Ex.NO:1 (A) **STUDY OF UNIX OS**
DATE:

INTRODUCTION

An operating system is software that acts as an interface between the user and the computer hardware. It is considered as the brain of the computer. It controls and co-ordinates the internal activities of the computer and provides user interface.

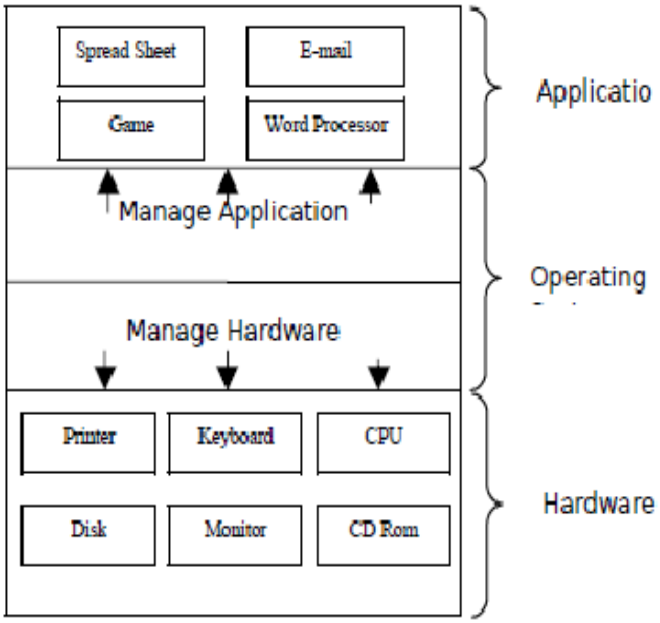
The computer system is built with the following general components

i) Hardware
ii) Application Software
iii) Operating System

(i) **Hardware:** This includes the physical components such as CPU, Keyboard, Hard disk and Printer.

(ii) **Application Software:** These are the programs that are used to accomplish specific tasks.

(iii)**Operating System:** It is the component or the set of programs to manage and control the hardware as well as co-ordinate the applications. Each system must have at least have the hardware and the OS.



Functions of an Operating System

- **Command interpretation:** The CPU can't understand the commands keyed in by a user. It is the function of the OS to make it understand.
- **Peripheral Interfaces:** The OS also has to take care of the devices attached to the system. The OS oversees communication between these devices and the CPU.
- **Memory management:** The OS handles the extremely important job of allocating memory for various processes running on the system.
- **Process management:** This is required if several programs must run concurrently. CPU time would then have to be rationed out by the OS to ensure that no programs get more than its fair share of the processor time.

Services of an OS:

1. Process Management
2. File Management
3. I/O Management
4. Scheduling
5. Security Management

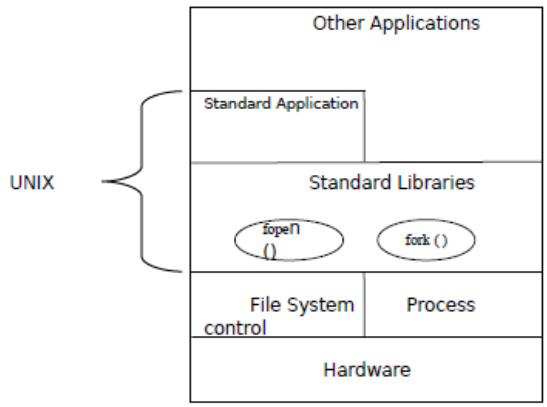
UNIX OPERARTING SYSTEM:

In the mid-1960s, AT &T Bell Laboratories developed a new OS called Multics. Multics was intended to supply large scale computing services as a utility; much like electrical power. In 1969 Ken Thompson, Dennis Ritchie and others developed and simulated an initial design for a file system that later evolved into the UNIX file system. The whole UNIX was rewritten in 'C' language in 1973.Today, UNIX is a giant Os and is much powerful than most of its counter parts.tld

UNIX operating System is like a layer between the hardware and the applications that run on the computer .It has functions that run on the computer. It has

functions that manage the executing applications.

UNIX system is an OS, which includes the traditional system components. UNIX system includes a set of libraries and a set of applications.



KERNEL is the heart of UNIX OS that manages the hardware and the executing process. The UNIX system views each device as a file called a device file. It implements security controls to protect the safety and privacy of information. The Unix System allocates resources including use of the CPU and mediates accesses to the hardware.

Application portability is the ability of a single application to be executed on various types of computer hardware without being modified. This is one of the important advantages of UNIX.

FEATURES OF UNIX:

1. **Multitasking**

   Multitasking is the capability of the Os to perform various tasks simultaneously. i.e. A single user can run multiple programs concurrently.

2. **Multiuser Capability**

   Multiuser capability allows several users to use the same computer to perform their tasks. Several terminals are connected to a single powerful computer and each user can work with their terminals.

3. Security

Unix allows sharing of data. Every user must have a Login name and a password. So, accessing another user's data is impossible without permission.

4. Portability

Unix is a portable because it is written in high level languages so it can run on different computers.

