1.a) Write a shell script that takes a valid directory name as an argument and recursively discard all the subdirectories,find the maximum length of any file in that hierarchy and write its maximum value to the standard output.

```
    clear
  if [ $# -ne 1 ]
   then
     echo "invalid number of arguments passed"
     exit
  fi

      if [ ! -d $1 ]
        then
        echo "directory not exits"
        exit
      fi

      echo "recursive listing of the $1 directory is:"
      ls -lR $1
      echo "length of the file with maximum legnth:"
ls -lR $1| tr  –s “ “|grep “^-“|cut –d “ “ –f5 | sort  –n|tail -1
```

1.b) Write a shell script that accepts a pathname and creates all the components in that pathname as directories. for example, if the script it named mpc,then the command mpc a/b/c/d should create directories a,a/b,a/b/c,a/b/c/d.

```
   clear
if [ $# -ne 1  ]
then
  echo "Invalid command line areguments"
  exit
fi

c=1
path=" "
while true
do
    dn=`echo $1 | cut -d "/" -f $c`
    if [ -z $dn ]
    then
       break
    fi
     path=$path$dn
     mkdir $path
     path=$path"/"
     c=`expr $c + 1`
 done
     echo "all directories created"
```

2.a)Write a shell script that accepts two file names as arguments, check if the permissions for these files are identical and if the permissions are identical, output common permissions and otherwise output each file names followed by its permissions.

```
clear
echo “program to print permissions of two files”
if [ $# -ne 2 ]
then
 echo “invalid number of arguments passed”
exit
fi
if [ ! –e $1 ]
then
 echo “first file not exists”
exit
fi
if [ ! –e $2 ]
then
 echo “first file not exists”
exit
fi
P1=`ls –l $1|cut –c 2-10`
P2=`ls –l $2|cut –c 2-10`
if [ $P1 = $P2 ]
then
    echo “both files have identical permissions”
    echo “ $1 permissions are:$p1”
    echo “$2 permissions are”$p2”
else
   echo “both files have not identical permissions”
    echo “ $1 permissions are:$p1”
    ep2”
fi
```

2.b)Write a shell script which accepts valid login names as arguments and print their corresponding home directories. if no arguments are specified, print a suitable error message.

```
clear
if [ $# -eq 0 ]
   then
     echo "No command line argument passed"
     exit
fi
     while [ $1 ]
      do
        cat /etc/passwd | cut -d ":" -f1 | grep "$1" > temp
        ck=`cat temp`
          if [ "$ck" != "$1" ]
             then
             echo "ERROR : $1 is an invalid Login name"
               else

             echo "Home Directory for $1 is:"
                 cat /etc/passwd | grep "$1" | cut -d ":" -f6
          fi
           shift
      done
```

3.a) Create a script file called file properties that reads a filename entered and out its properties.

```
   clear
echo "enter the filename"
read fn
if [ -f $fn ]
  then
   ls -l $fn
  else
   echo "file not found"
 fi
```

3.b)Write a shell script to implement terminal locking(similar to the lock command).It should prompt the user for a ppassword.After accepting the password enter by the user, it must be prompt again for the matching password as conformation. And it the match occurs, it must lock the keyword until a matching password is enter again by the user. Note that the script must be written to disregard break, Ctrl-D.No time limit need be implemented for the lock duration.

```
stty -echo
echo “Enter password”
read pass1
echo “Confirm password”
read pass2
if [ “$pass1” = “$pass2” ] ;

then
echo “Terminal is locked”
trap 1 2 15
while true

do
echo ” Enter password”
read pass3
if [ “$pass3” = “$pass2” ];

then
echo “Terminal Unlocked”
stty echo
exit
else
echo “Try again”
fi
done
else
echo “password do not match”
stty echo
fi
```

4.a) Write a shell script that accepts one or more file name as arguments and convert all of them to uppercase provided they exists in current directory.

```
  if [ $# -eq 0 ]
  then
    echo "invalid number of arguments"
    exit
fi

for fn in "$@"
do
 if [ -f $fn ]
 then
    echo $fn | tr '[a-z]' '[A-Z]'
 else
    echo "File not found -$fn"
 fi
done
```

