    - Otherwise:
        - Append the current character to result.
7. Print the modified string.
8. End.

Program

```
#Step 1: Use string for storing the
string# Step 2: Reading a string via
user inputecho "Enter the string:"
read string

# Step 3: Read the character to be
replacedecho "Enter the character to
be replaced:" read old_char
```

```
# Step 4: Read the new replacement
characterecho "Enter the new character:"
read new_char

# Step 5: Initialize an empty result
stringresult=""

# Step 6: Loop through each character of the
stringfor (( i=0; i<${#string}; i++ )); do
  # Extract the current
  character
  current_char="${string:$i:
  1}"

  # Check if current character is the one to be
  replacedif [ "$current_char" = "$old_char" ];
  then
    result="$result$new_char" # Replace with new
  characterelse
    result="$result$current_char" # Append original
  characterfi
done

# Step 7: Print the modified
stringecho "Modified string:
$result"
```

Test Case:



```
Enter the string:
thisisarandomstring
Enter the character to be replaced:
s
Enter the new character:
v
Modified string: thivivarandomvtring
```

h) Counting number of

vowelsAim:

To create a shell script that counts the number of vowels in a given string input by the user.

Algorithm Description:

The script reads a string from the user, iterates through each character, and checks if it is a vowel (a, e,i, o, u, both lowercase and uppercase). It maintains a count of vowels and prints the final count at the end.

Algorithm:

1. Start.
2. Read the string input from the user.
3. Initialize vowel_count to 0.
4. Iterate through each character of the input string:

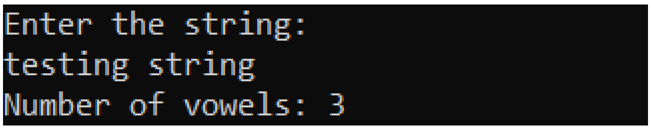
- If the character is a vowel (either lowercase or uppercase):
  Increment vowel_count by 1.

5. Print the vowel count.
6. End.

Program:

```
# Step 2: Reading a string via user
inputecho "Enter the string:"
read string

# Step 3: Initialize vowel_count
to 0vowel_count=0

# Step 4: Initialize index and loop through each
characteri=0
len=${#string}

while [ $i -lt $len ]; do
  # Extract the current character
  current_char=$(echo "$string" | cut -c
  $((i+1)))

  # Step 4: Check if the character is a vowel (lowercase or
  uppercase)case "$current_char" in
    [aAeEiIoOuU]) # If it's a vowel, increment the count
      vowel_count=$((vowel_count + 1))
      ;;
  esac

  # Increment the
  indexi=$((i + 1))
done

# Step 5: Print the vowel count
echo "Number of vowels: $vowel_count"
```

Test Case:



```
Enter the string:
testing string
Number of vowels: 3
```

I) Odd Even

Aim:
To create a shell script that checks if a given number is odd or even.

Algorithm Description:
Read a string from the user and initialize a vowel counter. Loop through each character, checking if it'sa vowel (a, e, i, o, u in any case). Count each vowel found and print the final count.

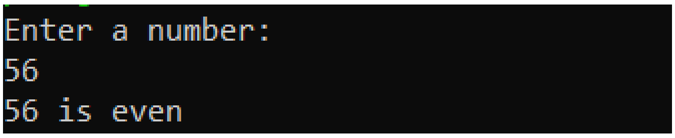
Algorithm:

1. Start.
2. Input the number from the user.
3. Check if the number is divisible by 2 using the modulus operator:
    - If number % 2 == 0, the number is even.
    - Otherwise, the number is odd.
4. Print the result (odd or even).
5. End.

Program:

```
# Step 2: Reading the number from the
userecho "Enter a number:"
read number

# Step 3: Check if the number is divisible
by 2if [ $((number % 2)) -eq 0 ]; then
  echo "$number is
even"else
  echo "$number is odd"
```

Test Case:



```
Enter a number:
56
56 is even
```

J) Given string is a palindrome or

notAim:

To create a shell script that checks if a given number is a palindrome.

Algorithm Description:

Read a number from the user and initialize variables for the original number and its reverse. Reverse the number by extracting and appending each digit to a new reversed value. Compare the original andreversed numbers, then print if the number is a palindrome.

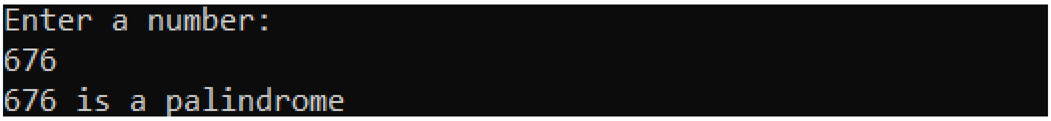
Algorithm:

1. Start.
2. Input the number from the user.
3. Initialize orginal number to store the input number and reverse to 0.
4. Reverse the digits of the number:
    - While the number is not 0:
        - Extract the last digit using number % 10.
        - Update reverse = reverse * 10 + last_digit.
        - Remove the last digit from number by number = number / 10.
5. Compare the reversed number with the original number.
6. Print if the number is a palindrome or not.
7. End.