5. Communication

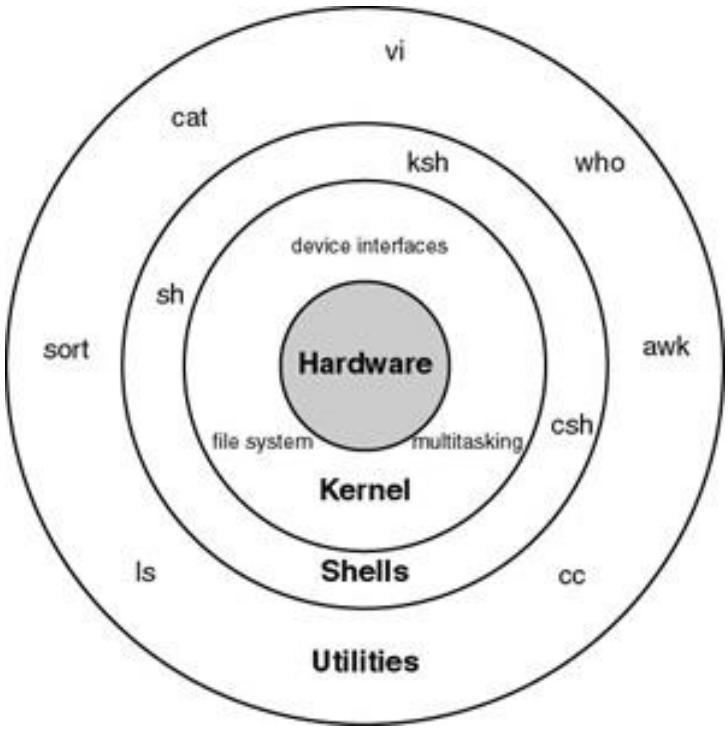
Unix supports communication between different terminals connected to the Unix server and also between the users of one computer to the users of another computer located elsewhere in the network.

6. Programming Facility

Unix is highly programmable, the Unix shell programming language has the conditional statements and control structures and variables.

UNIX ARCHITECTURE

The functioning of Unix OS is handled in 3 ways.



Kernel

It is the core of the OS. It controls all the tasks and carries out all the functions of an OS such as memory and file management etc., and it keep track of the programs that are executing. It also handles the information exchange between the terminals such as tape drives and printers etc.,

Functions of Kernel

1. Allocating and deallocating memory to each and every process.
2. Receiving instructions from the shell and carrying them out.
3. Managing files that held on various storage devices.
4. Scheduling, Coordinating and assigning various input/output devices simultaneously.
5. Enforcing security measures.
6. Providing Network services.
7. Providing Utility services.
8. Coordinating each and every process with signal handling.
9. Providing administrative functions or utilities.

Shell

It is the command interpreter of the OS. The commands given from the user are moved to the shell. The shell analyses and interprets these commands into the machine understandable form. The commands can be either typed in through the command line or contained in a file called shell script. Hence, Shell acts as an interface between the user and the kernel.

UNIX has a variety of shells, they are:

(i) Bourne Shell: It is developed by Steve Bourne and it is the most popular shell and widely used. This shell comes bundled with almost every Unix system.

(ii) Korn shell: It is developed by David G.Korn. This is superset of Bourne shell an
it has more capabilities.

(iii) C Shell: It is developed by Bil Joy. It is similar to C Programming language.

Features of Shell

1. All communications between user and Kernel takes place through the shell.
2. It allows the tasks to run on background.
3. It also enables us to construct scripts like a programming language.
4. A group of files can be executed using a single command.

Starting a UNIX session – Logging In

A user of Unix based system works as a user terminal. After the boot procedure is completed, that is the operating system is loaded in memory, the following message appears at each user terminal:

Logging

Each user has a identification called the user name are the login name which has to be entered when the login: message appears. The user is then asked to enter the password. Unix keeps track of all the Unix user names and the information about identity in a special file. If the login name entered does not match with any of the user names it displays the login message again. This ensures that, only authorized people use the system. When a valid user name is entered at the terminal the dollar symbol is displayed on the screen this is the Unix prompt.

Ending a UNIX session –Logging Out

Once a user has logged into the system the users works session continues until the user instructs the shell to terminate the session. This is done by pressing the ctrl and ‘d keys’ together or typing exit at the dollar prompt. Then the systems display the login:

Prompt on the screen.

RESULT:

Thus the unix os was studied successfully.

Ex.NO:1 (B) STUDY OF VI EDITOR & SHELL PROGRAMMING
DATE:

AIM:

To study about the vi editor & shell programming

An editor is program that allows to see a portion of a file on the screen and to modify characters and lines by simply typing at the cursor position. There are number of editors that may be included with the Unix system, including ed, ex, vi and EMACS.

The vi Editor

Vi - vi stands for visual.

Vi is a full screen editor and is widely acknowledged as one of the most powerful editors available. It is a full screen editor that allows the user to view and edit the entire document at the same time.

The vi editor was written in the University of California at Berkeley by Bill Joy, who is one of the co-founder of Sun Microsystems.

Features of vi Editor

1. It is very easy to learn and has more powerful and exciting features.
2. It works in great speed.
3. vi is case sensitive.
4. vi has powerful undo features than most other word processors ,but it has no formatting features.

Modes of vi Editor

Vi editor works in three modes of operations specified below:

Command Mode: In this mode, all the keys pressed by the user are interpreted to be editor commands. No text is displayed on the screen, even if corresponding key is pressed on keyboard.

