Parul University

FACULTY OF ENGINEERING AND TECHNOLOGY
BACHELOR OF TECHNOLOGY

OPERATING SYSTEM
(303105252)
4th SEMESTER
COMPUTER SCIENCE & ENGINEERING DEPARTMENT

Laboratory Manual

OPERATING SYSTEM PRACTICAL BOOK

COMPUTER SCIENCE & ENGINEERING DEPARTMENT

PREFACE

It gives us immense pleasure to present the first edition of *Operating Systems Book* for the B.Tech. 2nd year students for PARUL UNIVERSITY.

The Operating Systems theory and laboratory courses at **PARUL UNIVERSITY, WAGHODIA, VADODARA** are designed in such a way that students develop the basic understanding of the subject in the theory classes and then try their hands on the experiments to realize the various physical phenomena learnt during the theoretical sessions. The main objective of the Operating Systems laboratory course is: **Learning New Technology and Understanding internal working as well as programming with different Operating Systems**. All the experiments are designed to illustrate various phenomena in different areas of Programming with Operating System and also to expose the students to various operating systems and their uses.

The objective of this *Operating Systems Practical Book* is to provide a comprehensive source for all the experiments included in the Operating System laboratory course. It explains all the aspects related to every experiment such as: basic underlying physical principle, details of the internal working, how to use this operating system for the desired purpose, the theoretical formalism & formulae, procedure of performing the experiment and how to calculate the desired output from the observations etc. It also gives sufficient information on how to interpret and discuss the obtained results.

We acknowledge the authors and publishers of all the books which we have consulted while developing this Practical book. Hopefully this *Operating system Book* will serve the purpose for which it has been developed.

List of Practical:

1. Study of Basic commands of Linux.

2. Study the basics of shell programming.

3. Write a Shell script to print given numbers sum of all digits.

4. Write a shell script to validate the entered date. (eg. Date format is: dd-mm-yyyy).

5. Write a shell script to check entered string is palindrome or not.

6. Write a Shell script to say Good morning/Afternoon/Evening as you log in to system.

7. Write a C program to create a child process

8. Finding out biggest number from given three numbers supplied as command line arguments

9. Printing the patterns using for loop.

10. Shell script to determine whether given file exist or not.

11. Write a program for process creation using C. (Use of gcc compiler).

12. Implementation of FCFS & Round Robin Algorithm.

13. Implementation of Banker's Algorithm.

PRACTICAL NO: 1

Definition: Study of Basic commands of Linux/UNIX.

Command shell: A program that interprets commands is Command shell.

Shell Script: Allows a user to execute commands by typing them manually at a terminal, or automatically in programs called shell scripts.
A shell is not an operating system. It is a way to interface with the operating system and run Commands.

BASH (Bourne Again Shell)

- Bash is a shell written as a free replacement to the standard Bourne Shell (/bin/sh) originally written by Steve Bourne for UNIX systems.

- It has all of the features of the original Bourne Shell, plus additions that make it easier to program with and use from the command line.

- Since it is Free Software, it has been adopted as the default shell on most Linux systems.

BASIC LINUX COMMANDS:

1. pwd : Print Working Directory

DESCRIPTION:

pwd prints the full pathname of the current working directory.

SYNTAX:

pwd

EXAMPLE:

$ pwd

/home/directory_name

2. cd: Change Directory

DESCRIPTION:

It allows you to change your working directory. You use it to move around within the hierarchy of your file system.

SYNTAX:

cd directory_name

EXAMPLE:

To change into "work directory" in "documents" need to write as follows.

Input: $ cd /documents/work

3. cd ..

DESCRIPTION:

Move up one directory.

SYNTAX:

cd ..

EXAMPLE:

If you are in work directory and want to go to documents then write

cd ..

You will end up in /documents.

4. ls : list all the files and directories

DESCRIPTION:

List all files and folders in the current directory in the column format.

SYNTAX:

ls [options]

EXAMPLE: Using various options

- Lists the total files in the directory and subdirectories, the names of the files in the current directory, their permissions, the number of subdirectories in directories listed, the size of the file, and the date of last modification.

    ls -l

- List all files including hidden files

ls -a

5. cat

DESCRIPTION:

cat stands for "catenate". It reads data from files, and outputs their contents. It is the simplest way to display the contents of a file at the command line.

SYNTAX:

cat filename

EXAMPLES:

- Print the contents of files mytext.txt and yourtext.txt

    cat mytext.txt yourtext.txt

- Print the cpu information using cat command

    cat /proc/cpuinfo

- Print the memory information using cat command

    cat /proc/meminfo

6. head

DESCRIPTION:

head, by default, prints the first 10 lines of each FILE to standard output. With more than one FILE, it precedes each set of output with a header identifying the file name.

If no FILE is specified, or when FILE is specified as a dash ("-"), head reads from standard input.

SYNTAX:

head [option]…[file/directory]

EXAMPLE:

Display the first ten lines of myfile.txt.

head myfile.txt

7. tail

DESCRIPTION:

tail is a command which prints the last few number of lines (10 lines by default) of a certain file, then terminates.

SYNTAX:

tail [option]…[file/directory]

EXAMPLE:

Output the last 100 lines of the file myfile.txt.

tail myfile.txt   -n 100

8. mv : Moving (and Renaming) Files

DESCRIPTION:

The *mv* command lets you move a file from one directory location to another. It also lets you rename a file (there is no separate *rename* command).

SYNTAX:

mv [option] source directory

EXAMPLE:

- Moves the file myfile.txt to the directory destination-directory.

    mv myfile.txt   destination_directory

- Move the file myfile.txt into the parent directory.

    mv myfile.txt ../

- In this case, if JOE1_expenses does not exist, it will be created with the exact content ofjoe_expenses, and joe_expenses will disappear.

    If JOE1_expenses already exists, its content will be replaced with that of joe_expenses (and joe_expenses will still disappear).

    mv joe_expenses JOE1_expenses

9. mkdir : Make Directory

DESCRIPTION:

If the specified directory does not already exist, mkdir creates it. More than one directory may be specified when calling mkdir.

SYNTAX:

mkdir [option] directory

EXAMPLE:

Create a directory named work.

mkdir work

10. cp : Copy Files

DESCRIPTION:

The cp command is used to make copy of files and directories.

SYNTAX:

cp [option] source directory

EXAMPLE:

Creates a copy of the file in the currently working directory named origfile. The copy will be named newfile, and will be located in the working directory.

cp origfile newfile

11. rmdir : Romove Directory

DESCRIPTION:

The rmdir command is used to remove a directory that contains other files or directories.

SYNTAX:

rm directory_name

EXAMPLE:

Delete mydir directory along with all files and directories within that directory. Here, -r is for recursive and –f is for forcefully.

rmdir -rf mydir

12. gedit

DESCRIPTION:

The gedit command is used to create and open a file.

SYNTAX:

gedit filename.txt

EXAMPLE:

To create a file named abc.sh

gedit abc.sh

13. man

DESCRIPTION:

Displays on online manual page or manpage.

SYNTAX:

man   command

EXAMPLE:

To learn about listing files

man ls

14. echo

DESCRIPTION:

Display text on the screen.

SYNTAX:

echo yourtext

EXAMPLE:

Print Hello World on the screen

echo "Hello World"

15. clear

DESCRIPTION:

Used to clear the screen

SYNTAX:

clear

EXAMPLE:

Clear the entire screen

clear

16. whoami

DESCRIPTION:

whoami prints the effective user ID. This command prints the username associated with the current effective user ID.

SYNTAX:

whoami   [option]

EXAMPLE:

Display the name of the user who runs the command.

whoami

17. wc

DESCRIPTION:

wc (word count) command, can return the number of lines, words, and characters in a file.

SYNTAX:

wc [option]… [file]…

EXAMPLE:

- Print the byte counts of file myfile.txt

  wc -c myfile.txt

- Print the line counts of file myfile.txt

  wc -l myfile.txt

- Print the word counts of file myfile.txt

  wc -w myfile.txt

18. grep

DESCRIPTION:

grep command uses a search term to look through a file.