Program:

```
# Step 2: Reading the number from the
userecho "Enter a number:"
read number

# Step 3: Initialize original_number and
reverseoriginal_number=$number
reverse=0
# Step 4: Reverse the digits of the
numberwhile [ $number -gt 0 ]
do
  # Extract the last digit
  last_digit=$((number % 10))

  # Build the reversed number
  reverse=$((reverse * 10 +
  last_digit))

  # Remove the last digit from the
  numbernumber=$((number / 10))
done

# Step 5: Compare the original number and the reversed
numberif [ $original_number -eq $reverse ]; then
  echo "$original_number is a
palindrome"else
  echo "$original_number is not a
palindrome"fi
```

Test Case:



```
Enter a number:
676
676 is a palindrome
```

RESULT:

Therefore the shell script for the following problem description is implemented.

EXP NO: **REG NO:**

DATE: **NAME:**

SYSTEM CALLS PROGRAMMING - I

Aim:

The aim of this program is to create a parent-child relationship between processes in C

Problem description:

Create a child process using fork() system call.

Use wait system call to wait for parent.

- fork()
- getpid()
- getppid()
- wait()
    - Header file: #include <sys/types.h>, #include <sys/wait.h>

fork():

This function creates a new process by duplicating the calling process. The newly created process is called the child process, and the original process is the parent. fork() returns the process ID (PID) of the child process to the parent and returns 0 to the child process.

getpid():

This function returns the process ID (PID) of the calling process. Each running process has a unique PID, which can be used to track and manage processes in the operating system.

getppid():

This function returns the parent process ID (PPID) of the calling process. The parent process is the one that created the current process using fork().

wait():

This function makes the parent process wait until all of its child processes have finished executing. It returns the PID of the child that terminated and allows the parent to retrieve the child's exit status.

void factorial():

This function calculates the factorial of a given integer. It prompts the user for a number, then computes the factorial by multiplying the number by every integer less than it down to 1. Factorial is used in various mathematical and algorithmic problems, particularly in combinatorics.

void evenodd():

This function checks whether a given number is even or odd. It asks the user to input a number and based on whether the number is divisible by 2 (using the modulus operator), it prints whether the number is even or odd.

void addreal():

This function adds two real (floating-point) numbers. It prompts the user to input two numbers, then calculates and prints the sum of the two values.

void revnum():

This function reverses the digits of a given integer. It reads a number from the user, then uses a loop to reverse the order of its digits by repeatedly taking the remainder of division by 10 and constructing the reversed number.

void armstrong()

This function checks if a number is an Armstrong number. An Armstrong number is one that is equal to the sum of its own digits each raised to the power of the number of digits. The function calculates this sum and compares it with the original number to determine if it's an Armstrong number.

void fibonacci():

This function generates and prints the Fibonacci series up to a specified number of terms. The Fibonacci series is a sequence where each term is the sum of the two preceding ones, starting from 0 and 1.

void sortnumbers():

This function sorts an array of n numbers in ascending order. It first prompts the user to input the number of elements and the elements themselves, and then uses a simple sorting algorithm (like bubble sort) to arrange the numbers.

void palindrome():

This function checks if a given string is a palindrome. It takes a string input from the user, then compares it with its reverse to determine if it reads the same forward and backward. It prints the result accordingly.

void printFileLineCount():

This function counts and prints the number of lines in a given file. The user provides the filename, and the function reads the file, counting newline characters (\n) to determine the total number of lines.

void convertCase():

This function converts the case of each character in a string from uppercase to lowercase and vice versa. It reads a string from the user, and for each character, it checks if it’s lowercase or uppercase, converting it to the opposite case.

Program code:

```
#include <stdio.h>

#include <string.h>

#include <ctype.h>

#include <stdlib.h>
```