Insert Mode: This mode permits to insert new text, editing and replacement of existing text. Once vi editor is in the insert mode, letters typed in the keyboard are echoed on the screen.

The ex or escape colon (:) Mode: This mode allow us to give commands at the command line. The bottom line of the vi editor is called the command line.vi uses the command line to

display messages and commands.

Starting with Vi editor

Syntax: vi filename

Moving the cursor

The cursor movement commands are:

Command	**Action**
H or backspace	Left one character
l or spacebar	Right one character
K or -	Up one line
J or +	Down one line
I	Moves forward a word
#b	Moves back a word
#e	Moves to the last character in the word
F[character]	Moves right to the specified character in a line
T[character]	Moves right and places it one character before the specified character
0 or ^	Moves to the beginning of the file
#$	Moves to the end of the file
L	Moves to the last line of the file
G	Moves to the specified line number

Editing the file

- Open the file using $ vi filename
- To add text at the end of the file, position the cursor at the last character of the file.
- Switch from command mode to text input mode by pressing ‘a’.
- Here ‘a’ stands for append.
- Inserting text in the middle of the file is possible by pressing ‘i’. The editor accepts and inserts the typed character until Esc key is pressed.

Command	Purpose
I	Inserts text to the left of the cursor
I	Inserts text at the beginning of the line
A	Append text to the right of the cursor
A	Appends text at the end of the line
O	Appends a new line below
O	Appends a line above

Deleting Text

For deleting a character, move the cursor to the character , press ‘x’. The character will disappear.

Command	Purpose
X	Deletes one character
Nx	Deletes n number of characters
#x	Deletes on character at the cursor position
#X	Deletes on the character before the cursor position
D$ or d	Deletes a line from the cursor position to the end of the line
D0	Deletes from the cursor position to the starting of the line
#dd	Deletes the current line where the cursor is positioned
#dw	Deletes the word from the cursor position to the end of the word

The undo features

u-undo the recent changes

U- undo all changes in the current line

Saving text

:w – save the file and remains in edit mode
:wq – save the file and quits from edit mode
:q – quit without changes from edit mode

Quitting vi

Press zz or ':wq' in command mode.

SHELL PROGRAMMING

The format for the various conditional statements and looping statements and the relational operators used for those conditions are given below.

if condition

```
if condition then
        execute commands
fi
```

if – else condition

```
if condition then
        execute commands
else
        execute commands
fi
```

if – elif condition – multi way branching

```
if condition then
        execute commands elif condition
then
        execute commands
else
        execute commands
fi
```

Relational Operators

Operator	*Meaning*
-eq	Equal to
-ne	Not equal to
-gt	Greater than
-ge	Greater than or equal to
-lt	Less than
-le	Less than or equal to

Case condition

case expression in

pattern1) execute commands ;; pattern2) execute commands ;;

pattern3) execute commands ;;

……..

esac **while looping**

While expression Do

execute commands

done

until: while's complement until [condition]

for looping

for *variable* in list *d*o

execute commands done

RESULT:

Thus the study of vi editor &shell programming was studied successfully.

Ex.No:2(A) **BASIC UNIX COMMANDS**
Date:

Aim:

To Study about basic Unix Commands.

FILE MANIPULATION COMMANDS

Command : Cat
Purpose : It is used to display the contents of the file as well as used to create a new file.
Syntax : cat <file name >
Example : $ cat devi

Command : More
Purpose : It is used to display the contents of the file on the screen at a time.
Syntax : more <file name >
Example : $ more devi

Command : Wc
Purpose : It is used to count the number of lines ,words and characters in a file or group of files.
Syntax : wc [options] <file name >
Example : $ wc –l devi

Command : File
Purpose : It is used to determine the type of the file.
Syntax : file <file name >
Example : $ file devi

Command : Spell
Purpose : It is used to find the spelling errors in the file.
Syntax : spell [options] <file name >
Example : $ spell -b devi

Command : Split
Purpose : It is used to split the given file into smaller pieces of given size.
Syntax : split –size <file name > < splitted file name >
Example : $ split –2 devi de

Command : Cp
Purpose : It is used to copy one or more files.
Syntax : cat <source file name > <destination file name>
Example : $ cp devi latha

Command : Mv
Purpose : It is used to move a file within a directory with different names and also used to move a file to different directory with its original name.
Syntax : mv <source file name > <destination file name>
Example : $ mv devi jeya

Command : Rm
Purpose : It is used to remove a file from the disk.
Syntax : rm <file name >
Example : $ rm devi

GENERAL PURPOSE COMMANDS

Command : Banner
Purpose : It is used to display its argument in large letters.
Syntax : banner < string >
Example : $ banner BOOM

Command : Who
Purpose : It is used to get the information about all the users currently working in the system.
Syntax : who
Example : $ who

Command : Who am i
Purpose : It is used to know in which terminal the user is currently logged on.
Syntax : who am i
Example : $ who am i

Command : Date
Purpose : It is used to display the system date and time.
Syntax : date
Example : $ date

Command : Cal
Purpose : It prints the calendar for the specified year and month.
Syntax : cal <month> <year>
Example : $ cal 05 2003

Command : Id
Purpose : It is used to display the login name.
Syntax : id
Example : $ id