SYNTAX:

grep [option]… Pattern [file]…

EXAMPLE:

Search the word Hello in file named myfile.txt

grep "Hello" myfile.txt

19. free

DESCRIPTION:

Display RAM details in Linux machine.

SYNTAX:

free

EXAMPLE:

To display the RAM details in Linux machine need to write following command.

free

20. pipe (|)

DESCRIPTION:

Pipe command is used to send output of one program as a input to another. Pipes "|" help combine 2 or more commands.

SYNTAX:

Command 1 | command 2

EXAMPLE:

Display lines of input files containing "Aug" and send to standard output
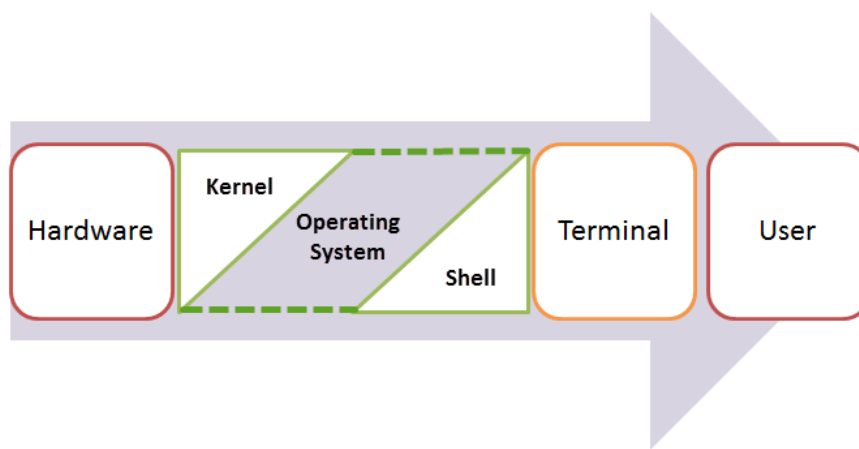
ls -l | grep "Aug"

PRACTICAL NO: 2

Aim: Study the basics of shell programming.

What is a Shell?

An Operating is made of many components, but its two prime components are -

- Kernel
- Shell

A Kernel is at the nucleus of a computer. It makes the communication between the hardware and software possible. While the Kernel is the innermost part of an operating system, a shell is the outermost one.

A shell in a Linux operating system takes input from you in the form of commands, processes it, and then gives an output. It is the interface through which a user works on the programs, commands, and scripts. A shell is accessed by a terminal which runs it.

When you run the terminal, the Shell issues **a command prompt (usually $),** where you can type your input, which is then executed when you hit the Enter key. The output or the result is thereafter displayed on the terminal.

The Shell wraps around the delicate interior of an Operating system protecting it from accidental damage. Hence the name **Shell**.

Types of Shell

There are two main shells in Linux:

1. The **Bourne Shell**: The prompt for this shell is $ and its derivatives are listed below:

- POSIX shell also is known as sh

- Korn Shell also knew as sh
- **B**ourne **A**gain **SH**ell also knew as bash (most popular)

2. The C shell: The prompt for this shell is %, and its subcategories are:

- C shell also is known as csh
- Tops C shell also is known as tcsh

What is Shell Scripting?

Shell scripting is writing a series of command for the shell to execute. It can combine lengthy and repetitive sequences of commands into a single and simple script, which can be stored and executed anytime. This reduces the effort required by the end user.

Let us understand the steps in creating a Shell Script

1. **Create a file using** a **vi** editor(or any other editor). Name script file with **extension .sh**
2. **Start** the script with **#! /bin/sh**
3. Write some code.
4. Save the script file as filename.sh
5. For **executing** the script type **bash filename.sh**

"#!" is an operator called shebang which directs the script to the interpreter location. So, if we use"#! /bin/sh" the script gets directed to the bourne-shell.

Let's create a small script -

```
#!/bin/sh
ls
```

Let's see the steps to create it -
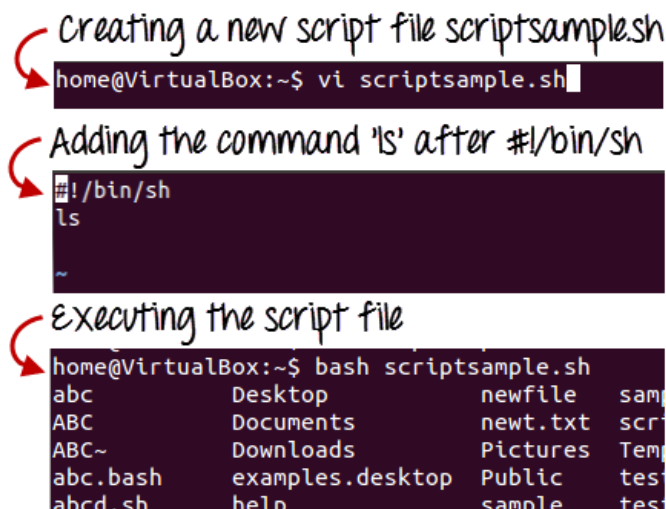
Command 'ls' is executed when we execute the scrip sample.sh file.

Adding shell comments

Commenting is important in any program. In Shell programming, the syntax to add a comment is

```
#comment
```

Let understand this with an example.

What are Shell Variables?

As discussed earlier, Variables store data in the form of characters and numbers. Similarly, Shell variables are used to store information and they can by the shell only.

For example, the following creates a shell variable and then prints it:

```
variable ="Hello"
echo $variable
```

Below is a small script which will use a variable.

```
#!/bin/sh
echo "what is your name?"
read name
echo "How do you do, $name?"
read remark
echo "I am $remark too!"
```

Let's understand, the steps to create and execute the script

Creating the script

```
#!/bin/sh
echo "what is your name?"
read name
echo "How do you do, $name?"
read remark
echo "I am $remark too!"
```

running the scriptfile

```
home@VirtualBox:~$ bash scriptsample.sh
what is your name?
```

Entering the input

script reads the name

```
home@VirtualBox:~$ bash scriptsample.sh
what is your name?
Joy
How do you do, Joy?
```

Entering the remark

```
home@VirtualBox:~$ bash scriptsample.sh
what is your name?
Joy
How do you do, Joy?
excellent
I am excellent too!
home@VirtualBox:~$
```

script repeats the remark

As you see, the program picked the value of the variable 'name' as Joy and 'remark' as excellent.

This is a simple script. You can develop advanced scripts which contain conditional statements, loops, and functions. Shell scripting will make your life easy and Linux administration a breeze.

Summary:

- Kernel is the nucleus of the operating systems, and it communicates between hardware and software
- Shell is a program which interprets user commands through CLI like Terminal
- The Bourne shell and the C shell are the most used shells in Linux
- Shell scripting is writing a series of command for the shell to execute
- Shell variables store the value of a string or a number for the shell to read
- Shell scripting can help you create complex programs containing conditional statements, loops, and functions
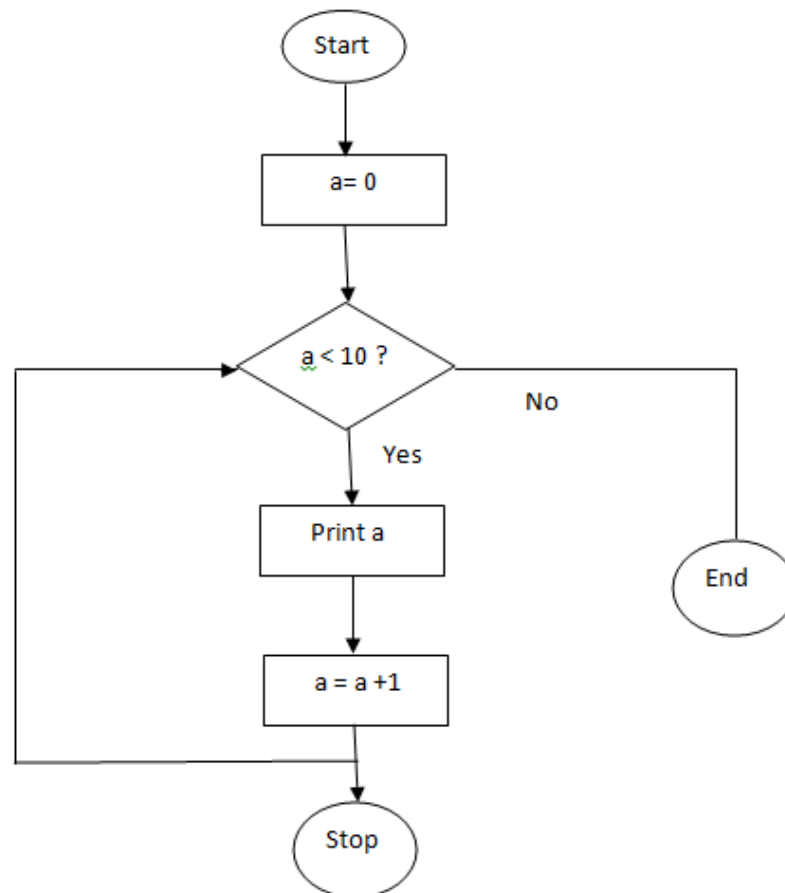
For more detail reference, follow this link: https://www.shellscript.sh/