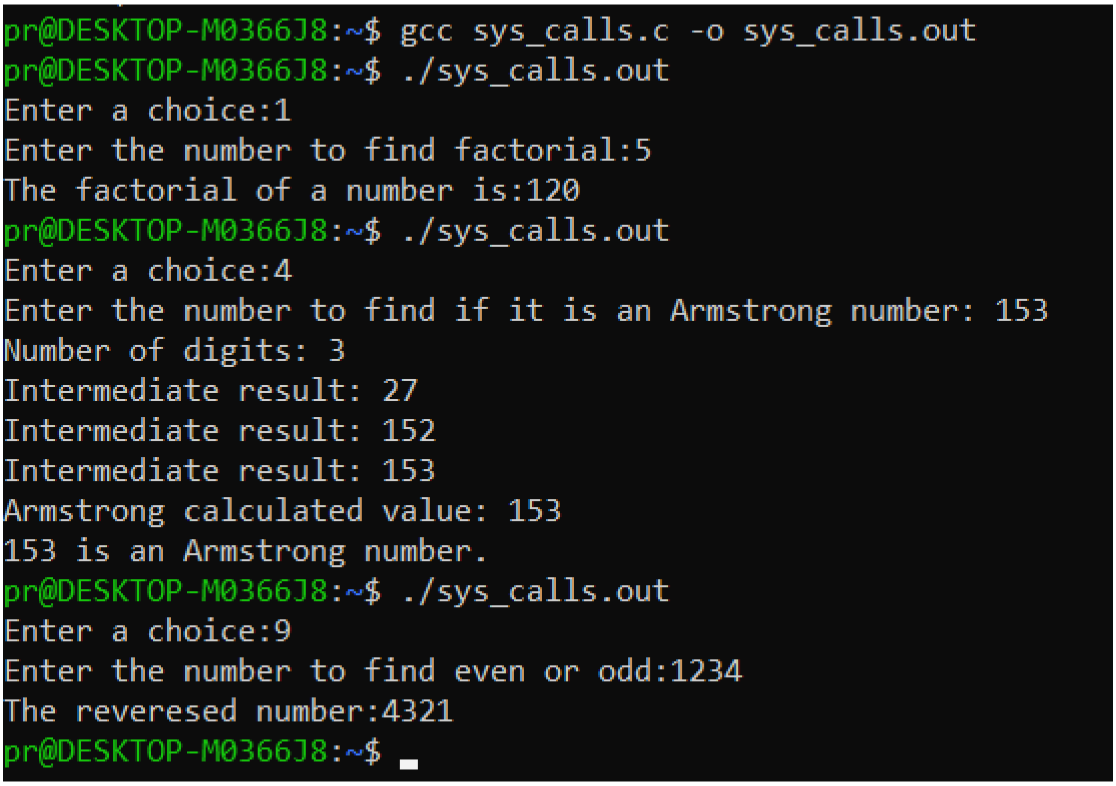
```
#include <unistd.h>
#include <sys/wait.h>

void factorial();
void evenodd();
void addreal();
void revnum();
void armstrong();
void fibonacci();
void sortnumbers();
void palindrome();
void printFileLineCount();
void convertCase();

int main() {
    int ch;
    printf("Enter a choice:\n");
    printf("1: Factorial\n");
    printf("2: Fibonacci Series\n");
    printf("3: Sorting Numbers\n");
    printf("4: Armstrong Number\n");
    printf("5: Palindrome\n");
    printf("6: Even or Odd\n");
    printf("7: Print File Line Count\n");
    printf("8: Addition of Two Real Numbers\n");
    printf("9: Reverse Number\n");
    printf("10: Convert Case (Upper <-> Lower)\n");
    scanf("%d", &ch);

    pid = fork();
            if (pid < 0) {
                perror("fork failed");
                exit(EXIT_FAILURE);
            }
            else if (pid == 0){
```

```
switch (ch) {
    case 1:
        factorial();
        break;
    case 2:
        fibonacci();
        break;
    case 3:
        sortnumbers();
        break;
    case 4:
        armstrong();
        break;
    case 5:
        palindrome();
        break;
    case 6:
        evenodd();
        break;
    case 7:
        printFileLineCount();
        break;
    case 8:
        addreal();
        break;
    case 9:
        revnum();
        break;
    case 10:
        convertCase();
        break;
    default:
        printf("Invalid choice!\n");
}
}
```

```
    else {
            waitpid(pid, &status, 0);
    }
}

// Factorial of a number
void factorial() {
    int result = 1, num;
    printf("Enter the number to find factorial: ");
    scanf("%d", &num);
    while (num > 0) {
        result = result * num;
        num--;
    }
    printf("Factorial: %d\n", result);
}

// Fibonacci series
void fibonacci() {
    int n, t1 = 0, t2 = 1, nextTerm;
    printf("Enter the number of terms: ");
    scanf("%d", &n);
    printf("Fibonacci Series: ");
    for (int i = 1; i <= n; ++i) {
        printf("%d, ", t1);
        nextTerm = t1 + t2;
        t1 = t2;
        t2 = nextTerm;
    }
    printf("\n");
}

// Sorting of n numbers
void sortnumbers() {
    int n;
```

```c
    printf("Enter number of elements: ");
    scanf("%d", &n);
    int arr[n];
    printf("Enter the numbers:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (arr[i] > arr[j]) {
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
    }

    printf("Sorted numbers: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}

// Check for palindrome
void palindrome() {
    char str[100];
    printf("Enter a string: ");
    scanf("%s", str);
    int len = strlen(str);
    int flag = 1;

    for (int i = 0; i < len / 2; i++) {
        if (str[i] != str[len - i - 1]) {
            flag = 0;
            break;
        }
    }

    if (flag) {
        printf("%s is a palindrome.\n", str);
    } else {
        printf("%s is not a palindrome.\n", str);
    }
}

// Armstrong number
void armstrong() {
    int num;
    printf("Enter the number to find if it is an Armstrong number: ");
    scanf("%d", &num);

    int numc = num, val = 0, res = 0;

    while (num > 0) {
        val++;
        num = num / 10;
    }

    num = numc;

    while (num > 0) {
        int m = num % 10, v = 1;
        for (int i = 0; i < val; i++) {
            v = v * m;
        }
        res += v;
        num = num / 10;
    }
    if (res == numc) {
        printf("%d is an Armstrong number.\n", numc);
    } else {
        printf("%d is not an Armstrong number.\n", numc);
    }
}

// Even or odd number
void evenodd() {
    int num;
    printf("Enter the number to find even or odd: ");
    scanf("%d", &num);
    if (num % 2 == 0) {
        printf("Number is even\n");
    } else {
        printf("Number is odd\n");
    }
}

// Print file with line count
void printFileLineCount() {
    char filename[100];
    printf("Enter the filename: ");
    scanf("%s", filename);

    FILE *file = fopen(filename, "r");
    if (file == NULL) {
        printf("File not found!\n");
        return;
    }

    int lineCount = 0;
    char ch;
    while ((ch = fgetc(file)) != EOF) {
        if (ch == '\n') {
            lineCount++;
        }
    }
    fclose(file);
    printf("Total number of lines: %d\n", lineCount);
}

// Addition of two real numbers
void addreal() {
    double num1, num2;
    printf("Enter the two numbers: ");
    scanf("%lf %lf", &num1, &num2);
    double result = num1 + num2;
    printf("The result is: %lf\n", result);
}

// Reverse a number
void revnum() {
    int num, rev = 0;
    printf("Enter the number: ");
    scanf("%d", &num);

    while (num > 0) {
        int mod = num % 10;
        rev = rev * 10 + mod;
        num = num / 10;
    }
    printf("The reversed number is: %d\n", rev);
}

// Convert string from upper to lower and vice versa
void convertCase() {
    char str[100];
    printf("Enter a string: ");
    scanf("%s", str);

    for (int i = 0; str[i] != '\0'; i++) {
        if (islower(str[i])) {
            str[i] = toupper(str[i]);
        } else if (isupper(str[i])) {
            str[i] = tolower(str[i]);
        }
    }

    printf("Converted string: %s\n", str);
```