Command : Clear
Purpose : It is used to clear the screen.
Syntax : clear
Example : $ clear

Command : Tput
Purpose : It is used to manipulate the screen.
Syntax : tput < attributes >
Example : $ tput rmso

Command : Uname
Purpose : It is used to display the details about the OS in which we are working.
Syntax : uname [options]
Example : $ uname –n

Command : Tty
Purpose : It is used to know the terminal name on which we work.
Syntax : tty
Example : $ tty

Command : Pwd
Purpose : It is used to display the absolute pathname of current working directory.
Syntax : pwd
Example : $ pwd

Command : Bc
Purpose : It is used to perform simple mathematical calculations.
Syntax : bc <operation>
Example : $ bc 3+5 8 ^d

Command : Ls
Purpose : It is used to display the files in the current working directory.
Syntax : ls [options] <arguments>
Example : $ ls –p

Command : Echo
Purpose : It echoes the argument on the standard output device.
Syntax : echo [options] <string>
Example : $ echo 'BOOM'

Command : Man
Purpose : It gives details about the unix commands.
Syntax : man < command name >
Example : $ man echo

COMMAND GROUPING & FILTER COMMANDS

Command : Head
Purpose : It is used to display the top portion of the file.
Syntax : head [options] <file name>
Example : $ head -5 devi

Command : Tail
Purpose : It is used to display the bottom portion of the file.
Syntax : tail [options] <file name >
Example : $ tail –5 devi

Command : Pr
Purpose : It is used to display the contents of the file by separating them into pages and each page begins with the header information.
Syntax : pr [options] <file name >
Example : $ pr devi

Command : Cut
Purpose : It is used to extract selected fields or columns from each line of one or more files and display them on the standard output device.
Syntax : cut [options] <file name >
Example : $ cut –c5 devi

Command : Paste
Purpose : It concatenates the line from each input file column by column with tab characters in between them.
Syntax : paste [options] <file name >
Example : $ paste f1 f2

Command : Join
Purpose : It is used to extracts common lines from two sorted files and there should be the common field in both file.
Syntax : join [options] <file name1 > <file name 2>
Example : $ join –a1 f1 f2

Command : Uniq
Purpose : It compares adjacent lines in the file and displays the output by eliminating duplicate adjacent lines in it.
Syntax : uniq [options] <file name >
Example : $ uniq -c devi

Command : Sort
Purpose : It sorts one or more files based on ASCII sequence and also to merge the file.
Syntax : sort [options] <file name >
Example : $ sort -r devi

Command : Nl
Purpose : It is used to add the line numbers to the file.
Syntax : nl [options] [filename]
Example : $ nl devi

Command : Tr
Purpose : It is used to translate or delete a character or a string from the standard input to produce the required output.
Syntax : tr [options] <string1> <string2>
Example : $ tr –t ‘a’ ‘b’ < devi>

Command : Tee
Purpose : It is used to read the contents from standard input or from output of another command and reproduces the output to both in standard output and direct into output to one or more files.
Syntax : tee [options] <file name >
Example : $ tee date dat.txt

Command : grep
Purpose : It is used to search the specified pattern from one or more files.
Syntax : grep [options] <pattern> <file name >
Example : $ grep “anand” devi

RESULT:

Thus the above unix commands are executed and the output was verified successfully.

Ex.No:2 (B) DIRECTORY COMMANDS AND PROCESS MANAGEMENT
Date: COMMANDS

Aim:

To study about directory handling and Process Management Commands

DIRECTORY COMMANDS

Command : mkdir
Purpose : It is used to create new directory or more than one directory.
Syntax : mkdir <directory name >
Example : $ mkdir riya

Command : rmdir
Purpose : It is used to remove the directory if it is empty.
Syntax : rmdir <directory name >
Example : $ rmdir riya

Command : cd
Purpose : It is used to change the control from one working directory to another specified directory.
Syntax : cd <directory name >
Example : $ cd riya

Command : cd ..
Purpose : It is used to quit from current directory and move to the previous directory.
Syntax : cd ..
Example : $ cd ..

PROCESS COMMANDS

Command : echo $$
Purpose : It is used to display the process number of the current shell.
Syntax : echo $$
Example : $ echo $$

Command : ps
Purpose : It is used to display the attributes of a process.
Syntax : ps
Example : $ ps
$ ps –f (Display the ancestry of a process)
$ ps –u (Display the activities of a user)
$ ps –a (Lists processes of all users but not the system processes)

Command : &
Purpose : It is shell operator which is used to run a process in the background.
Syntax : <command> &
Example : $ sort emp.txt &

Command : nohup
Purpose : It permits the execution of the process even after the user has logged out.
Syntax : nohup <command>
Example : $ nohup sort emp.txt (result is available on nohup.out)

Command : kill
Purpose : It is used to terminate the process.
Syntax : kill <PID>
Example : $ kill 105

Command : kill $!
Purpose : $! is the system variable which stores the process id of the last background job. The command kill $! is used to kill the last process.
Syntax : kill $!
Example : $ kill $!

Command : at
Purpose : It is used to execute the process at the time specified.
Syntax : echo <time>
Example : $ at 14:08 (or) $ at 3 PM (or) $ at 4 :50 AM

RESULT:

Thus the above directory commands are executed and the output was verified successfully.