PRACTICAL NO: 3

Definition: Take any number from the user. Get each and every digit one by one and make addition of that to find the sum of digits of a given number. E.g. if user has entered 342 then the sum of digits is 3+4+2 = 9.

Set 1: Printing numbers from 0 to 9 using while loop.

Flowchart:

Sample Code:

```
a=0
while [  $a  -lt  10  ]
do
      echo  $a
      a=`expr  $a  +  1`
done
```
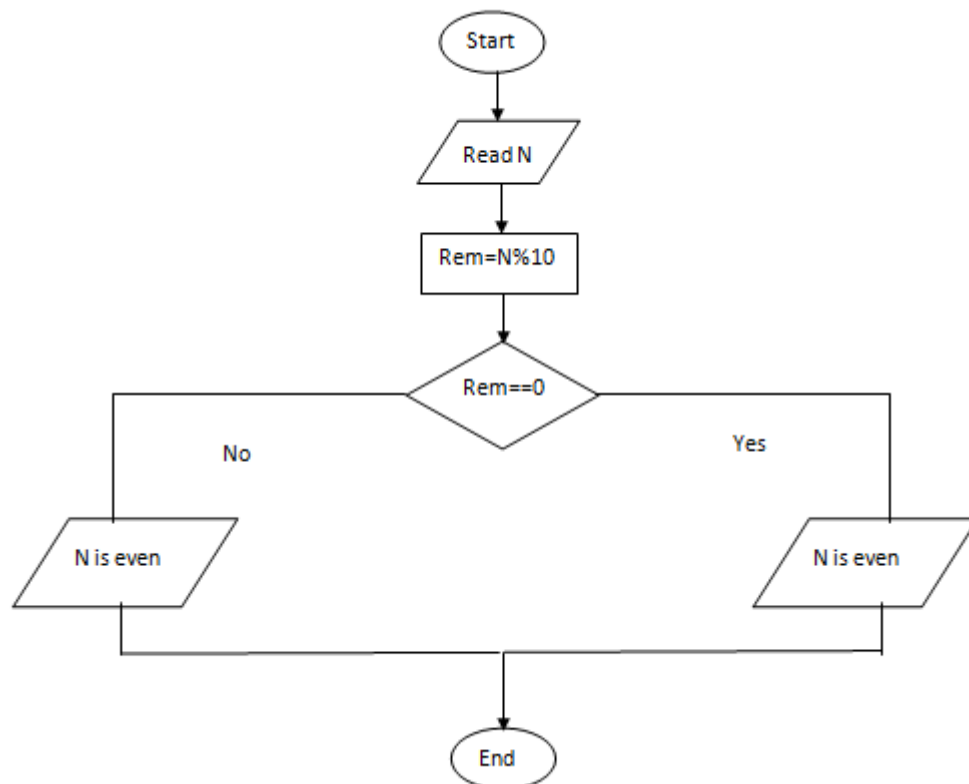
Input: -

Output: To execute shell script need to write: **sh filename**
$sh num.sh

0
1
2
3
4
5
6
7
8
9

Set 2: To find whether a number is even or odd.

Flowchart:

Sample Code:

Input:

Output:

Set 3: Write a shell script for Sum of all digits of a number.

Flowchart:

Sample Code:

Input:

Output:

PRACTICAL NO: 4

Definition: In a year there are 12 months and in each month the no. of days are 30 or 31. For February month, the no. of days is 28 and if it is a leap year then the no. days will be 29. By checking these conditions for days, month and year using various control statements of Linux can validate the date entered by the user. (eg. Date format is: dd-mm-yyyy)

Flowchart:
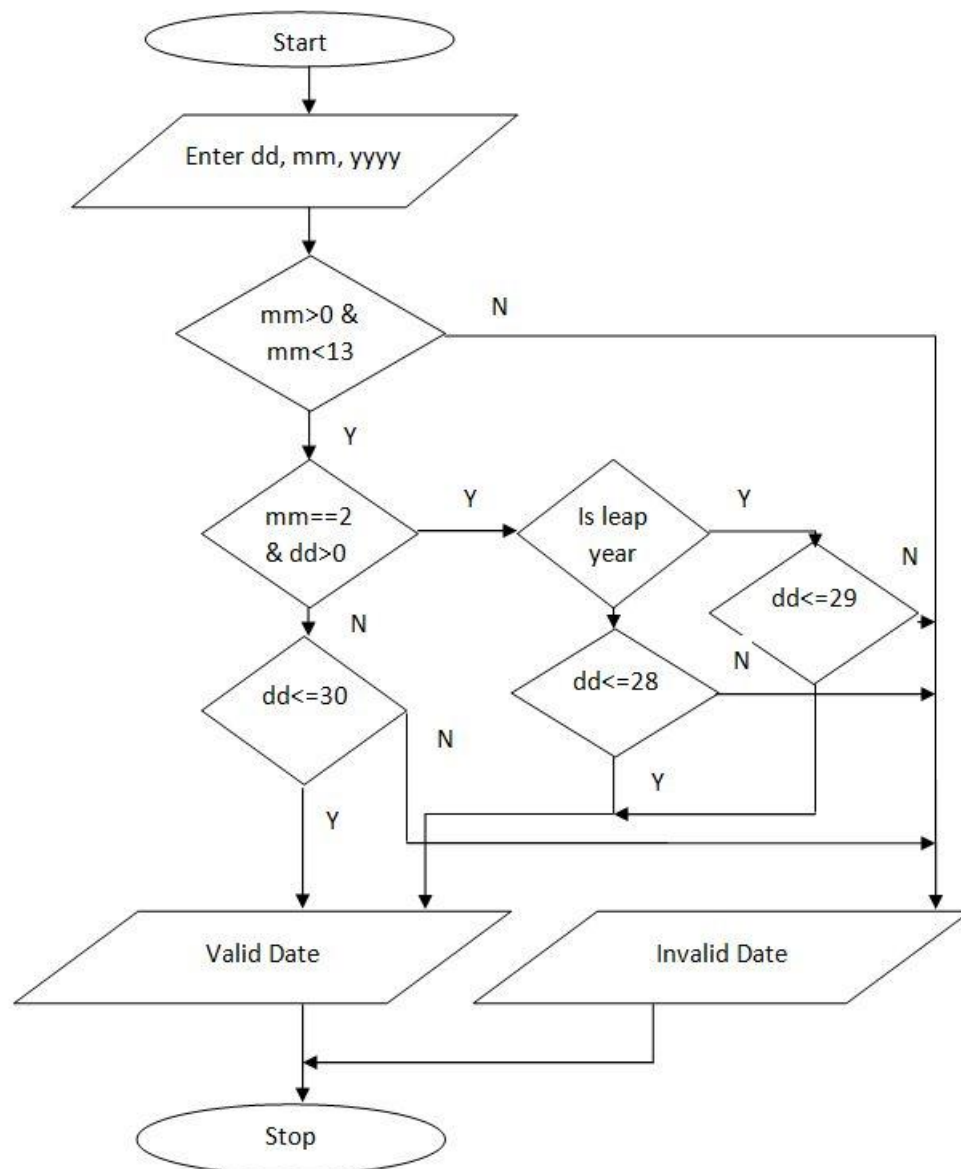
Set 1: Read a string from the user and if it is hello then print "Hello yourself", if it is bye then print "See you again" otherwise print "Sorry, I don't understand" using case statement.