Test Case:



```
pr@DESKTOP-M0366J8:~$ gcc sys_calls.c -o sys_calls.out
pr@DESKTOP-M0366J8:~$ ./sys_calls.out
Enter a choice:1
Enter the number to find factorial:5
The factorial of a number is:120
pr@DESKTOP-M0366J8:~$ ./sys_calls.out
Enter a choice:4
Enter the number to find if it is an Armstrong number: 153
Number of digits: 3
Intermediate result: 27
Intermediate result: 152
Intermediate result: 153
Armstrong calculated value: 153
153 is an Armstrong number.
pr@DESKTOP-M0366J8:~$ ./sys_calls.out
Enter a choice:9
Enter the number to find even or odd:1234
The reveresed number:4321
pr@DESKTOP-M0366J8:~$
```

Result:

Thus, the program has been executed successfully.

EXP NO: **REG NO:**

DATE: **NAME:**

SYSTEM CALLS PROGRAMMING - II

AIM:

To execute the given system calls using C.

PROBLEM DESCRIPTION:

The following system calls are used for process creation, termination, file handling, and interaction with the operating system. Each call serves a distinct purpose and can be used to manageprocesses, manipulate file descriptors, and interact with the system. The objective is to write a C program to demonstrate the functionality of each system call listed:

a) fork() b) exit() c) getpid() d) getppid() e) sleep()

f) setpriority() g) wait() h) open() i) read() j) write()

k) close() l) chmod()

SYSTEM CALLS DESCRIPTION:

1) fork()

- Header Files: <unistd.h>
- Function: Creates a child process.
- Synopsis: pid_t fork(void);
- Arguments: No arguments required.
- Return Value: Returns the process ID of the child process to the parent and 0 to the child process, or -1 on failure.

2) exit()

- Header Files: <stdlib.h>
- Function: Terminates the process.
- Synopsis: void exit(int status);
- Arguments: status - The exit status of the process.
- Return Value: Does not return.

3) getpid()

- Header Files: <unistd.h>
- Function: Gets the process ID of the current process.
- Synopsis: pid_t getpid(void);
- Arguments: No arguments required.
- Return Value: Returns the process ID of the calling process.

4) getppid()

- Header Files: <unistd.h>
- Function: Gets the process ID of the parent process.
- Synopsis: pid_t getppid(void);
- Arguments: No arguments required.
- Return Value: Returns the parent process ID.