Ex.NO:3 (A) **SIMPLE SHELL PROGRAMS**

DATE:

GETTING AND DISPLAYING THE ACADEMIC AND PERSONAL DETAILS

Aim:

To write a shell program to get the input and display the academic, personal details.

Algorithm:

1. Get name, age, and address from the user.
2. Print that message as similar.
3. Get mark1, mark2, and mark3 from the user.
4. Print that message as similar.

Program:

```
echo –n “Enter the name”  read s
echo –n “Enter the age”  read a
echo –n “Enter the address”  read adr
echo –n “The name is $s”
echo –n “The age is $a”
echo –n “The address is $adr”
echo –n “Enter the mark1”  read m1
echo –n “Enter the mark2”  read m2
echo –n “Enter the mark3”  read m3
echo –n “The mark1 is $m1”
echo –n “The mark2 is $m2”
echo –n “The mark3 is $m3”
```

RESULT:

Thus the above shell program was executed and the output was verified successfully.

Ex No. 3 (B) **ARITHMETIC OPERATIONS**

DATE:

Aim:

To write a shell program to perform all arithmetic operations

Algorithm:

1. Read two numbers from the user.
2. Add both the numbers.
3. Print the summation of two numbers.
4. Subtract the second number from the first number
5. Print the result.
6. Multiply the two numbers
7. Print the result.
8. Divide the first number by second number.

Program:

```
echo enter the numbers
read n1          n2
n3=`expr $n1     + $n2`
n4=`expr $n1     - $n2`
n5=`expr $n1     \* $n2`
n6=`expr $n1     / $n2`
echo summation of $n1 and $n2 is $n3
echo difference of $n1 and $n2 is $n4
echo multiplication of $n1 and $n2 is $n5
echo division of $n1 by $n2 is $n6
```

RESULT:

Thus the above shell program was executed and the output was verified successfully

Ex No: 3 C) **PROGRAMS USING CONDITIONAL STATEMENTS**

DATE: **ODD (OR) EVEN**

Aim:

To write a shell program to check whether the given number is odd or even.

Algorithm:

1. Get a number from the user.
2. Divide the number by 2.
3. Check the remainder of the division.
4. If it is zero then the given number is even.
5. Otherwise the given number is odd.

Program:

```
echo –n “enter the number”
read num
x = `expr $num % 2`
if [ $x –eq 0 ]
then
echo “$x is an even number”
else
echo “$x is a odd number”
fi
```

RESULT:

Thus the above shell program was executed and the output was verified successfully

Ex. No. 3(D) **POSITIVE OR NEGATIVE NUMBER**

DATE:

Aim:

To write a shell program to check whether the given number is +ve or –ve or zero.

Algorithm:

1. Get a number from the user.
2. Check the number is greater than or less than zero or equal to zero.
3. If it is greater than zero then the given number is +ve.
4. If it is less than zero then the given number is –ve. Otherwise the given number is zero.

Program:

```
echo –n “enter the number”
read num
if [ $num –gt 0 ] then
echo “$num is a positive number”
elif [ $num –lt 0 ]
then
echo “$num is a negative number”
else
echo “$num is zero”
fi
```

RESULT:

Thus the above shell program was executed and the output was verified successfully

Ex. No: 3(E) **BIGGEST AMONG THREE NUMBERS**

DATE:

Aim:

To write a shell program to find a biggest number among three numbers.

Algorithm:

1. Get three numbers from the user.
2. If the 1st number is greater than other two numbers then the 1st number is a biggest numbers.
3. Otherwise if the 2nd number is greater than 3rd number then the 2nd number is a biggest number.
4. Otherwise 3rd number is a biggest number.

Program:

```
echo -n "Enter the numbers"
read num1 num2 num3
if [ $num1 -gt $num2 ] && [ $num1 -gt $num3 ]
then
    echo "$num1 is a biggest number"
        elif [ $num2 -gt $num3 ] then
    echo "$num2 is a biggest number"
    else
    echo "$num3 is a biggest number"
fi
```

RESULT:

Thus the above shell program was executed and the output was verified successful

Ex No.3(F) **ARITHMETIC OPERATIONS - SWITCH CASE**

DATE:

Aim:

To write a shell program to perform all the arithmetic operations using case statements.

Algorithm:

1. Get two numbers from the user.
2. Select the options .
3. Execute arithmetic operations using case statements.. 4.Print the output

Program:

```
echo “Enter two numbers” read a b
echo “menu” echo “1.add” echo
“2.sub” echo “3.mul” echo “ 4. div”
echo “Enter your option” read option
case $option in
1)c=`expr $a + $b`;;
2)c=`expr $a - $b`;;
3)c=`expr $a \* $b`;;
4)c=`expr $a / $b`;;
esac
echo $c
```

RESULT:

Thus the above shell program was executed and the output was verified successfully

Ex No.3(G) FINDING THE AREA OF CIRCLE, SQUARE, RECTANGLE AND TRIANGLE

DATE:

Aim:

To write a shell program to find the area of circle, square, rectangle and triangle using case statements.

Algorithm:

1. Get two numbers from the user.
2. Select the option.
3. Execute the statement to find the area using case statements..
4. Print the output

Program:

```
echo 1.circle
echo 2. rectangle
echo 3.square
echo 4.triangle
echo enter your choice
read opt
case $opt in
1) echo enter the radius
 read r
 area=`expr $r \* $r \* 22 / 7`
 echo $area;;
2) echo enter the length and breadth
read l b
 area=`expr $l \* $b`
 echo $area;;
3) echo enter the side
read s
 area=`expr $s \* $s`
 echo $area;;
4)echo enter the height and breadth
read b h
 area=`expr $b \* $h \* 1 / 2`
 echo $area;;
 *) echo wrong choice;;
 esac
```

RESULT:

Thus the above shell program was executed and the output was verified successfully

TESTING AND LOOPS

Ex.NO: 3 (H) **FIBONACCI SERIES**
DATE:

Aim:

To write a shell program to generate a fibonacci series of first n numbers

Algorithm:

1. Get the value n from the user.
2.Initial numbers of fibonacci series is 0,1.
3.Using the sum of last adjacent two numbers generate a next number in the series.
4.Repeat the step 3 until n numbers are generated.

Program:

```
echo -n "enter the
limit"
read n
echo "the fibonacci series is : "
 b=0
c=1
i=0
if [ $n -ge 2 ]
then
   echo -n " $b $c"
   n=`expr $n - 2`
   while [ $i -lt $n ]
   do
      a=`expr $b + $c`
      b=$c
      c=$a
      echo -n " $c"
      i=`expr $i + 1`
   done
fi
```

RESULT:

Thus the above shell program was executed and the output was verified successfully

Ex No. 3 (I) **ARMSTRONG NUMBER**
DATE:

Aim:

To write a shell program to Check Whether the given number is Armstrong or not.

Algorithm:

1. Get the number from the user.
2. Extract each and every digit of the given number.
3. Find the cube of every digit and calculate the sum of all cubes.
4. Check whether the sum is equal to the given number, if it is equal, the given number is Armstrong number otherwise not.

Program:

```
echo -n "enter the number"
read num
x=$num
sum=0
while [ $num -gt 0 ] do
  y=`expr $num % 10`
  z=`expr $y \* $y \* $y`
  sum=`expr $sum + $z`
  num=`expr $num / 10`
done
if [ $x -eq $sum ] then
  echo $x is an Armstrong number
  else
  echo $x is not an Armstrong number
fi
```

RESULT: Thus the above shell program was executed and the output was verified successfully.

Ex No: 3 (J) **SUM OF N EVEN NUMBERS**
DATE:

Aim:

To write a shell program to find the sum of even numbers

Algorithm:

1. Start the program
2. Read the number
3. Initialize i=2
4. Calculate sum=sum+1
5. Print the sum
6. Stop the program execution

Program:

```
echo “enter the number of upper limit”
read n
I=2
C=0
while [ $I –lt $n ] do
echo “number is $I”
I=`expr $I + 2’
C=`expr $I + $C’
done
echo “ the sum is $c”
```

RESULT:

Thus the above shell program was executed and the output was verified successfully

Ex. No. 3 (K) **COMBINATIONS OF 1 2 3**

DATE:

Aim:

To write a shell program to generate all combinations of 1 2 3 using for loop

Algorithm:

1. Assign values 1 2 3 to first for loop
2. Form another for loop within the first for loop and assign values 1 2 3
3. Within the second for loop execute another for loop with values 1 2 3
4. Close all for loops
5. Display the outputs

Program:

```
for I in 1 2 3 do
for j in 1 2 3 do
for k in 1 2 3 do
echo $I $j $k
done
done
done
```

RESULT: Thus the above shell program was executed and the output was verified successfully

Ex. No: 3(L) **N^{th} POWER OF A GIVEN NUMBER**

DATE:

Aim:

To write a shell program to compute the n^{th} power of a given number

Algorithm:

1. Get the value for the power & number to be computed.
2. Initialize the value for I as 2.
3. Set the variable ans as a.
4. Repeat the steps 5,6 until the value of I ie less than or equal to the power value.
5. Multiply the value of ans and a.
6. Increment the I value.
7. Display the value ans as result

Program:

```
echo enter the number and its
power
read a pow
i=2
ans=$
a
while [ $i -le $pow
] do
ans=`expr $ans \*
$a`
i=`expr $i + 1`
done
echo
$ans
```

RESULT: Thus the above shell program was executed and the output was verified successfully

COMMAND LINE SUBSTITUTION

Ex No.3(M) **PALLINDROME CHECKING**

DATE:

Aim:

To write a c program to check whether the given string is palindrome or not using Command line substitution.

Algorithm:

1. Get the filename and string in a command line.
2. Reverse the original string.
3. If the original string is equal to the reversed string then display a given string is palindrome.
4. Otherwise display a given string is not palindrome.

Program:

```
#include <stdio.h>
#include <string.h>
void main()
{
      char string[25], reverse_string[25] = {'\0'};
      int i, length = 0, flag = 0;
      printf("Enter a string \n");
      gets(string);
      for (i = 0; string[i] != '\0'; i++)
     {
        length++;
     }
     printf("The length of the string '%s' = %d\n", string, length);
     for (i = length - 1; i >= 0 ; i--)
     {
        reverse_string[length - i - 1] = string[i];
     }
        for (flag = 1, i = 0; i < length ; i++)
     {
        if (reverse_string[i] != string[i])
           flag = 0;
     }
     if (flag == 1)
       printf ("%s is a palindrome \n", string);
     else
       printf("%s is not a palindrome \n", string);
  }
```

RESULT:

Thus the above shell program was executed and the output was verified successfully

Ex No.3(N) **CONCATENATION PF TWO STRING**

DATE:

Aim:

To write a c program to concatenate given two strings using Command line substitution.

Algorithm:

1. Get the two strings in command line.
2. Copy the two arguments in two variables from the command line.
3. Concatenate the two strings using the built – in concatenate function.
4. Display the resultant string.

Program:

```
#include <stdio.h>
 #include <string.h>
main(int argc, char *argv[])
{
char str1[10],str2[20],str3[10];
strcpy(str1,argv[1]);
strcpy(str2,argv[2]);
printf("%s",strcat(str1,str2));
}
```

RESULT:

Thus the above shell program was executed and the output was verified successfully.

Ex.No.4(A) **C PROGRAMMING ON UNIX**
Date: **BIGGEST AMONG 'N' NUMBERS USING FUNCTIONS**

AIM:

To write a C program to find biggest among list of numbers.

ALGORITHM:

1. Get the value of n as input.
2. Get the list of numbers in an array of size n.
3. Set the initial value as a biggest value.
4. Set the value of I is 1.
5. Compare it with other elements in an array.
6. If the biggest element is less than array element then interchange both the values.
7. Repeat steps 4 & 5 until I reaches n-1.
8. Display the biggest value.

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
float largest(float a[],int n);
void main()
{
int n;
float value[10];
clrscr();

printf("enter the value of n\n");
scanf("%d",&n);
for(i=0;i<n;i++)
{
scanf("%f",&value[i]);
}
printf("The Largest number among set of numbers is\n");
printf("%f",largest(value,n));
getch();
}
float largest(float a[],int n)
{
int i; float max;
max=a[0];

for(i=1;i<n;i++)
{
if(max<a[i])
{
```

```
max= a[i];
}
}
return(max);
}
```

RESULT:

Thus the above shell program was executed and the output was verified successfully

Ex.No.4(B) **SWAPPING OF TWO NUMBERS USING POINTERS**
Date:

AIM:

To write a C program to swap two numbers using pointers.

ALGORITHM:

1. Declare the two pointer variables.
2. Get the value for both the variables.
3. Swap the values.
4. Print the result.

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
void swap(int *,int *);
void swap1(int,int);
void main()
{
int a,b,c,d;
printf("Enter the values of a and b:=   ");
scanf("%d %d",&a,&b);
printf("Enter the values of c and d:=   ");
scanf("%d %d",&c,&d);
printf("\n BEFORE SWAPPING : ");
printf("\n The value of a and b is : %d\t %d ",a,b);
printf("\n The value of c and d is : %d\t %d ",c,d);
printf("\n AFTER SWAPPING : ");
swap(a,b);
swap1(&c,&d);
printf("\n Method is:-Call by Value");
printf("\n ***************************");
printf("\n The value of a and b is : %d\t %d",a,b);
printf("\n Method is:-Call by Address or Reference");
printf("\n ***************************");
printf("\n The value of c and d is : %d\t %d",c,d);
}
void swap(int *c,int *d)
{
int t;
 t=*c;
*c=*d;
```

```
   *d=t;
   }
  void swap1(int a,int b)
{
int t;
 t=a;
 a=b;
 b=t;
}
```

RESULT:

Thus the above shell program was executed and the output was verified successfully.

Ex.No.5(A) ***ls – System Calls***
Date:

AIM:

To write a C program to illustrate the simulation of ls command.

Algorithm:

1. Start.
2. Read the directory name.
3. Check the directory in unix.
4. Display the directory name.
5. Stop.

Program:

```
#include<dirent.h>
#include<stdio.h>
#inclue<stdlib.h>
main()
{
char dirname[10];
DIR *p;
struct dirent *d;
printf("Enter directory name");
scanf("%s",dirname);
p=opendir(dirname);
if(p==NULL)
{
perror("Cannot find dir.");
exit(-1);
}
while(d=readdir(p))
printf("%s\n",d->d_name);
}
```

RESULT:

Thus the above shell program was executed and the output was verified successfully

Ex.No.5(B) ***grep – System Calls***
Date:

AIM:

To write a C program to illustrate the simulation of grep command.

Algorithm:

1. Start.
2. Read the file name.
3. Read the pattern to be searched.
4. Check whether the pattern present in the input file or not.
5. If present, display the line which has the pattern.
6. Stop.

Program:

```
#include<stdio.h>
#include<string.h>
main()
{
char fn[10],pat[10],temp[200];
FILE *fp;
printf("\n Enter file name : ");
scanf("%s",fn);
printf("Enter the pattern to be searched :“);
scanf("%s",pat);
fp=fopen(fn,"r");
while(!feof(fp))
{
fgets(temp,1000,fp);
if(strstr(temp,pat))
printf("%s",temp);
}
fclose(fp);
}
```

RESULT:

Thus the above shell program was executed and the output was verified successfully

Ex.No.5(C) ***execl – System Calls***
Date:

AIM:

To write a C program to illustrate the execl command using system Calls

Algorithm:

1. Start.
2. Assign fork() to pid.
3. If pid equals to zero, then it becomes child process.
4. Else cannot create process.
5. Else it is parent.
6. Stop.