4.b) Write a shell script that displays all the links to a file specified as the first argument to the script. The second argument, which is optional, can be used to specify in which the search is to begin. If the second argument is not present, the search is to begin in current working directory. In either case the starting directory as well as all its sub-directories at all levels must be searched. The script need not include any error checking.

```
   clear
if [ "$2" != " " ]
  then
    cwd=`pwd`
    cd $2
    link=`ls -l $1 | tr -s " " | cut -d " " -f2`
    cd $cwd
 else
   link=`ls -l $1 | tr -s " " | cut -d " " -f2`
 fi
echo "number of links of file $1=$link"
```

5.a)Write a shell script that accepts as filename as argument and display its creation time if the file exists and if it does not send output error message.

```
    if [ $# -ne 1 ]
   then
   echo "invalid no. of arguments"
   exit
fi
  if [ -e $1 ]
  then
   echo "file $1 is created on :`ls -l $1 | tr -s " "| cut -d " " -f 6,7,8`"
  else
   echo " file not found"
fi
```

5.b)Write a shell script to display the calendar for current month with current date replaced by * or ** depending on whether the date has one or two digit.

```
a=`date +%e`
if [ $a -lt 10 ]
then
 cal | sed "s/$a/*/"
  else
    cal | sed "s/$a/**/"
fi
```

6.a)Write a shell script to find a file/s that matches a pattern given as command line argument in the home directory, display the contents of the file and copy the file into the directory ~/mydir.

```
clear
 if [ $# -ne 1 ]
 then
    echo “invalid arguments”
    exit
fi

    dir=mydir
   if [ -d $dir ]
    then
        echo “directory $dir exists”
    else
         mkdir $dir
 fi
     echo “the file matching the pattern $1 are”
       ls *$1* >filenames.txt
        cat filenames.txt

   echo “the contents of the files are”
         for i in `cat filenames.txt`
             do
                echo “filenames:$i”
                cat $i
                cp $i ./mydir
            done
 echo “the files copied into ~/mydir”
 cd mydir
ls -l
```

6.b)Write a shell script to list all the files in a directory whose file name is atleast 10 characters(Use expr command to check the length).

```
clear
for i in `ls`
 do
   len=`expr “$i” : ‘.*’`
  if [ $len –ge 10 ]
  then
    echo $i >> outfile
    fi
done
echo “filenames having more than 10 charaacters are”
cat outfile.
```

7.a)Write a shell script that get executed display the message either “good morning” or “good afternoon” or “good evening” depending upon time at which the user logs-in.

```
     h=`who am i  | tr -s " " | cut -d " " –f4 | cut -d ":" -f1`
if [ $h -lt 12 ]
    then
       echo "good morning"
    else if [ $h -ge 12 -a $h -lt 17 ]
    then
      echo "good afternoon"
    else
      echo "good evening"
  fi
fi
```

7.b)Write a shell script that accept a list of filenames as its arguments,count and report occurance of each word that is present in the first argument file on other argument files.

```
if [ $# -ne 2 ]
 then
 echo "Invalid number of arguments"
 exit
 fi

  str=`cat $1`

  for a in $str
  do
    echo "word=$a, count=`grep -c "$a" $2`"
  done
```

8.a) Write a shell script that determine the period for which a specified user is working on system and display appropriate message.

```
clear
 echo "enter the username"
 read usr

 tm=`who | grep "$usr" | tr -s " " | head -1 | cut -d " " –f4`
 uhr=`echo $tm | cut -d ":" -f1`
 umin=`echo $tm | cut -d ":" -f2`
 shr=`date "+%H"`
 smin=`date "+%M"`
 h=`expr $shr - $uhr`
 m=`expr $smin - $umin`
if [ $m -lt 0 ]
then
    m=`expr 60 + $m`
    h=`expr $h - 1`
fi
 echo "Username:$usr"
 echo "LoginPeriod:$h:$m"
```