5) sleep()

- Header Files: <unistd.h>
- Function: Suspends execution for a specific time interval.
- Synopsis: unsigned int sleep(unsigned int seconds);
- Arguments: seconds - Number of seconds to suspend execution.
- Return Value: Returns 0 or the number of seconds left if interrupted.

6) setpriority()

- Header Files: <sys/resource.h>
- Function: Sets the scheduling priority of a process.
- Synopsis: int setpriority(int which, int who, int prio);
- Arguments:
  - which - Specifies the kind of process.
  - who - Process ID; 0 for the current process.
  - prio - Priority value.
- Return Value: Returns 0 on success and -1 on failure.

7) wait()

- Header Files: <sys/wait.h>
- Function: Makes the parent process wait for the child process to terminate.
- Synopsis: pid_t wait(int *wstatus);
- Arguments: wstatus - Pointer to store the exit status of the child process.
- Return Value: Returns the process ID of the terminated child or -1 on error.

8) open()

- Header Files: <fcntl.h>, <sys/stat.h>, <sys/types.h>
- Function: Opens a file descriptor.
- Synopsis: int open(const char *pathname, int flags);
- Arguments:
  - pathname - Specifies the file to open.
  - flags - Control how the file is opened.
- Return Value: Returns a file descriptor on success and -1 on failure.

9) read()

- Header Files: <unistd.h>
- Function: Reads data from a file descriptor.
- Synopsis: ssize_t read(int fd, void *buf, size_t count);
- Arguments:
  - fd - The file descriptor.
  - buf - The buffer to store data.
  - count - The number of bytes to read.

- Return Value: Returns the number of bytes read or -1 on failure.

10) write()

- Header Files: <unistd.h>
- Function: Writes data to a file descriptor.
- Synopsis: ssize_t write(int fd, const void *buf, size_tcount);
- Arguments:
  - fd - The file descriptor.
  - buf - The data to write.
  - count - The number of bytes to write.
- Return Value: Returns the number of bytes written or -1 on failure.

11) close()

- Header Files: <unistd.h>
- Function: Closes a file descriptor.
- Synopsis: int close(int fd);
- Arguments: fd - The file descriptor to close.
- Return Value: Returns 0 on success and -1 on failure.

12) chmod()

- Header Files: <sys/stat.h>
- Function: Changes the file permissions of a file.
- Synopsis: int chmod(const char *pathname, mode_t mode);
- Arguments:
  - pathname - The file whose permissions will be changed.
  - mode - Specifies the new permissions.
- Return Value: Returns 0 on success and -1 on failure.

PROGRAM CODE:

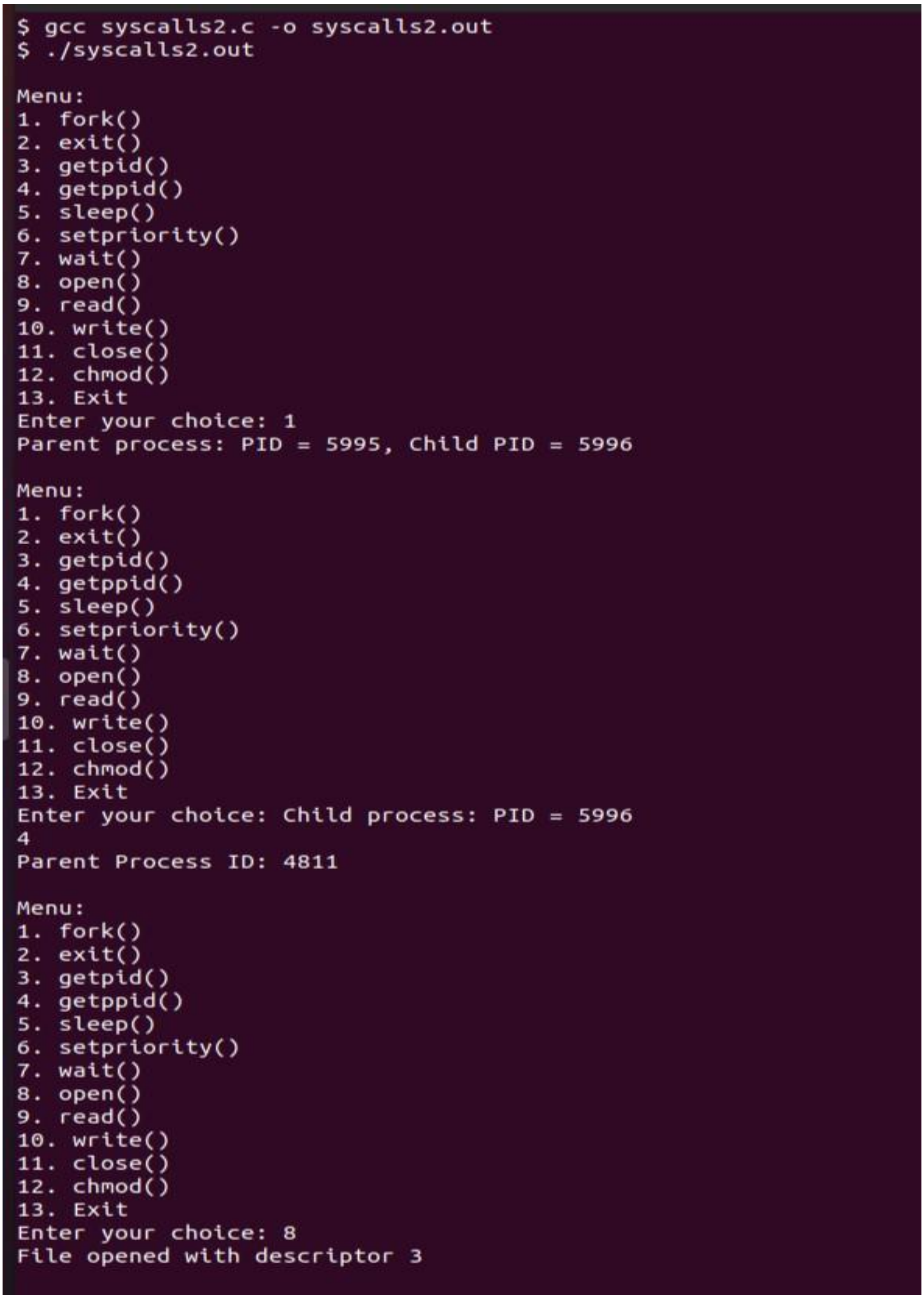
```
#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <fcntl.h>

#include <sys/stat.h>

#include <errno.h>

#include <sys/time.h>

#include <sys/resource.h>

#include <string.h>


int main() {

   int number;
```

```
char buffer[100]; // Buffer for reading from files
int fileDescriptor = -1; // Initialize to -1 to indicate no file is open
char *filename = "testfile.txt";

while (1) { // Loop indefinitely until user chooses to exit
    printf("\nMenu:\n");
    printf("1. fork() \n");
    printf("2. exit() \n");
    printf("3. getpid() \n");
    printf("4. getppid() \n");
    printf("5. sleep() \n");
    printf("6. setpriority() \n");
    printf("7. wait() \n");
    printf("8. open() \n");
    printf("9. read() \n");
    printf("10. write() \n");
    printf("11. close() \n");
    printf("12. chmod() \n");
    printf("13. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &number);

    switch(number) {
        case 1: {
            pid_t pid = fork();
            if (pid == -1) {
                perror("fork");
            } else if (pid == 0) {
                // Child process
                printf("Child process: PID = %d\n", getpid());
                exit(0); // Ensure child process exits here
            } else {
                // Parent process
                printf("Parent process: PID = %d, Child PID = %d\n", getpid(), pid);
            }
    break;
}
case 2:
 printf("Exiting program \n");
    exit(0); // Exit the program
    break;
case 3:
    printf("Process ID: %d\n", getpid());
    break;
case 4:
    printf("Parent Process ID: %d\n", getppid());
    break;
case 5:
    sleep(5);
    printf("Woke up after 5 seconds\n");
    break;
case 6: {
    int priority;
    printf("Enter priority value (lower value means higher priority): ");
    scanf("%d", &priority);
    if (setpriority(PRIO_PROCESS, 0, priority) == -1) {
        perror("setpriority");
    } else {
        printf("Priority set to %d\n", priority);
    }
    break;
}
case 7: {
    int status;
    wait(&status);
    if (WIFEXITED(status)) {
        printf("Child exited with status %d\n", WEXITSTATUS(status));
    }
    break;
}
```