Program:

```
#include<stdio.h>
#include<unistd.h>
main()
{
int pid;
pid=fork();
if(pid==0)
{
printf("child process");
execl("/bin/ls","ls","-r",(char*)0);
}
else if(pid<0)
{
printf("cannot create process");
}
else
{
printf("parent process");
}
}
```

RESULT:

Thus the above shell program was executed and the output was verified successfully

Ex.No.5(D) ***execv – System Calls***
Date:

AIM:

To write a C program to illustrate the execv command using system Calls

Algorithm:

1. Start.
2. Assign fork() to pid.
3. If pid equals to zero the display the files in long format.
4. Else display parent or cannot create process.
5. Stop.

Program:

```
#include<stdio.h>
main()
{
int pid;
pid=fork();
if(pid==0)
{
char *argv[]={“ls”,”-l”,”-r”,(char*)0};
execv(“bin/ls”,argv);
printf("child process");
}
else if(pid<0)
{
printf("cannot create");
}
else
{
printf("parent process");
}
}
```

RESULT:

Thus the above shell program was executed and the output was verified successfully

Ex.No.5(E) ***wait – System Calls***
Date:

AIM:

To write a C program to illustrate the wait command using system Calls

Algorithm:

1. Start.
2. Assign pid as fork().
3. If it equals to zero then print child process.
4. Else wait for some time and display parent process.
5. Stop.

Program:

```
#include<stdio.h>
main()
{
int pid;
pid=fork();
if(pid==0)
{
printf("child process");
}
else if(pid<0)
{
printf("cannot create");
}
else
{wait();
printf("parent process");
}
}
```

RESULT:

Thus the above shell program was executed and the output was verified successfully

Ex.No.5(F) ***getpid – System Calls***
Date:

AIM:

To write a C program to illustrate the getpid command using system Calls

Algorithm:

1. Start.
2. Assign fok() to pid.
3. If pid equals to zero, then call sleep with 3 minutes and print the pid for parnt and child process.
4. Else print the original process id and their parent id.
5. Stop.

Program:

```
#include<stdio.h>
#include<unistd.h>
main()
{
int pid;
pid=fork();
if(pid==0)
{
sleep(3);
printf(“parent process id is %d”,getppid());
printf("child process id is %d”,getpid());
}
else if(pid<0)
{
printf("cannot create");
}
else
{
 printf(“parent process id is %d”,getppid());
printf("child process id is %d”,getpid());
}
}
```

RESULT:

Thus the above shell program was executed and the output was verified successfully

Ex.No.5(G) ***lstat – System Calls***
Date:

AIM:

To write a C program to illustrate the lstat command using system Calls

Algorithm:

1. Start.
2. Read file name to display the status.
3. Display file name, uid,gid, file size and inode number.
4. Stop.

Program:

```
#include<stdio.h>
#include<sys/stat.h>
main(int argc,char **argv)
{
struct stat statbuf;
if(lstat(argv[1],&statbuf)==-1)
printf("Error : cannot find file information ");
else
{
 printf("\n File %s ", argv[1]);
printf("\n Inode number %d",statbuf.st_ino);
printf("\n UID %d",statbuf.st_uid);
printf("\n GID %d",statbuf.st_gid);
printf("\n File size in bytes %d",statbuf.st_size);
}
}
```

RESULT:

Thus the above shell program was executed and the output was verified successfully

Ex.No.5(H) ***open, read and close directory – System Calls***
Date:

AIM:

To write a C program to illustrate the open directory, read directory and close directory command using system Calls

Algorithm:

1. Start.
2. Read the directory name.
3. Display the directory name with ino.
4. Stop.

Program:

```
#include<stdio.h>
#include<stdlib.h>
#include <sys/types.h>
#include <dirent.h>
int main(int argc, char *argv[])
{
DIR *dp;
struct dirent *dirp;
if (argc != 2)
{
printf("a single argument (the directory name) is required\n");
exit(1);
}
if ( (dp = opendir(argv[1])) == NULL)
{
printf("can't open %s\n",argv[1]);
exit(1);
}
while ( (dirp = readdir(dp)) != NULL)
printf("%s %d\n",dirp->d_name,dirp->d_ino);
closedir(dp);
exit(0);
}
```

RESULT:

Thus the above shell program was executed and the output was verified successfully

Ex.No.5(I) ***open, read and write – System Calls***
Date:

AIM:

To write a C program to illustrate the open, read and write for file copy using system Calls.

Algorithm:

1. Start.
2. Read the source and destination file name.
3. Open source file and copy the contents into destination file.
4. Open the destination file in write mode and copy the contents.
5. Close both files
6. Stop.

Program:

```
#include<stdio.h>
#include<fcntl.h>
main()
{
char buf[1000],fn1[10],fn2[10];
int fd1,fd2,n;
printf("Enter source file name ");
scanf("%s",fn1);
l
scanf("%s",fn2);
fd1=open(fn1,O_RDONLY);
n=read(fd1,buf,1000);
fd2=open(fn2,O_CREAT|0666);
n=write(fd2,buf,n);
close(fd1);
close(fd2);
}
```

RESULT:

Thus the above shell program was executed and the output was verified successfully