Syntax: bash case statement.

case expression in

```
    pattern1 )
       statements ;;
    pattern2 )
       statements ;;
    ...
esac
```

Following are the key points of bash case statements:

- Case statement first expands the expression and tries to match it against each pattern.
- When a match is found all of the associated statements until the double semicolon (;;) are executed.
- After the first match, case terminates with the exit status of the last command that was executed.
- If there is no match, exit status of case is zero.

Sample Code:

```
read str
 case $str in
      hello)
              echo "Hello yourself!"
              ;;
      bye)
              echo "See you again!"
              ;;
      *)
              echo "Sorry, I don't understand"
              ;;
 esac
```

Input:

Hi

Output:

Sorry, I don't understand

Set 2: Check given year is leap or not.

Sample Code:

echo "Enter year "
read yy

*# **find** out **if** it is a leap year or not*
#in shell script comment is written using # symbol

if [$((yy % 4)) -ne 0]
then
echo "It is not a leap year"
elif [$((yy % 400)) -eq 0]
then
echo " It is a leap year"
elif [$((yy % 100)) -eq 0] then
echo "It is not a leap year"
else
echo " It is a leap year"
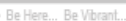fi

Input:

Enter a year 2017

Output:

2017 is not a leap year

Set 3: Write a shell script to validate the entered date. (eg. Date format is: dd-mm-yyyy)
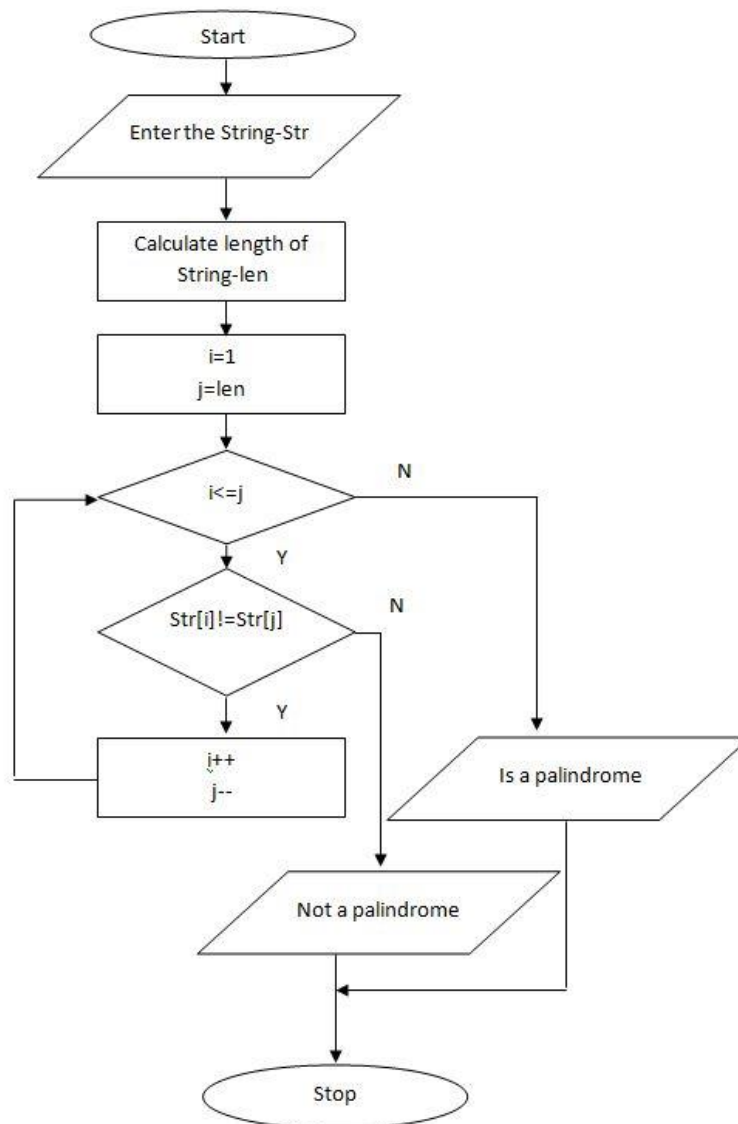
Sample Code:

Input:

Output:

PRACTICAL NO: 5

Definition: A string is said to be palindrome if reverse of the string is same as string. For example, "abba" is palindrome, but "abbc" is not palindrome. We can use Linux command to find the length of the string and then by comparing first character and last character of the string, second character and second last character of the string and so on., we can identify that whether the string is palindrome or not.

Flowchart:

Set 1: Extract the second character of the entered string.

Sample Code:

```
echo "Enter a String"
read str
```

```
k=`echo $str | cut -c 2`
echo "$k"
```

Input:

Parul

Output:

a

Set 2: Count the number of character in given string.

Sample Code:

```
clear
echo "Enter a string"
read str
len=`echo $str | wc -c`
len=`expr $len - 1`
echo "Length of the string is : $len"
```

Input:

Output:

Set 3 Write a shell script to check entered string is palindrome or not.

Sample Code:

Input:

Output:

PRACTICAL NO: 6

Definition: The date command is used to print out the system's time and date information. By extracting hours from the date using Linux 'cut' command and by using if else ladder can wish user an appropriate message like Good morning/Afternoon/Evening.
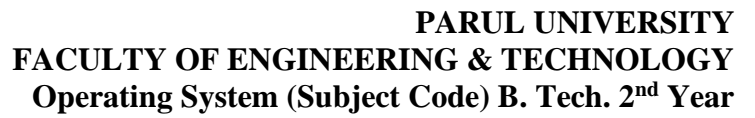
Flowchart:
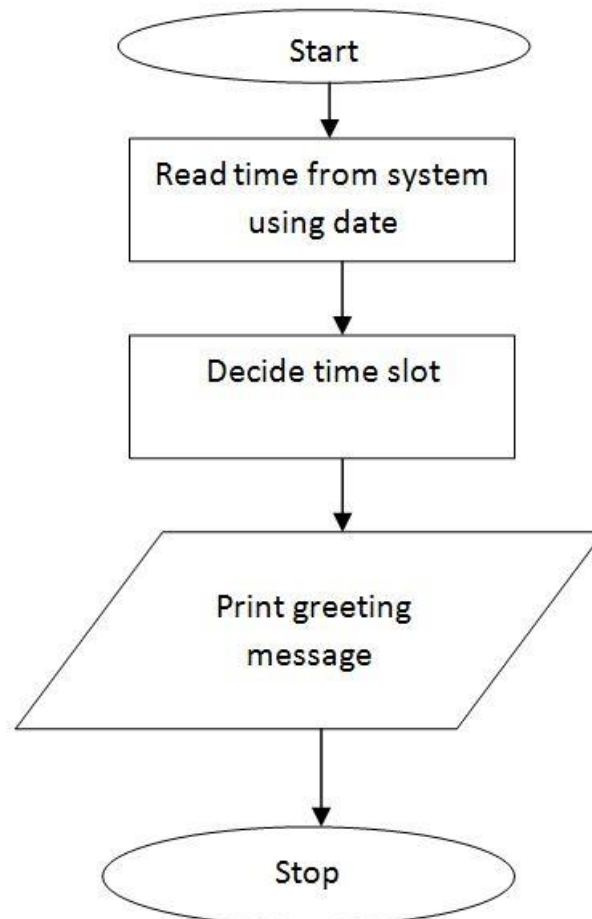
Set 1: Read the minutes from system.