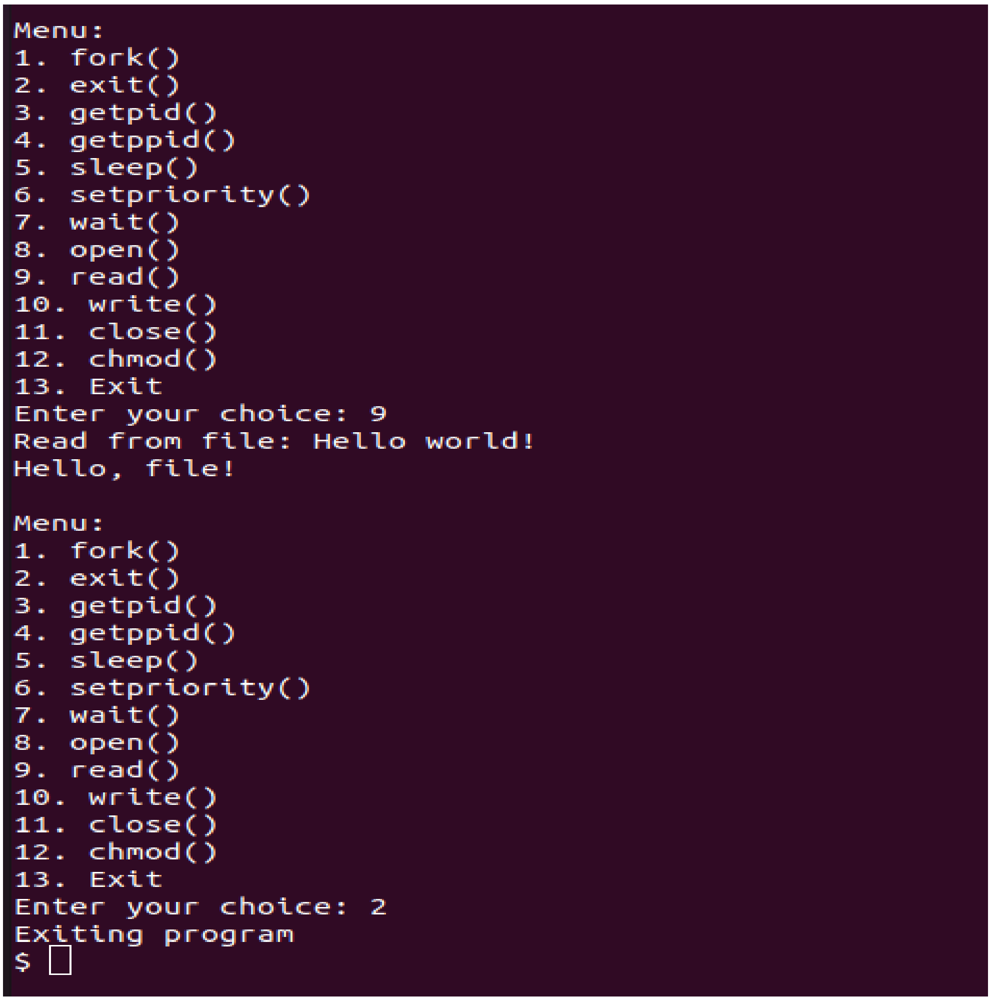
```
case 8:
    fileDescriptor = open(filename, O_RDWR | O_CREAT, S_IRUSR | S_IWUSR);
    if (fileDescriptor == -1) {
        perror("open");
    } else {
        printf("File opened with descriptor %d\n", fileDescriptor);
    }
    break;

case 9:
    if (fileDescriptor != -1) {
        ssize_t bytesRead = read(fileDescriptor, buffer, sizeof(buffer) - 1);
        if (bytesRead >= 0) {
            buffer[bytesRead] = '\0'; // Null-terminate the buffer
            printf("Read from file: %s\n", buffer);
        } else {
            perror("read");
        }
    } else {
        printf("File not opened\n");
    }
    break;
case 10:
    if (fileDescriptor != -1) {
        const char *text = "Hello, file!";
        ssize_t bytesWritten = write(fileDescriptor, text, strlen(text));
        if (bytesWritten == -1) {
            perror("write");
        } else {
            printf("Wrote %zd bytes to file\n", bytesWritten);
        }
    } else {
        printf("File not opened\n");}
    break;
case 11:
```

```
                if (fileDescriptor != -1) {
                    close(fileDescriptor);
                    printf("File closed\n");
                } else {
                    printf("File not opened\n");
                }
                break;
            case 12:
                if (chmod(filename, S_IRUSR | S_IWUSR | S_IXUSR) == -1) {
                    perror("chmod");
                } else {
                    printf("File mode changed\n");}
                break;
            case 13:
                printf("Exiting...\n");
                return 0; // Exit the program
            default:
                printf("Invalid choice\n");
        }
    }
    return 0;
}
```

SAMPLE INPUT & OUTPUT:



```
$ gcc syscalls2.c -o syscalls2.out
$ ./syscalls2.out

Menu:
1. fork()
2. exit()
3. getpid()
4. getppid()
5. sleep()
6. setpriority()
7. wait()
8. open()
9. read()
10. write()
11. close()
12. chmod()
13. Exit
Enter your choice: 1
Parent process: PID = 5995, Child PID = 5996

Menu:
1. fork()
2. exit()
3. getpid()
4. getppid()
5. sleep()
6. setpriority()
7. wait()
8. open()
9. read()
10. write()
11. close()
12. chmod()
13. Exit
Enter your choice: Child process: PID = 5996
4
Parent Process ID: 4811

Menu:
1. fork()
2. exit()
3. getpid()
4. getppid()
5. sleep()
6. setpriority()
7. wait()
8. open()
9. read()
10. write()
11. close()
12. chmod()
13. Exit
Enter your choice: 8
File opened with descriptor 3
```



```
Menu:
1. fork()
2. exit()
3. getpid()
4. getppid()
5. sleep()
6. setpriority()
7. wait()
8. open()
9. read()
10. write()
11. close()
12. chmod()
13. Exit
Enter your choice: 9
Read from file: Hello world!
Hello, file!

Menu:
1. fork()
2. exit()
3. getpid()
4. getppid()
5. sleep()
6. setpriority()
7. wait()
8. open()
9. read()
10. write()
11. close()
12. chmod()
13. Exit
Enter your choice: 2
Exiting program
$ 
```

RESULT:

Therefore, the given system calls have been programmed, implemented and executed using C.

EXP NO: **REG NO:**

DATE: **NAME:**

SIMULATION OF LINUX COMMANDS

AIM:

To execute the following Linux Commands in C

1.ls

2.cat

3.cp

4.grep

5.stat

6.ps

PROBLEM DESCRIPTION:

The task is to simulate the execution of various Linux commands using the exec family of functions ina C program. The program will allow the user to choose from a set of commands (cp, grep, stat, and ps), and then execute the chosen command by replacing the current process image with a new process image corresponding to the command.

SYSTEM CALLS USED:

exec() and execlp():

Header File:#include<unistd.h>

Function:replaces the current process image with the new process image**.**

Synopsis: #include <unistd.h>

int execl(const char *path,const char *arg)

int execlpconst(const char *file,const char *arg)

Note: a null terminated string should be the last arguments for both execl() and execlp()

Arguments: * path in execl() refers to path of the file corresponding to new process image

* file in execlp() refers to the name of the file to be executed.

* arg refers to the argument as new process

Return Value: Returns -1 on successful.

PROGRAM CODE:

```
 #include <stdio.h>
#include <unistd.h>

int main() {
    int choice;
    char str1[50], str2[50], str3[50];

    printf("\n Enter your choice:\n");
    printf("\n1.ls\n2.cat\n3.cp\n4.grep\n5.stat\n6.ps");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            printf("\n Enter your option:\n");
            scanf("%s", str1);
            execl("/bin/ls", "ls", str1, NULL);
            break;

        case 2:
            printf("Enter name of file 1:\n");
            scanf("%s", str1);

            printf("Enter name of file 2:\n");
            scanf("%s", str2);

            printf("Enter name of file 3:\n");
            scanf("%s", str3);

            execl("/bin/cat", "cat", str1, str2, str3, NULL);
            break;
        case 3:  // cp command
            printf("\nEnter the source filename:\n");
            scanf("%s", str1);
            printf("\nEnter the destination filename:\n");
            scanf("%s", str2);
            execl("/bin/cp", "cp", str1, str2, NULL);  // Using cp to copy files
            break;

        case 4:  // grep command
            printf("\nEnter the pattern to search:\n");
            scanf("%s", str1);
            printf("\nEnter the filename to search in:\n");
            scanf("%s", str2);
            execlp("grep", "grep", str1, str2, NULL); // Using grep to search for a pattern
            break;

        case 5:  // stat command
            printf("\nEnter the filename to get statistics:\n");
            scanf("%s", str1);
            execlp("stat", "stat", str1, NULL);  // Using stat to display file information
            break;

        case 6:  // ps command
            execlp("ps", "ps", NULL); // Using ps to list processes
            break;


        default:
            printf("Invalid choice\n");
            break;
    }

    return 0;
```

```
}
```

SAMPLE INPUT AND OUTPUT:

```
 Enter your choice:
 1. ls
2. cat
3. cp
4. grep
 5. stat
6. ps


Enter your option:
Desktop Documents Downloads Pictures Public Templates Videos

Enter your choice:
1. ls
2. cat
 3. cp
4. grep
5. stat
6. ps
2

Enter name of file 1:
file1.txt

Enter name of file 2:
 file2.txt

Enter name of file 3:
(Leave empty)
```

```
This is content of file1.txt

This is content of file2.txt


Enter your choice:

1. ls

2. cat

3. cp
4. grep

5. stat

6. ps

3


Enter the source filename: source.txt

Enter the destination filename: destination.txt


Enter your choice:

1. ls

2. cat

3. cp

4. grep

5. stat

6. ps

4


Enter the pattern to search:

error

Enter the filename to search in:

 log.txt


(Output depends on the content of log.txt)


Lines containing "error" in log.txt:
```

```
This is an error message
Another error occurred

 Enter your choice:
1. ls
2. cat
3. cp
4. grep
 5. stat
6. ps
5

Enter the filename to get statistics:
file.txt
```

(Output depends on the file system information)

```
File: file.txt
Size: 1024 Blocks: 2 IO Block: 4096 directory
Device: fd01h/6600d Inode: 1234567 Links: 1
Access: (0644/-rw-r--r--) Uid: ( 1000/user) Gid: ( 1000/user)
Access time: 2024-09-17 14:30:00.000000000 +0530
Modification time: 2024-09-17 14:30:00.000000000 +0530
Change time: 2024-09-17 14:30:00.000000000 +0530
```

RESULT:

Therefore, the given linux commands have been programmed, implemented and executed using C.