8. b)Write a shell script that reports the login in of a specified user with one minute after he/she log in. The script automatically terminate if specified user does not log in during a specified period of time.

```
clear
echo "Enter the login name of the user:"
read user
tm=0
while [ true ]
do
var=`who | grep "$user" | cut -d  " " -f1`
 if [ "$var" = "$user" ]
then
echo "$user logged in"
exit
else
sleep 1
tm=`expr $tm  + 1`
fi
if [ $tm -eq 61 ]
then
echo "$user did not login within 1 minute"
exit
fi
done
```

9.a) Write a shell script that accept the file name, starting and ending line number as an argument and display all the lines between the given line number.

```
if [ $# -ne 3 ]
  then
  echo "invalid number of arguments"
  exit
fi
 if [ $2 -gt $3 ]
   then
   echo "invalid range value"
   exit
 fi

la=`expr $3 - $2 + 1`
cat $1|tail  +$2|head -$la
```

9.b)Write a shell script that folds long lines into 40 columns. Thus any line that exceeds 40 characters must be broken after 40th, a “\” is to be appended as the indication of folding and the processing is to be continued with the residue. The input is to be supplied through a text file created by the user.

```
clear
echo "enter the filename"
read fn
while read ln
do
    lgth=`echo $ln | wc -c`
      s=1
      e=40
        if [ $lgth -gt 40 ]
          then
             while [ $lgth -gt 40 ]
              do
                echo "`echo $ln | cut -c $s-$e`\\"
                s=`expr $e + 1`
                e=`expr $e + 40`
                lgth=`expr $lgth - 40`

              done
               echo $ln | cut -c $s-

           else
                echo $ln
         fi
     done < $fn
```

10.a) Write an awk script that accepts date argument in the form of dd-mm-yy and displays it in the form if month,day,year.The script should check the validity of the argument and in the case of error,display a suitable message.

```
clear
BEGIN { printf "date validation" }
 {
   if (($2==2 && $1>28)||($1>31)||($2==4 || $2==6 || $2==9 || $2==11 &&
$1>30) ||($3%4!=0 && $2==2 && $1>29))
     printf ("invalid date")
else
    printf ("\n %d:%d:%d\n",$2,$1,$3)
 }
   end { "date conversion" }
```

10.b)Write an awk script to delete duplicated line from a text file.The order of the original lines must be remain unchanged.

```
clear
BEGIN { print "script to delete duplicate line:" }
         { a[n++]=$0 }
END {
       for(i=0;i<n;i++)
        {
            flag=0
            for(j=0;j<i;j++)
             {
               if(a[i]==a[j])
                {
                   flag=1
                    break
                 }
             }

               if(flag==0)
                 printf("%s\n",a[i])
         }
     }
```

11.a)Write an awk script to find out total number of books sold in each discipline as well as total book sold using associate array down table as given below.

Electrical	34
Mechanical	67
Electrical	80
Computer science	43
Mechanical	65
Civil	98
Computer science	64

```
BEGIN { print "total number of books sold in each category"}
      { books[$1]+=$2 }

END   {
        for(item in books)
        {
            printf("\t%-17s%1s%-5d\n",item,"=",books[item])
            total+=books[item]
        }


       printf("%-17s%1s%-5d\n","total books sold","=",total)
    }
```

11.b)Write an awk script to compute gross salary of an employee accordingly to rule given below

If basic salary is<10000 then HRA=15% of basic & DA=45% of basic
If basic salary is>=10000 then HRA=15% of basic & DA=45% of basic

```
BEGIN { printf "enter the basic pay:"
        getline bp <"/dev/tty"
        if(bp<10000)
         {
             hra=.15*bp
             da=.45*bp
         }
          else
            {
              hra=.2*bp
              da=.5*bp
            }
              gs=bp+hra+da
              printf "gross salary=%.2f\n",gs
       }
```