Sample Code:

```
minutes=`date + M%`
echo $minutes
```

Input: -

Output:

39

Set 2: Read the hours from system using cut command of unix.

Sample Code:

```
clear
hours=`date | cut -c12-13`
echo "the value of hour is $hours"
```

Input: -

Output:

the value of hour is 4

Set 3: Write a Shell script to say Good morning/Afternoon/Evening as you log in to system.

Sample Code:

Input:

Output:

PRACTICAL NO: 7

Definition: Create child process using fork( ), which is a system call to create a child process. Use getpid( ) for getting the id of the process, getppid( ) for getting the parent process id.

Flowchart:
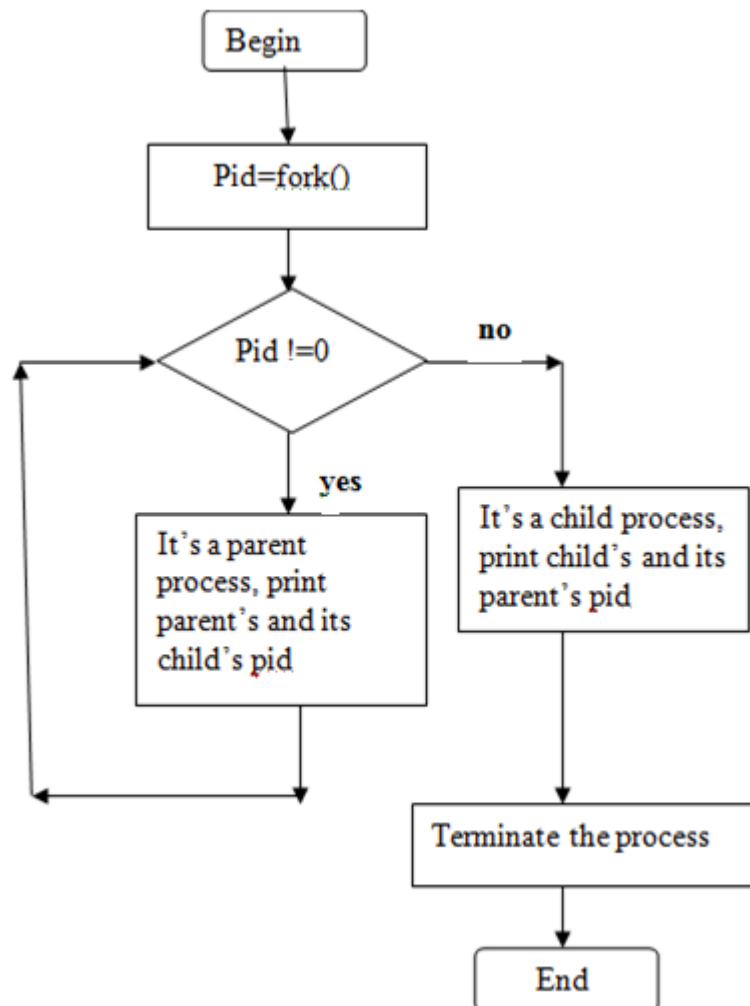
Set 1: Create a child process using one fork( ), getpid( ), getppid( ).

Sample Code:

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/wait.h>
main()
{
intpid ;
```

```
printf("I'am the original process with PID %d and PPID %d.\n", getpid(), getppid()) ;
pid = fork ( ) ; /* Duplicate. Child and parent continue from here */
if ( pid != 0 ) /* pid is non-zero,so I must be the parent*/
 {
printf("I'am the parent with PID %d and PPID %d.\n", getpid(), getppid()) ;
printf("My child's PID is %d\n", pid ) ;
 }
else /* pid is zero, so I must be the child */
 {
sleep(4); /* make sure that the parent terminates first */
printf("I'm the child with PID %d and PPID %d.\n",getpid(), getppid()) ;
 }
printf ("PID %d terminates.\n", getpid()) ;
}
```

Input: -

Output:

I'am the original process with PID 5100 and PPID 5011.
I'am the parent process with PID 5100 and PPID 5011.
My child's PID is 5101
PID 5100 terminates. /* Parent dies */
I'am the child process with PID 5101 and PPID 1.
 /* Orphaned, whose parent process is "init" with pid 1 */
PID 5101 terminates.

Set 2: Write a program to use sleep( ) function. sleep() is used to delay the output for particular time.

Input:

Output:

Set 3: Create child process using fork( ), which is a system call to create a child process.
Use getpid( ) for getting the id of the process, getppid( ) for getting the parent process id.
Creates 3 child processes using two fork().

Input:

Output:

PRACTICAL NO: 8

Definition: Finding out biggest number from given three numbers supplied as command line argument. Command argument means at run time, the user will enter the input. First input will be considered as number 1, second input will be consider as number 2 and so on. It depends on the no of command line argument.

Flowchart:
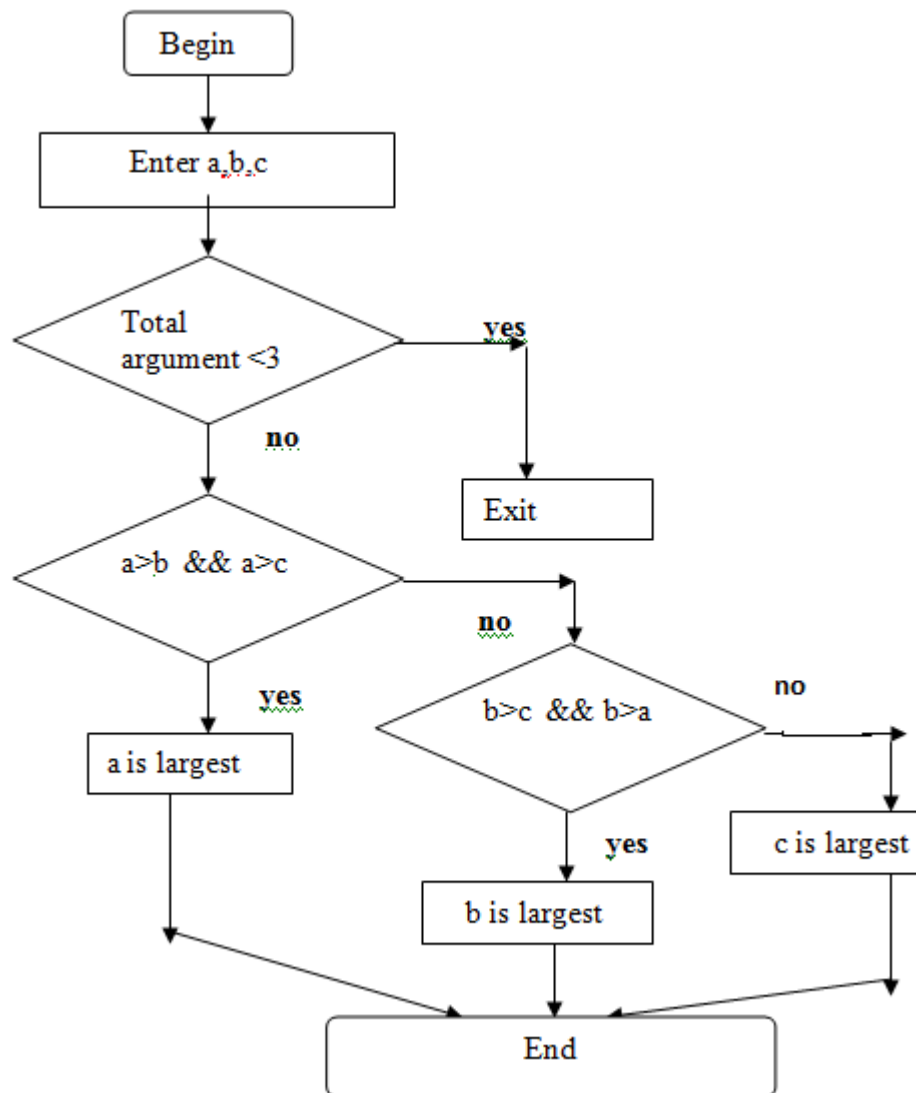
Set 1: Show the use of command line argument for finding the biggest number among three numbers using if else ladder.

Sample Code:

```
a=$1
b=$2
 c=$3
```

```
if [ $# -lt 3 ]
then
echo "Enter arguments"
exit 1
fi
if [ $a -gt $b -a $a -gt $c ]
then
echo "$a is largest integer"
elif [ $b -gt $a -a $b -gt $c ]
then
echo "$b is largest integer"
elif [ $c -gt $a -a $c -gt $b ]
then
echo "$c is largest integer"
else
echo "Sorry cannot guess number"
fi
```

Input:

5
6
7

Output:

7 is largest number

Set 2: Show the use of command line argument for finding the biggest number from two numbers using if else.

_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____

Input:

Output:

Set 3: Show the use of command line argument for finding the biggest number among three numbers using nested if else.
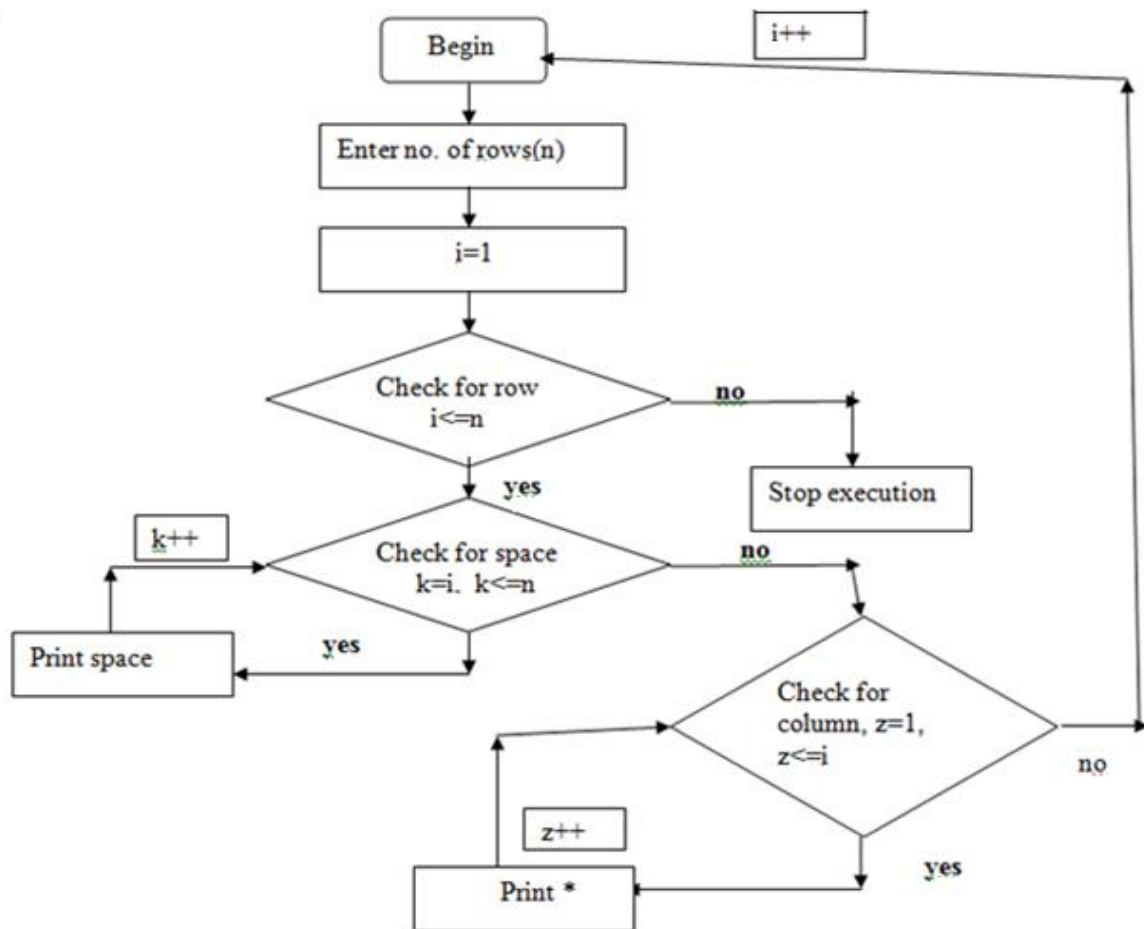
Input:

Output:

PRACTICAL NO: 9

Definition: Print the pattern using for loop which is used to do the same thing again and again until some condition is satisfied**.**

For loop is used to do the same thing until some condition is there.

Flowchart:

Set 1: Print following pattern using for loop.

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

Sample Code:

#Here $1 is the parameter you passed. It specifies the no. of rows i,e 5

```
  n=$1;
   #outer loop is for printing number of rows in the pyramid
for((i=1;i<=n;i++))
do
    #This loop print spaces required
for((k=i;k<=n;k++))
do
echo -ne " ";
done
    #This loop print part1 of the the pyramid
for((j=1;j<=i;j++))
do
echo -ne "*";
done

    #This loop print part 2 of the pryamid.
for((z=1;z<i;z++))
do
echo -ne "*";
done
    #This echo used for printing new line
echo;
done
```
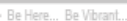
Input:

5

Output

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

Set 2: Print following pattern using for loop.

```
1
2 3
4 5 6
7 8 9 10
```

Input:

Output

Set 3: Print following pattern using for loop.

```
1
A B
2 3 4
C D E F
5 6 7 8 9
```
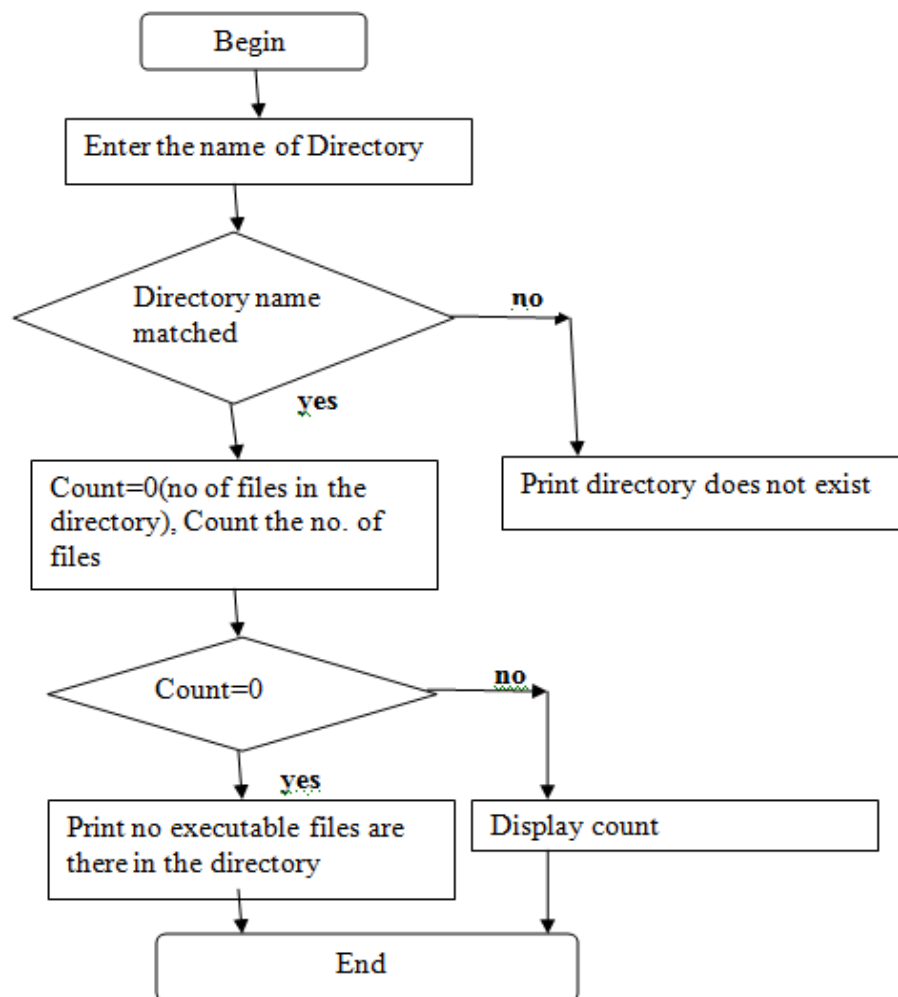
Input:

Output

PRACTICAL NO: 10

Definition: Various commands are available in Linux to check whether the given directory or file are exist or not in the system. Several options are also available to check special conditions like file is empty or not. We can use these commands in shell script as well.

Flowchart:

Set 1: Shell script to determine whether given directory, exist or not.

Sample Code:

```
clear
echo "Enter name of the directory :"
read directory
if [ ! -e $directory ]
then
```

echo "Directory not exist"
else
count=0
echo "Files with executable rights are"
for i **in** $**(find** $directory -type f -perm +111**)**
do
echo $i
count=$**(echo** count + 1 |**bc** -l**)**
done
if[$count -eq 0 **]**
then
echo "No files found with executable rights"
fi
fi

Input:

Enter name of the directory
Student

Output:

Directory not exist

Set 2: Write a shell script to check, whether the given file exist or not.

Input:

Output:

Set 3: Write a shell script to check, whether the given file, which should not be empty, exist or not.
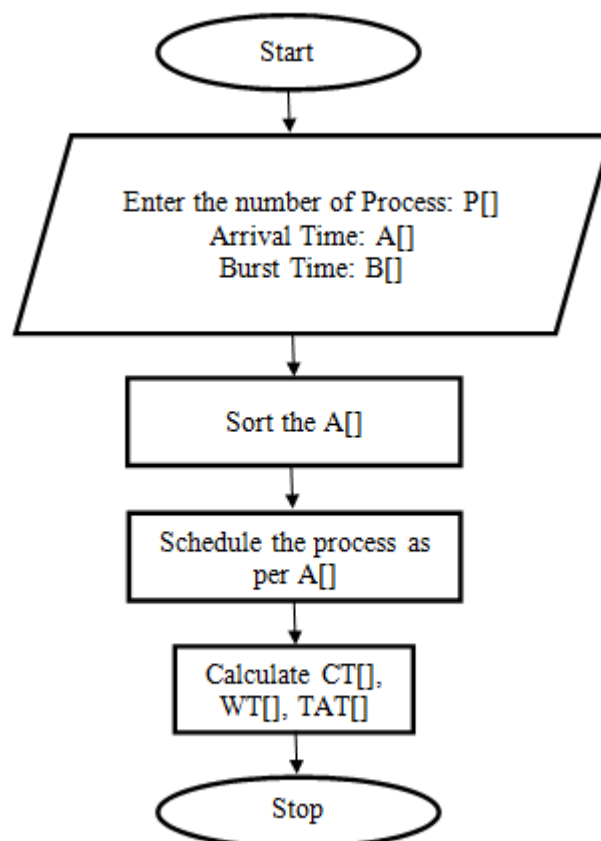
Input:

Output:

PRACTICAL NO: 11

Definition: First come, first served (FCFS) is an operating system process scheduling algorithm and a network routing management mechanism that automatically executes queued requests and processes by the order of their arrival.

Flowchart:

Set 1: How to take Arrival Time and Burst Time of the processes.

Sample Code:

```c
#include<stdio.h>
  int main()
{
 int n,count;
 int at[10],bt[10];
 printf("Enter Total Process:\t ");
 scanf("%d",&n);
 for(count=0;count<n;count++)
 {
```

```
printf("Enter Arrival Time and Burst Time for Process Number %d :",count+1);
scanf("%d",&at[count]);
scanf("%d",&bt[count]);
}
return 0;
}
```

Input:

Enter Total Process: 3

Output:

Enter Arrival Time and Burst Time for Process Number 1: 0 4
Enter Arrival Time and Burst Time for Process Number 2: 2 5
Enter Arrival Time and Burst Time for Process Number 3: 1 4

Set 2: How to sort processes based on their Arrival Time for 5 processes.

Sample Code:

```
#include<stdio.h>
void main()
{
        int i,temp,j,f,array[10];
        printf("Enter 5 Processes");
        //read array
        for(i=0;i<5;i++)
        {
                scanf("%d",&array[i]);
        }
        //bubble sort
        for(i=0;i<10;i++)
        {       f=1;
                for(j=0;j<9;j++)
                {
                        if(array[j]>array[j+1])
                        {
                                temp=array[j];
                                array[j]=array[j+1];
                                array[j+1]=temp;
                                f=0;
                        }

                }
                if(f==1)
```

```
            break;
        }
        printf("after swaping");
        for(i=0;i<5;i++)
        printf(" %d",array[i]);
}
```

Input:

Enter 5 Processes
1 5 2 8 7

Output:

1 2 5 7 8

Set 3: Calculate Turn Around Time and Waiting Time.

Input:

Output:

PRACTICAL NO: 12

Definition: Round robin (RR) scheduling is a job-scheduling algorithm that is considered to be very fair, as it uses time slices that are assigned to each process in the queue or line. Each process is then allowed to use the CPU for a given amount of time, and if it does not finish within the allotted time, it is preempted and then moved at the back of the line so that the next process in line is able to use the CPU for the same amount of time.

Flowchart:
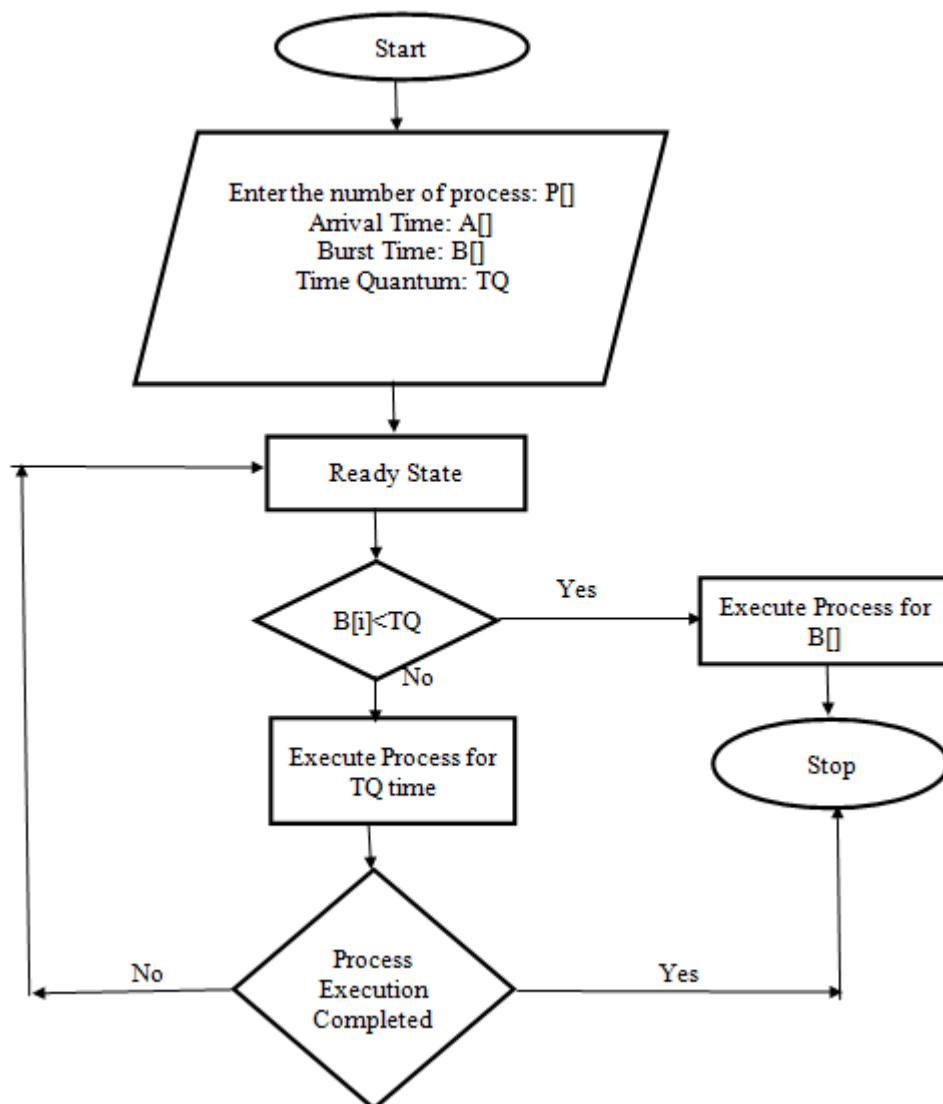
Set 1: How to take Arrival Time, Burst Time and Time Quantum of the processes.

Sample Code:

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

```
int main()
{

 int n,count, tq;
 int at[10],bt[10];
 printf("Enter Total Process:\t ");
 scanf("%d",&n);
 for(count=0;count<n;count++)
 {
  printf("Enter Arrival Time and Burst Time for Process Number %d :",count+1);
  scanf("%d",&at[count]);
  scanf("%d",&bt[count]);
 }
 printf("Enter Time Quantum:\t ");
 scanf("%d",&tq);
 return 0;
 }
```

Input:

Enter Total Process: 3

Output:

Enter Arrival Time and Burst Time for Process Number 1: 0 3
Enter Arrival Time and Burst Time for Process Number 2: 0 5
Enter Arrival Time and Burst Time for Process Number 3: 2 4
Enter Time Quantum: 3

Set 2: Calculate Turn Around Time.

Input:

Output:

Set 3: Calculate Waiting Time. Find out Average Turnaround Time and Waiting Time.

Input:

Output:

PRACTICAL NO: 13

Definition: The Banker's algorithm, sometimes referred to as the detection algorithm, is a resource allocation and deadlock avoidance algorithm. It tests for safety by simulating the allocation of predetermined maximum possible amounts of all resources. When a new process enters a system, it must declare the maximum number of instances of each resource type that it may ever claim; clearly, that number may not exceed the total number of resources in the system. Also, when a process gets all its requested resources it must return them in a finite amount of time.

Flow Chart:

Resource-Request Algorithm
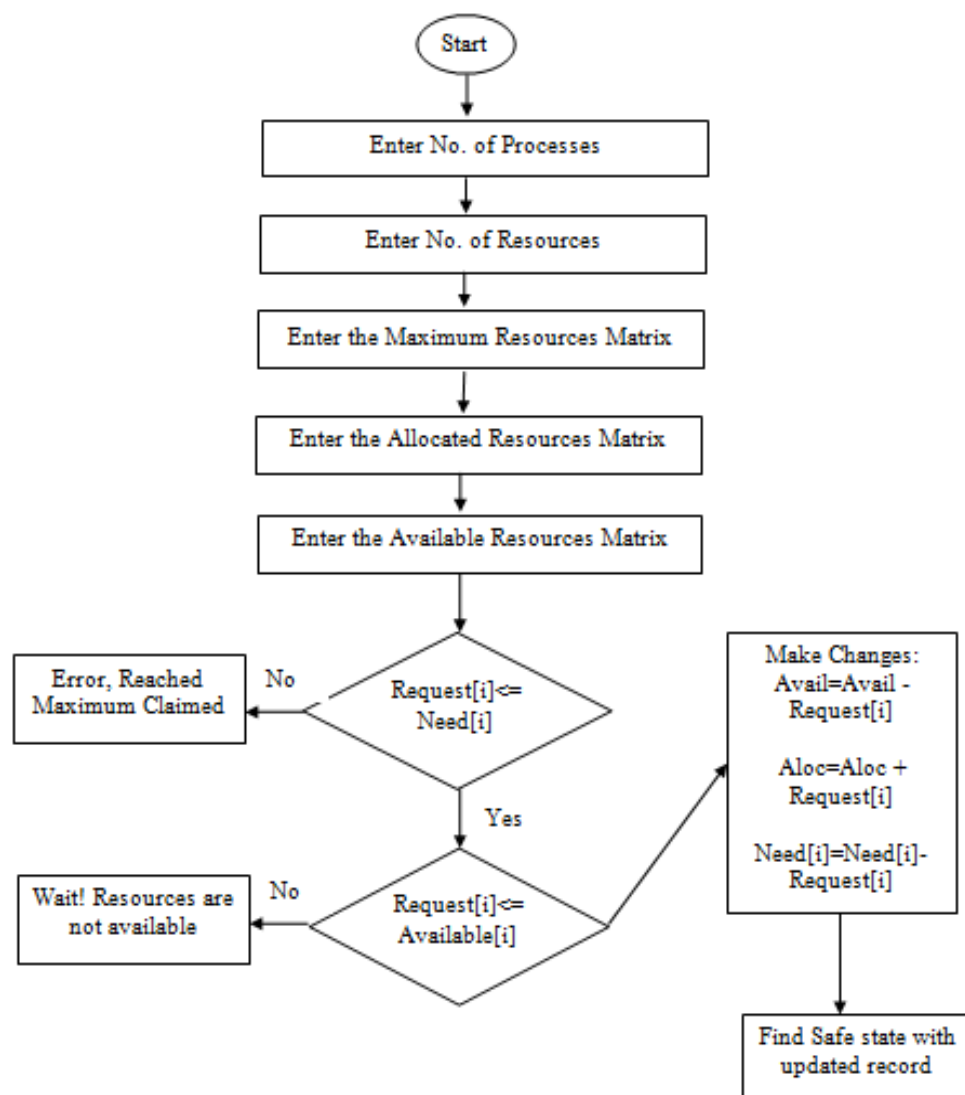
Set 1: How to take no. of processes, no. of resources, maximum resource matrix, allocated resource

matrix and available resources for each process.

Sample Code:

```c
#include <stdio.h>
#include <stdlib.h>
int main()
{
int Max[10][10], need[10][10], alloc[10][10], avail[10], completed[10];
int p, r, i, j, process, count;
count = 0;
printf("Enter the no of processes : ");
scanf("%d", &p);
for(i = 0; i< p; i++)
completed[i] = 0;
printf("\n\nEnter the no of resources : ");
scanf("%d", &r);
printf("\n\nEnter the Max Matrix for each process : ");
for(i = 0; i < p; i++)
   {
printf("\nFor process %d : ", i + 1);
for(j = 0; j < r; j++)
scanf("%d", &Max[i][j]);
   }
printf("\n\nEnter the allocation for each process : ");
for(i = 0; i < p; i++)
   {
printf("\nFor process %d : ",i + 1);
for(j = 0; j < r; j++)
scanf("%d", &alloc[i][j]);
   }
printf("\n\nEnter the Available Resources : ");
for(i = 0; i < r; i++)
scanf("%d", &avail[i]);

for(i = 0; i < p; i++)
{
for( j = 0; j < r; j++)
printf("%d ", Max[i][j]);
printf("\t\t");
for( j = 0; j < r; j++)
printf("%d ", alloc[i][j]);
printf("\n");
}

}
```

Input:

Output:

Set 2: Find the Need matrix of each process from allocated resources and Maximum resource matrix.

Input:

Output:

Set 3: Perform the Banker's Algorithm and find out that whether the System is Safe or not and also print the Safe Sequence.

Safety Algorithm:

1. Let Work and Finish be vectors of length m and n, respectively. Initially,

    Work = Available
    Finish[i] =false for i = 0, 1, ... , n - 1.

    This means, initially, no process has finished and the number of available resources is represented by the Available array.

2. Find an index i such that both

    Finish[i] ==false
    Needi <= Work

    If there is no such i present, then proceed to step 4.

    It means, we need to find an unfinished process whose need can be satisfied by the available resources. If no such process exists, just go to step 4.

3. Perform the following:

    Work = Work + Allocation;
    Finish[i] = true;

    Go to step 2.

    When an unfinished process is found, then the resources are allocated and the process is marked finished. And then, the loop is repeated to check the same for all other processes.

4. If Finish[i] == true for all i, then the system is in a safe state.

    That means if all processes are finished, then the system is in safe state.

Input:

Output: