**Section I**

**Command**

1. **Shell and its types.**

The shell is a program that takes your commands from the keyboard and gives them to the operating system to perform. Different types of shells are:

* Bourne Shell(sh)
* C Shell(csh)
* TC Shell(tcsh)
* Korn Shell(ksh)
* Bourne Again Shell(bash)

1. **Usage of shell scripts**

The shell script is used to manipulate files, execute programs, and print text.

1. **Echo command with options “–n” and “–e” explained with escape sequential values.**

Syntax:

echo [OPTION]... [STRING]...

Description:

display a line of text

-n do not output the trailing newline

-e enable interpretation of the backslash-escaped characters listed below:

\NNN the character whose ASCII code is NNN (octal)

\\ backslash

\a alert (BEL)

\b backspace

\c suppress trailing newline

\f form feed

\n new line

\r carriage return

\t horizontal tab

\v vertical tab

1. **“read” command and its syntax explanation along with details about variable usage in Linux.**

Syntax:

read variable

Description:

The read command is useful in scripts when reading or asking an input from user.

Variables can be used in two ways in Linux:

* + variable1=value
  + Read variable1,variable2

1. **Usage of variable names in “echo” command with double, single and back quotes.**
   * Double quotes: Anything enclose in double quotes removed meaning of that characters(except \ and $)
   * Single quotes: Enclosed in single quotes remains unchanged
   * Back quotes: To execute command
2. **Conditional structure “case” and looping structures “while” syntax explanation.** 
   * Case:

Syntax:

case$variable in

option)statement;;

\*)statement3;;

esac

Description:

Evaluate one of the several scripts, depending upon a given value

* + While

Syntax:

while [ condition ]

do

Statement

done

Description:

Execute scripts repeatedly as long as a condition is met

1. **Methods to calculate arithmetic expression.**

We can calculate arithmetic expression in two ways:

* + Using expr command

expr $variable1 operation $variable2

* + Without using expr command

$((variable1 operation variable2))

1. **“vi” editor usage, modes and key combinations used for inserting text, save and save as the contents, copy and paste text, delete lines, go to beginning of line, end of line, top of the file and bottom of the file.**

vi editor is used to edit the contents of the file. It has two modes: Command Mode and Insert Mode. The key combinations are as follows:

* + insert text: i or a
  + save: :w
  + save as: :new\_filename
  + copy: :y
  + paste: :p
  + delete: :dd
  + go to beginning: ^
  + end of line: $
  + top of file: H
  + bottom of file: L

**Lab Questions**

1. **Write a script to accept name, subject name and marks of three subjects. Print the result details as name, subject name, marks, total, average and grade. Use the following criteria for grade calculation.**

**Average is > 75, grade “A”; Between 50 and 74, grade “B” ; Average < 50, grade “C”**

* **Coding**:

echo "Enter name of the student"

read name

echo "Enter the marks scored in English:"

read eng

echo "Enter the marks scored in Maths:"

read maths

echo "Enter the marks scored in Science:"

read sci

echo –e "Name of the student: $name\nMarks:\nEnglish: $eng\nMaths: $maths\nScience: $sci"

echo "Total: `expr $eng + $maths + $sci`\300"

total=$((eng+maths+sci))

avg=$((total/3))

echo "average $avg"

if [ $avg -gt 75 ];

then

echo "A"

elif [ $avg -lt 74 -a $avg -ge 50 ];

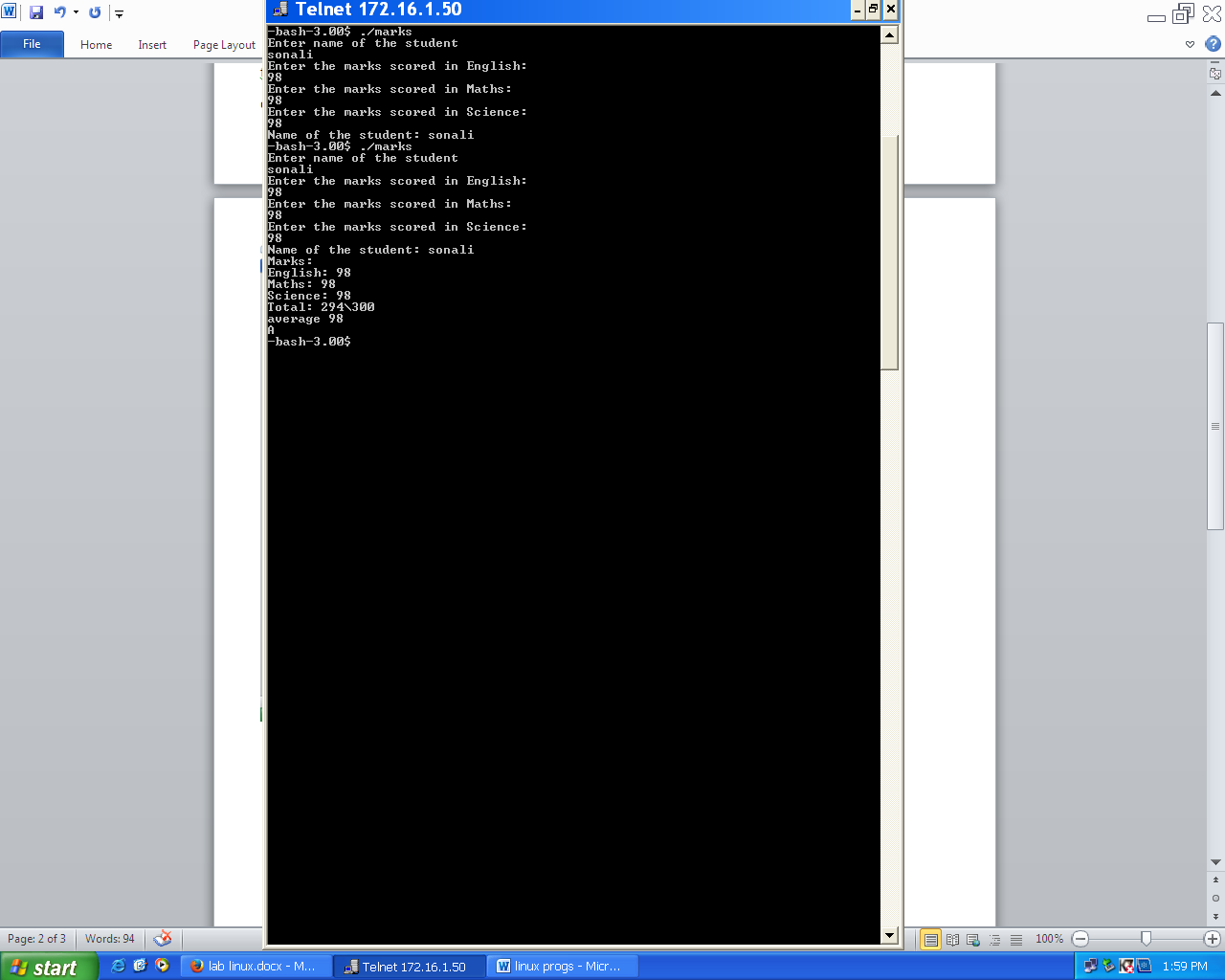
then

echo "B"

else

echo "C"

fi

* **Output:**

1. **Write a shell script to display menu for addition, subtraction, multiplication, division and exit. Accept choice from the user. If the choice is not “exit”, then accept two numbersand perform selected operation and print the result. Repeat the process until the user selects “exit” option.**
   * **Coding:**

cho=0

while [ $cho -lt 5 ]

do

echo -e "Enter Choice:\n1. Addition\n2. Subtraction\n3. Multiplication\n4. Division\n5. Exit"

read cho

if [ $cho -lt 5 ]; then

echo -n "Enter first number"

read frst

echo -n "Enter second number"

read scnd

case $cho in

1)echo "Addition of $frst + $scnd is `expr $frst + $scnd`";;

2)echo "Subtraction of $frst - $scnd is `expr $frst - $scnd`";;

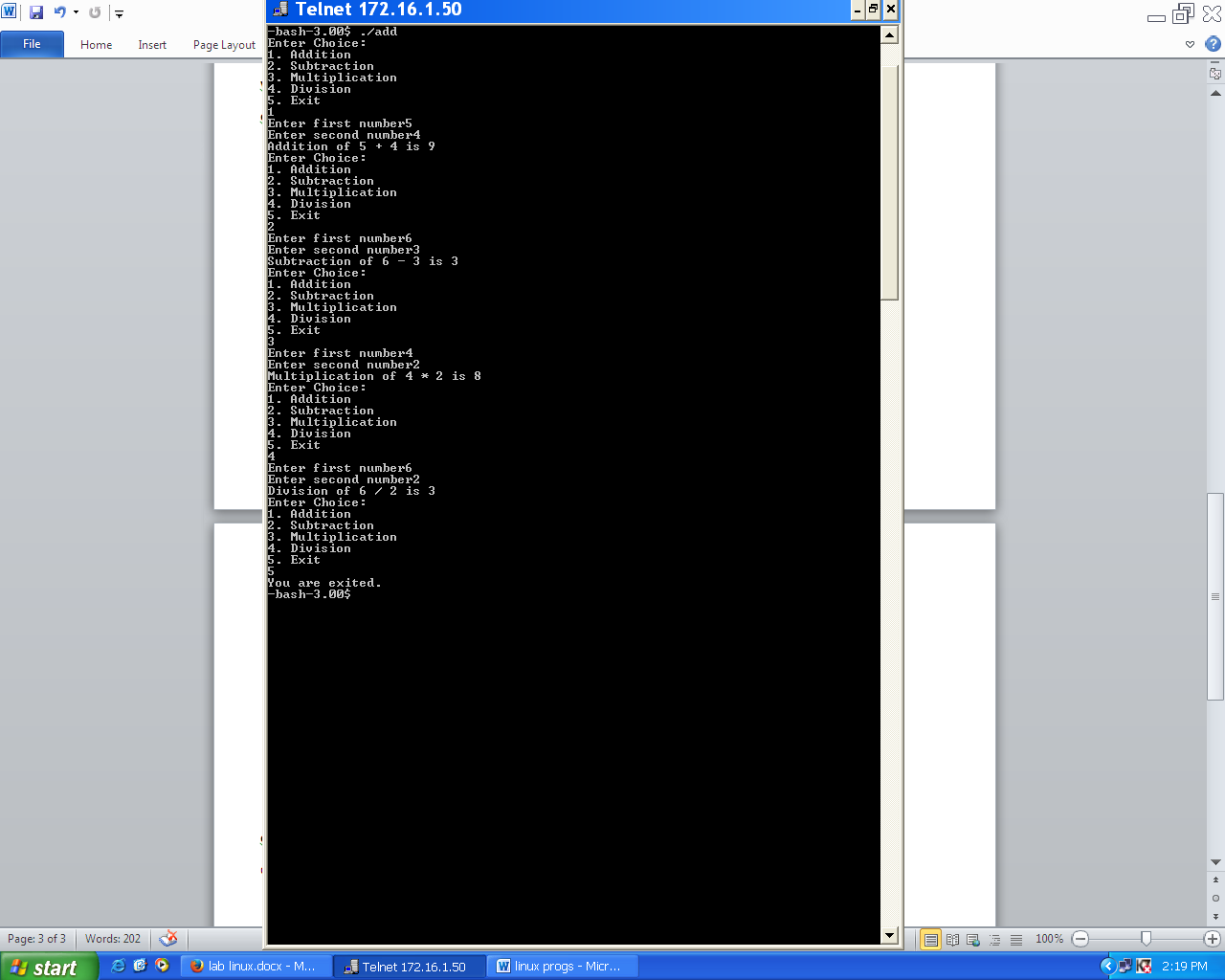
3)echo "Multiplication of $frst \* $scnd is `expr $frst\\* $scnd`";;

4)echo "Division of $frst / $scnd is `expr $frst / $scnd`";;

esac

fi

done

* + **Output:**

1. **Write a script to print the pattern given below**

**\***

**\* \***

**\* \* \***

**\* \* \* \***

* + **Coding:**

i=0

while [ $i -lt 5 ]

do

j=0

while [ $j -lt $i ]

do

echo -n "\*"

echo -n " "

j=$((j+1))

done

echo " "

k=0

while [ $k -lt $((5-i)) ]

do

echo -n " "

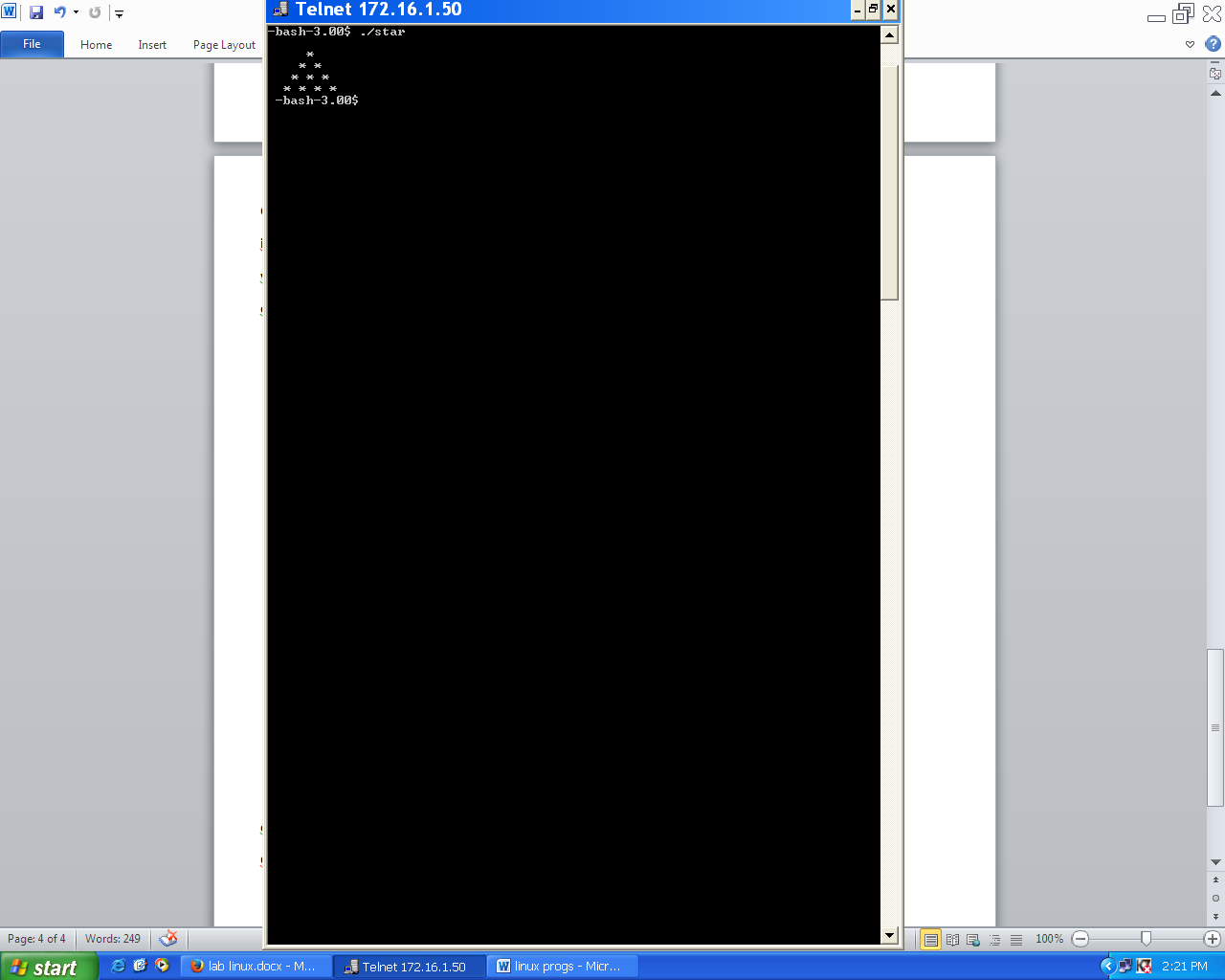
k=$((k+1))

done

i=$((i+1))

done

* + **Output:**

****

1. **Write a script to print the pattern given below**

**\* \* \* \***

**\* \* \***

**\* \***

**\***

* + **Coding:**

i=4

while [ $i -gt 0 ]

do

j=$i

while [ $j -gt 0 ]

do

echo -n "\*"

echo -n " "

j=$((j-1))

done

echo " "

k=4

while [ $k -gt $((i-1)) ]

do

echo -n " "

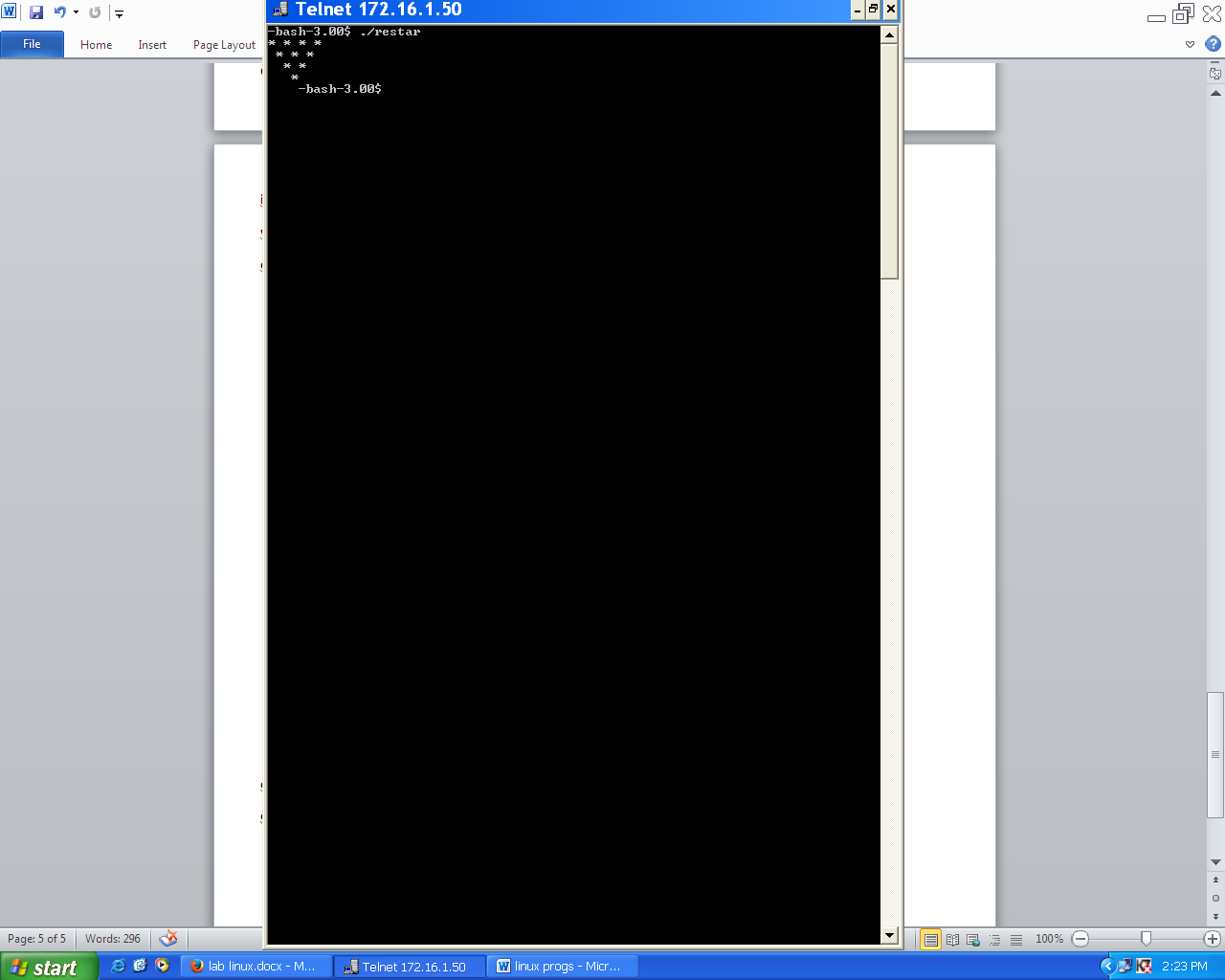
k=$((k-1))

done

i=$((i-1))

done

* + **Output:**

****

1. **Write a script to print the pattern given below**

**1**

**1 2 1**

**1 2 3 2 1**

**1 2 3 4 3 2 1**

* + **Coding**:

i=0

while [ $i -lt 5 ]

do

j=0

while [ $j -lt $i ]

do

echo -n "$((j+1))"

j=$((j+1))

done

j=$((j-1))

while [ $j -gt 0 ]

do

echo -n "$j"

j=$((j-1))

done

echo " "

k=0

while [ $k -lt $((5-i)) ]

do

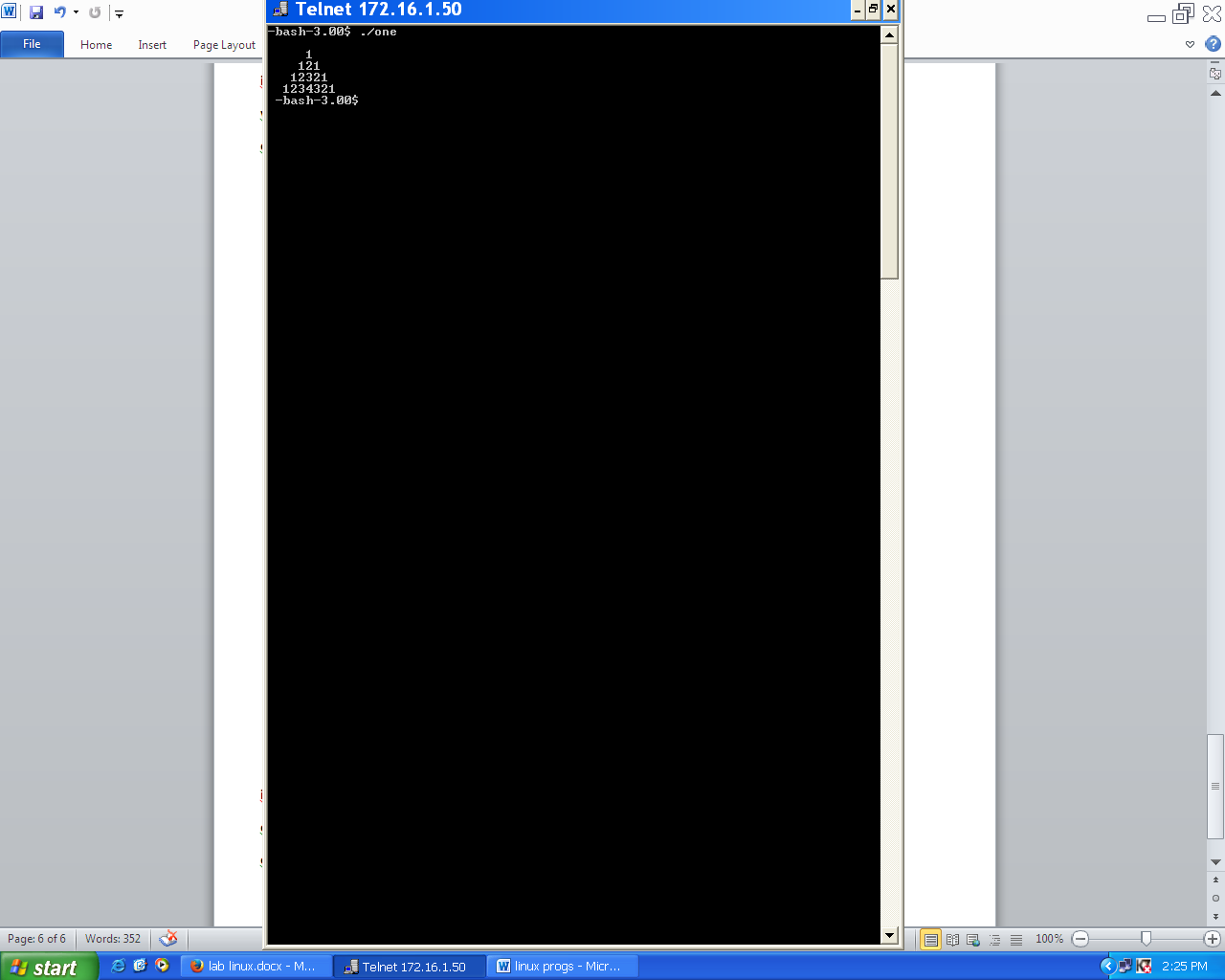
echo -n " "

k=$((k+1))

done

i=$((i+1))

done

* + **Output:**

1. **Write a script to print the pattern given below**

**\***

**\* \***

**\* \* \***

**\* \* \* \***

**\* \* \***

**\* \***

**\***

* + **Coding:**

i=0

while [ $i -lt 5 ]

do

j=0

while [ $j -lt $i ]

do

echo -n "\* "

j=$((j+1))

done

echo " "

k=0

while [ $k -lt $((5-i)) ]

do

echo -n " "

k=$((k+1))

done

i=$((i+1))

done

i=3

while [ $i -gt 0 ]

do

echo -n " "

j=$i

while [ $j -gt 0 ]

do

echo -n "\* "

j=$((j-1))

done

echo " "

k=4

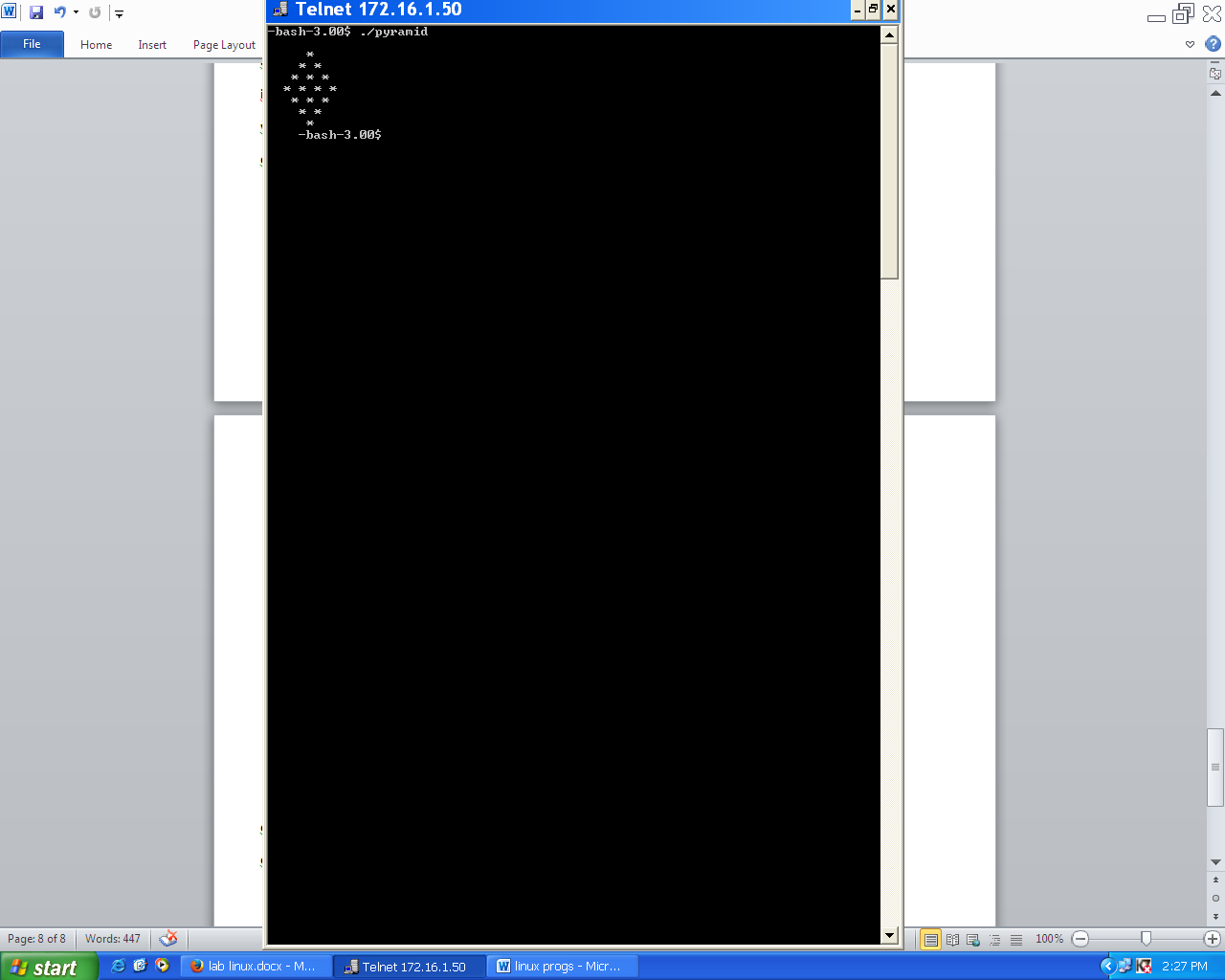
while [ $k -gt $((i-1)) ]

do

echo -n " "

k=$((k-1))

done

i=$((i-1))

done

* + **Output:**

1. **Write a program to print the sum of accepted numbers? Amount of number to be accepted has to be given by the user.**
   * **Coding:**

echo "Enter the amount of numbers"

read n

sum=0

while [ $n -gt 0 ]

do

echo "Enter the number."

read a

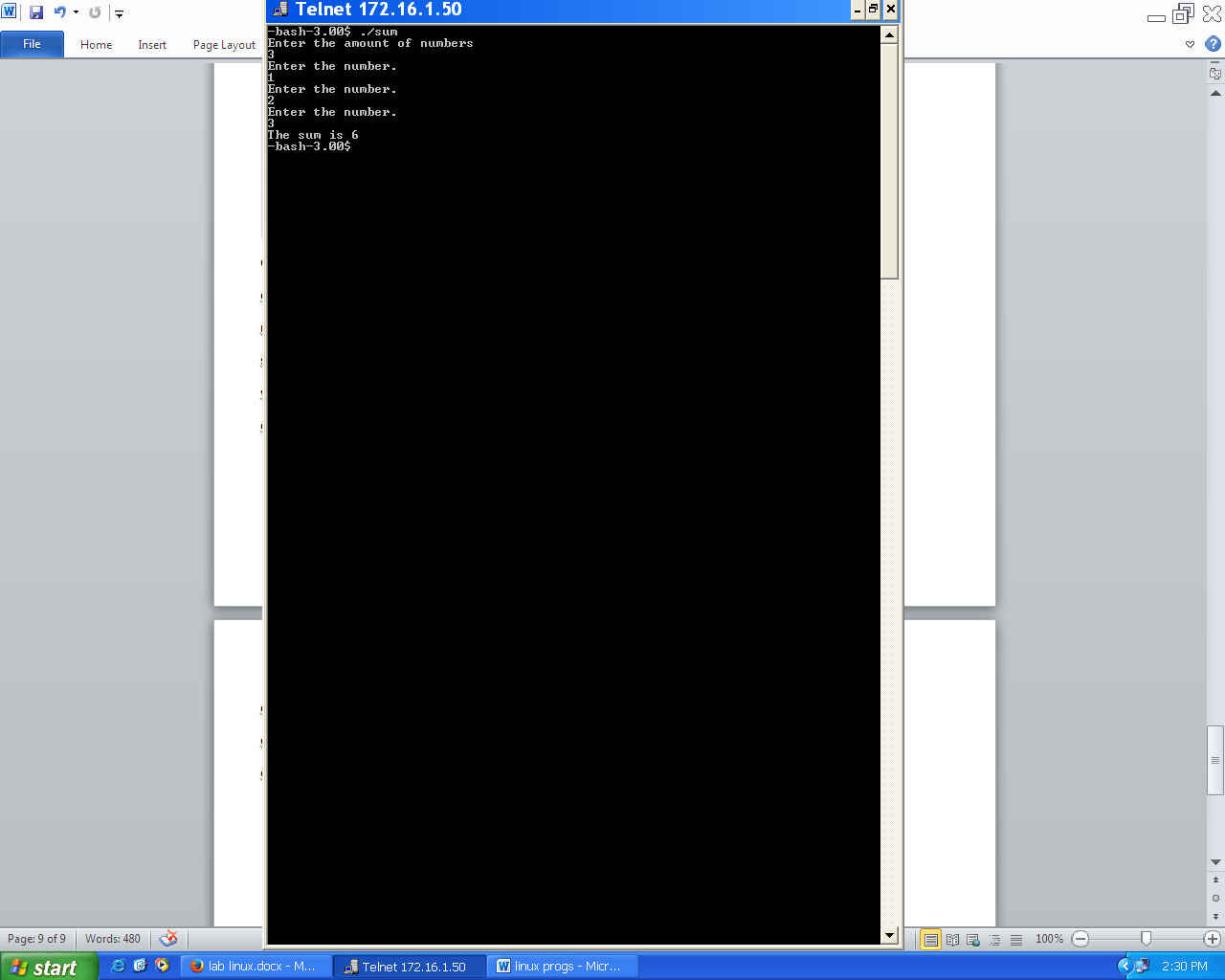
sum=$((sum+a))

n=$((n-1))

done

echo "The sum is $sum"

* + **Output:**



1. **Accept a number (multiplicand and multiplier) and print its multiplication table in proper format.**
   * **Coding**:

echo "Enter the number to print its table"

read n

echo "Enter limit of table"

read m

i=1

while [ $i -le $m ]

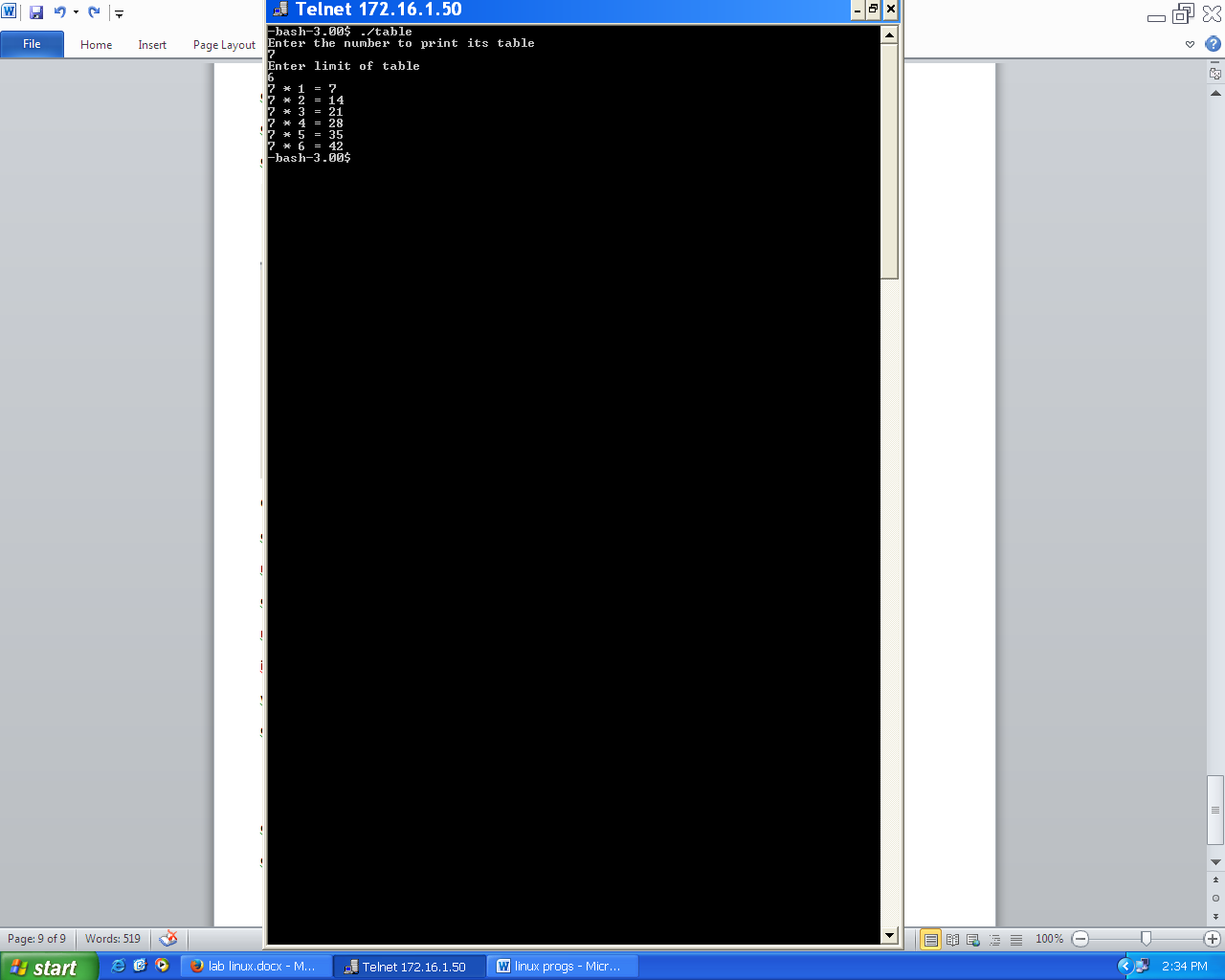
do

echo "$n \* $i = `expr $i \\* $n`"

i=$((i+1))

done

* + **Output:**



**Section II**

**Command**

1. **Date command with percentage formats explanation for a,A,b,B,d,D,g,G,j,k,l,m,M,p,P,r,R,S,T,u,U.**

Syntax:

date [OPTION]... [+FORMAT]

Description:

printor set the system date and time

%a abbreviated weekday name (Sun..Sat)

%A full weekday name, variable length (Sunday..Saturday)

%b abbreviated month name (Jan..Dec)

%B full month name, variable length (January..December)

%d day of month (01..31)

%D date (mm/dd/yy)

%g the 2-digit year corresponding to the %V week number

%G the 4-digit year corresponding to the %V week number

%j day of year (001..366)

%k hour ( 0..23)

%l hour ( 1..12)

%m month (01..12)

%M minute (00..59)

%p upper case AM or PM indicator (blank in many locales)

%P lower case am or pm indicator (blank in many locales)

%r time, 12-hour (hh:mm:ss [AP]M)

%p upper case AM or PM indicator (blank in many locales)

%P lower case am or pm indicator (blank in many locales)

%r time, 12-hour (hh:mm:ss [AP]M)

%R time, 24-hour (hh:mm)

%S second (00..60); the 60 is necessary to accommodate a leap sec-ond

%T time, 24-hour (hh:mm:ss)

%u day of week (1..7); 1 represents Monday

%U week number of year with Sunday as first day of week (00..53)

1. **“ls” command with options a, c, g, I, l, m, r, S, t.**

Syntax:

ls [OPTION]... [FILE]...

Description:

list directory contents

-a: do not hide entries starting with.

-c: sort by, and show, ctime (time of last modification of file status information) with -l: show ctime and sort by name otherwise: sort by ctime

-g : like -l, but do not list owner

-i, --inode: print index number of each file

-l: use a long listing format

-m fill width with a comma separated list of entries

-r, --reverse: reverse order while sorting

-S: sort by file size

-t: sort by modification time

1. **Conditional statements to check files and directories to be explained.**
   * For files:

if test –f file

* + For directories:

if test –d directory

1. **“cat” command to create a new file from existing and also from keyboard.**
   * From Existing file

cat existing\_file > new\_file

* + From keyboard

cat

(write in file)

1. **“mkdir”, “cp”, “mv”, “rm”, “rmdir”, “wc”, “cal” and “touch” command usage and syntax.**
   * Mkdir

Syntax:

mkdir [OPTION] DIRECTORY...

Usage:

Create the DIRECTORY(ies), if they do not already exist.

* + Cp

Syntax:

cp [OPTION]…SOURCE…DIRECTORY…

Description:

Copy SOURCE to DEST, or multiple SOURCES(s) to DIRECTORY.

* + Mv

Syntax:

mv [option]…SOURCE…DIRECTORY

Usage:

Rename SOURCE to DEST, or move SOURCE(s) to DIRECTORY.

* + Rm

Syntax:

rm [OPTION]... FILE...

Usage:

rm removes each specified file.

* + Rmdir

Syntax:

rmdir [OPTION]... DIRECTORY...

Usage:

Remove the DIRECTORY (ies), if they are empty.

* + Wc

Syntax:

wc [OPTION]... [FILE]...

Usage:

Print newline, word, and byte counts for each FILE and a total line if more than one FILE is specified.

* + Cal

Syntax:

cal [-smjy13] [[month] year]

Usage:

Cal displays a simple calendar.

* + Touch

Syntax:

touch [OPTION]... FILE...

Usage:

Update the access and modification times of each FILE to the current time.

1. **“cd” command with absolute and relative addressing methods.**

Syntax:

cd [directory]

Usage:

Changes the directory

* + Absolute Addressing Mode:

cd /etc/X11

* + Relative Addressing Mode:

cd ../../etc/X11

**Lab Questions**

1. **Write a shell script to accept name and greet the person depending on the system time.**
   * **Coding:**

echo "Enter Name."

read name

i=`date +%H`

if [ $i -lt 12 ]

then

echo -n "Good Morning"

elif [ $i -lt 16 -a $i -ge 12 ]

then

echo -n "Good Afternoon"

elif [ $i -lt 21 -a $i -ge 16 ]

then

echo -n "Good Evening"

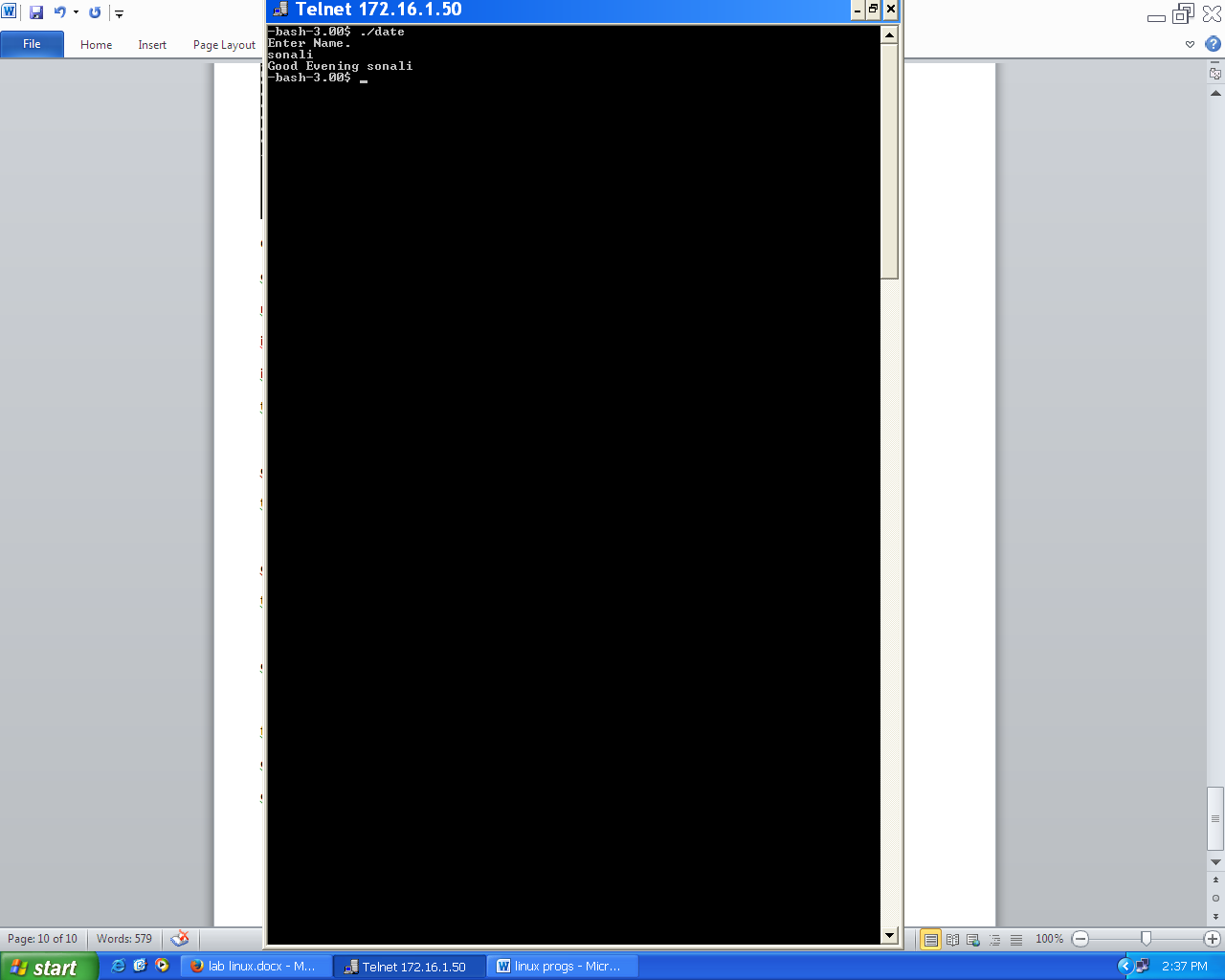
else

echo -n "Good Night"

fi

echo " $name"

* + **Output:**



1. **Write a script to accept a file name, check for file existence and list down contents of file along with the details of number of lines, characters and words.**
   * **Coding:**

echo "Enter the file name"

read file

if test -f $file

then

cat $file

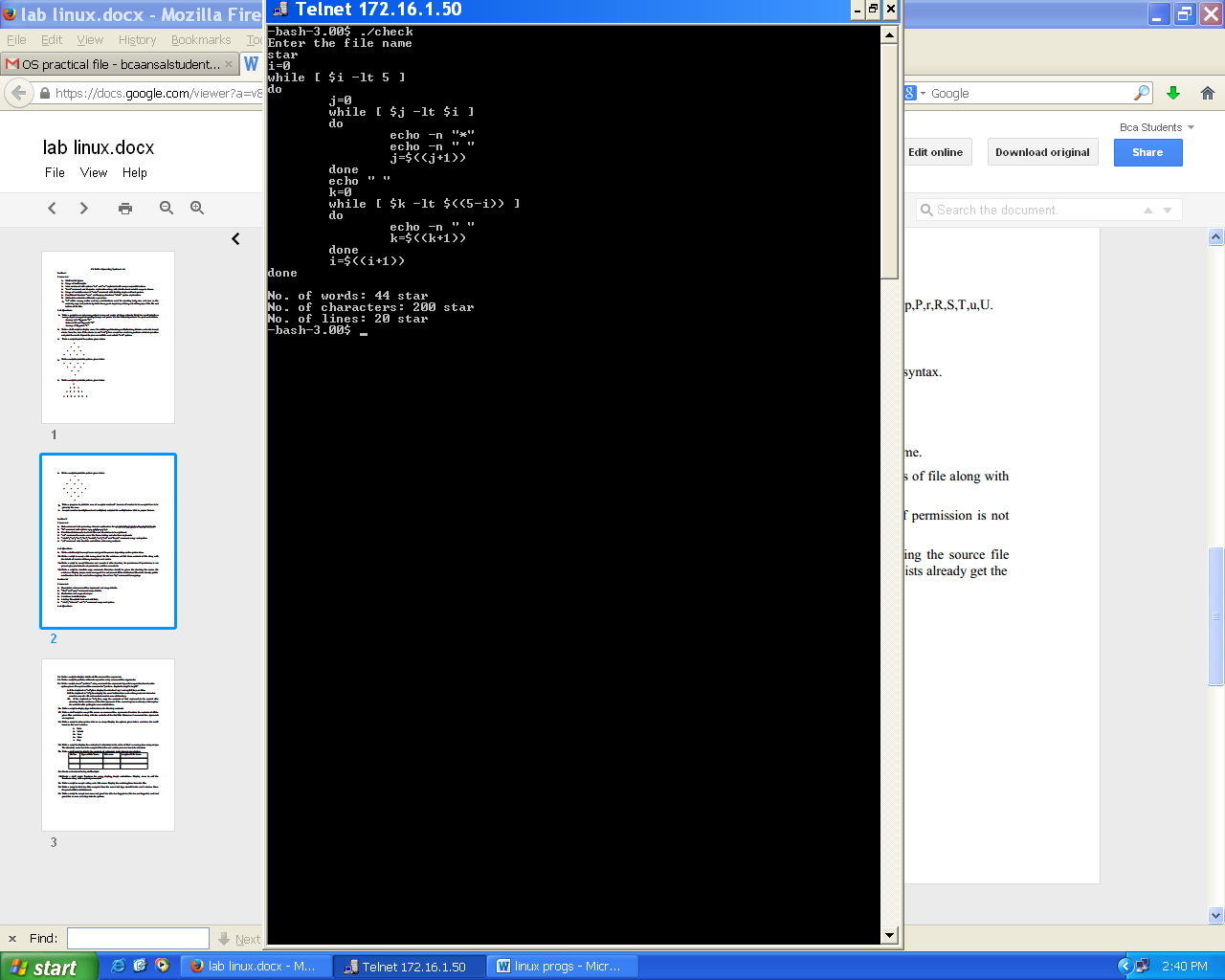
echo "No. of words: `wc -w $file`"

echo "No. of characters: `wc -m $file`"

echo "No. of lines: `wc -l $file`"

fi

* + **Output:**



1. **Write a script to accept filename and execute it after checking for permission. If permission is not present give provision to set permission and then execute it.**
   * **Coding:**

echo "Enter the file name"

read file

if test -x $file

then

./$file

else

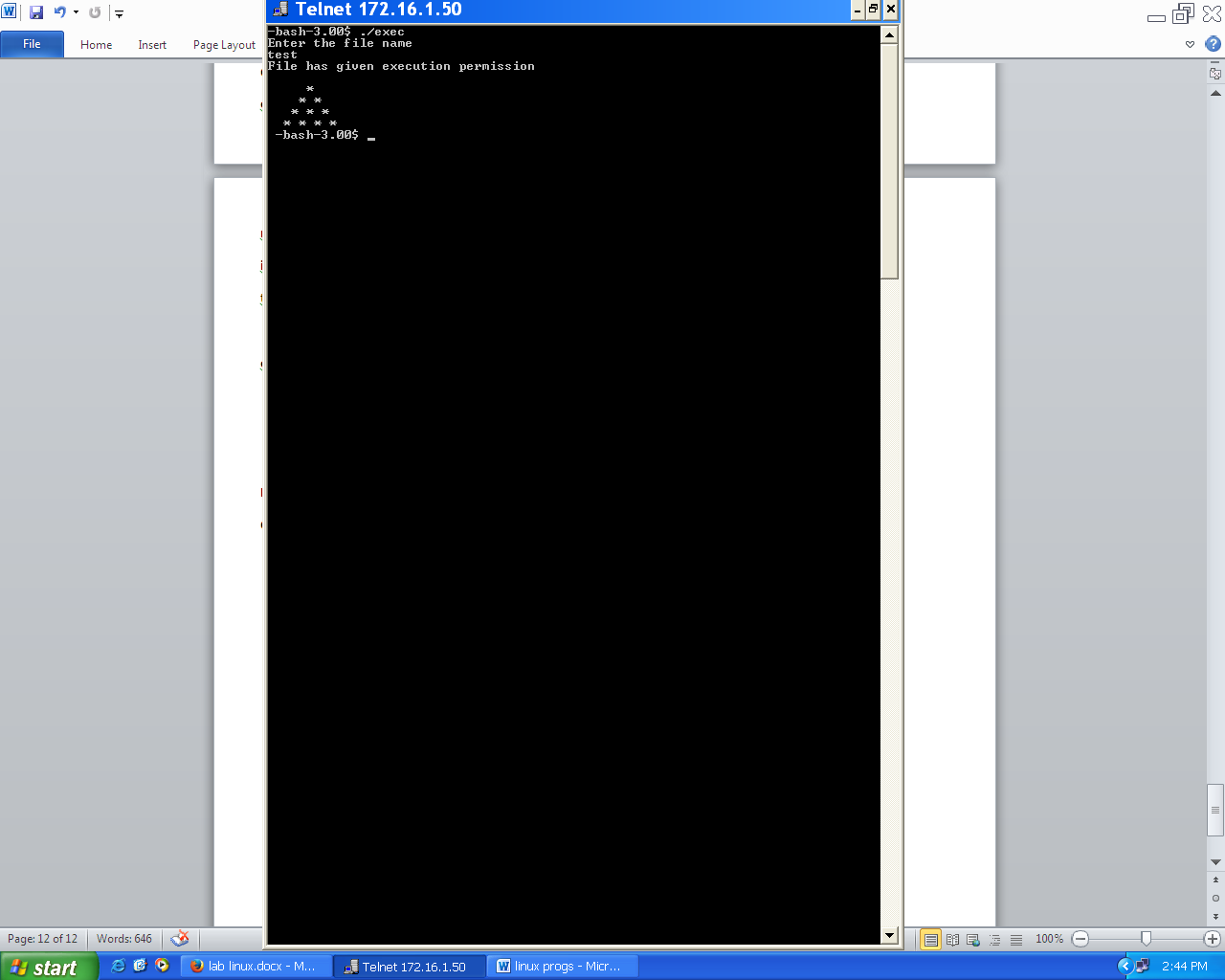
chmod +x $file

echo "File has given execution permission"

./$file

fi

* + **Output:**



1. **Write a script to simulate copy command. Provision should be given for checking the source file existence. Display proper error message if it is not present. If the destination file exists already get the confirmation from the user before copying. Do not use “cp” command for copying.**
   * **Coding:**

echo "Enter source file"

read file

if test -f $file

then

echo "Enter destination file."

read file1

if test -f $file1

then

echo "File Already Exists."

echo -e "Do you want to copy it?\n Enter y for yes and n for no"

readch

if test $ch == 'y'

then

cat $file > $file1

echo “Successfully Copied.”

else

echo -e "Content Not Copied.\nYou are existed."

fi

else

cat $file > $file1

echo “Successfully Copied.”

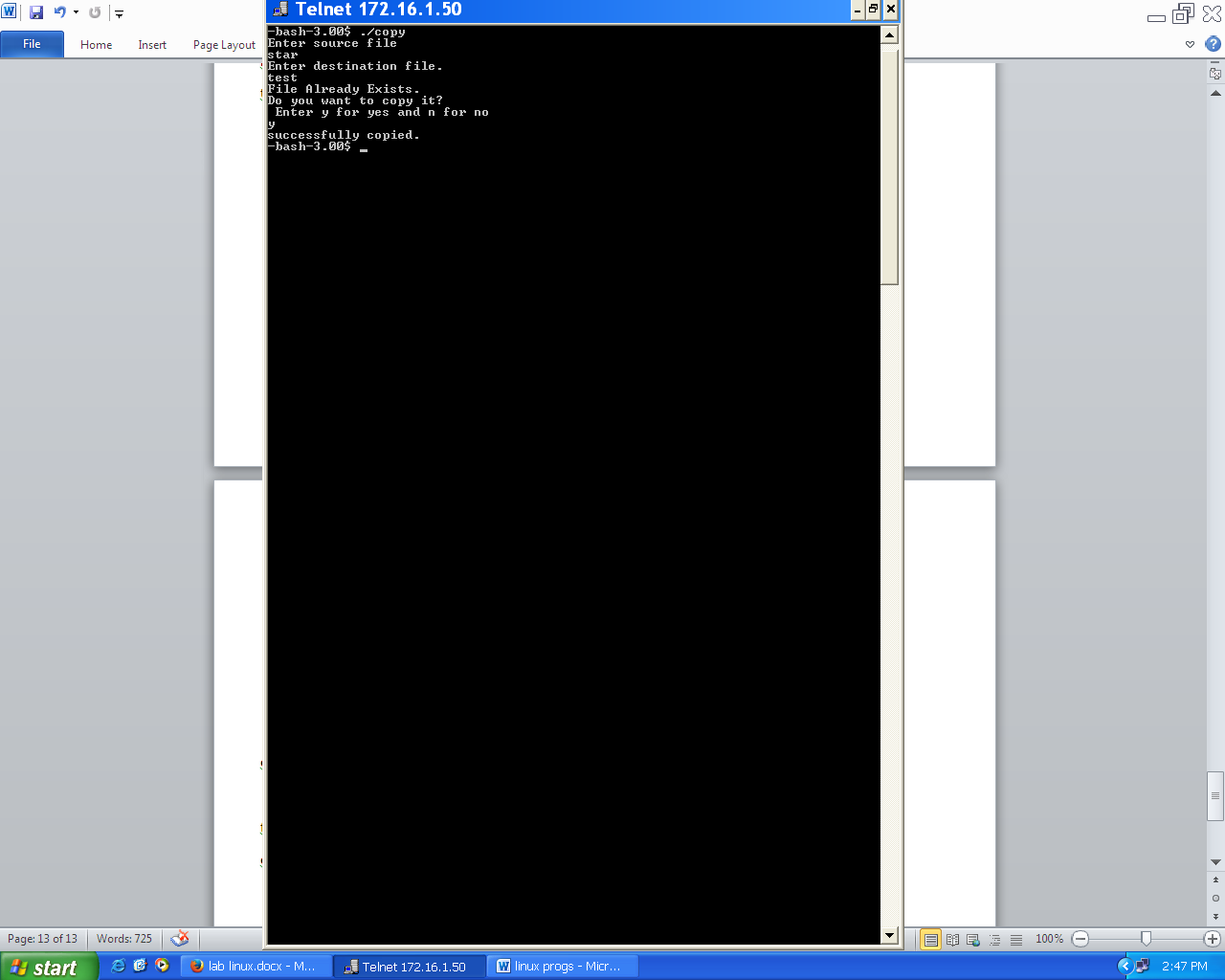
fi

else

echo "File Does not Exists"

fi

* + **Output:**



**Section III**

**Command**

1. **Description of command line arguments and usage details.**

An argument, also called a command line argument, is a file name or other data that is provided to a command in order for the command to use it as an input.

1. **“find” and “grep” command usage details.**
   * Find

Syntax:

find [path...] [expression]

Usage:

Search for files in a directory hierarchy

* + Grep

Syntax:

grep [options] PATTERN [FILE...]

Usage:

Print lines matching a pattern

1. **Declaration and usage of arrays.**
   * Declaration:
     1. ARRAY[INDEXNUMBER]=value
     2. ARRAY=(value1 value2 ... valueN)
   * Usage:

echo ${ARRAY[INDEXNUMBER]}

1. **Functions in shell scripts.**

Shell functions are sets of commands are grouped together and jointly called by a single name. The format for function is:

function\_name()

{ statements }

1. **Linking files. (both hard and soft link)**
   * Symbolic links refer to a symbolic path indicating the abstract location of another file.

ln -s $file1 $file2

* + Hard links refer to the specific location of physical data.

ln $file1 $file2

1. **“who”, “whoami” and “w” command usage and syntax.**

* who

Syntax: who [*OPTION*]... [ *FILE | ARG1 ARG2*]

Usage: Show who is logged on

* whoami

Syntax: whoami [*OPTION*]...

Usage: Print the user name associated with the current effective user id.

* w

Syntax: w [*user*]

Usage: Show who is logged on and what they are doing

**Lab Questions**

1. **Write a script to display details of all command line arguments.**
   * **Coding:**

echo "The arguments are $\*"

echo "The arguments count is $#"

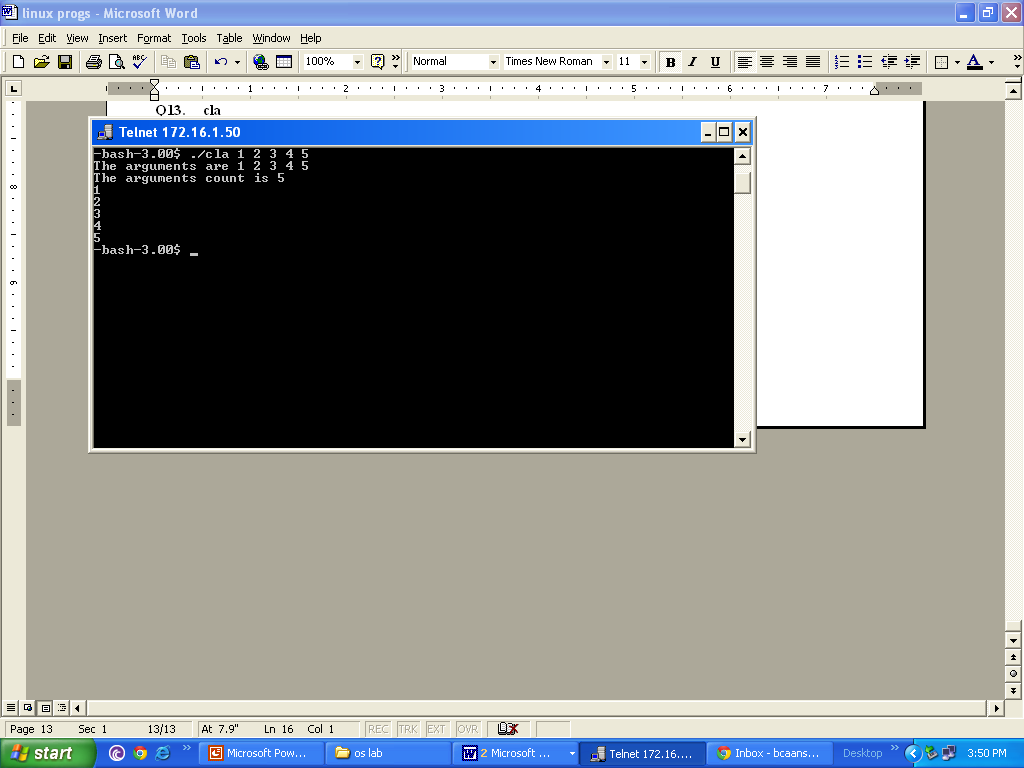
fori in $\*

do

echo $i

done

* + **Output:**

****

1. **Write a script to perform arithmetic operation using command line arguments.** 
   * **Coding:**

if [ $# -eq 3 ]

then

case $3 in

+) echo "Sum: `expr $1 + $2`";;

-) echo "Difference: `expr $1 - $2`";;

\\*) echo "Product: `expr $1 \\* $2`";;

/) echo "Quotient: `expr $1 / $2`";;

%) echo "Remainder: `expr $1 % $2`";;

\*) echo "Invalid operator";;

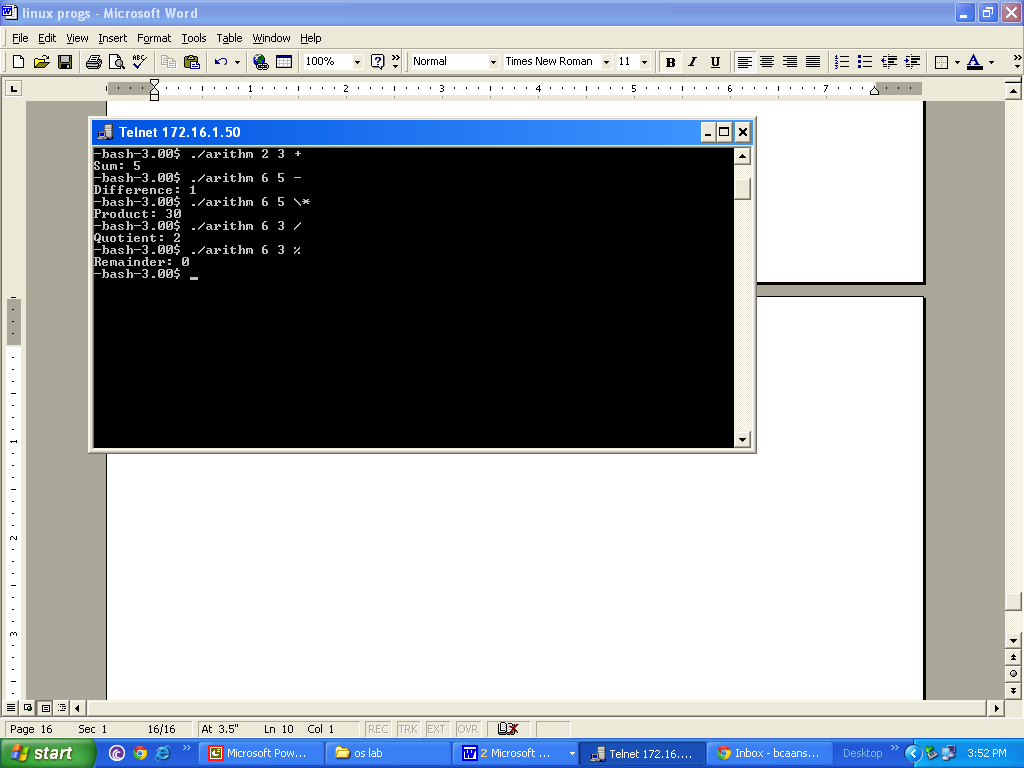
esac

else

echo "Invalid Number of arguments."

fi

* + **Output:**

****

1. **Write a script named “perform” using command line arguments to perform operation based on the option given. The syntax of the command is “perform <option><arg1><arg2>”**
2. **If the <option> is “–d”, then display the details of arg1 and arg2 if they are files.**
3. **If the <option> is “–l”, then display the count information such as line, word and character count in case of a file and content count in case of directory.**

**If the <option> is “-c”, then copy the contents of first argument to the second after checking for the existence of the first argument. If the second argument already exists replace the contents after getting the user confirmation.**

* + **Coding:**

case $1 in

-d)if test -f $2; then

echo "Contents of file $2"

cat $2

else

echo "File $2 not found"

fi

if test -f $3; then

echo "Contents of file $3"

cat $3

else

echo "File $3 not found"

fi;;

-l)if test -d $2; then

cd $2

ls>test

echo "No. of files: `wc -l test`"

else

echo "No. of characters: `wc -m $2`"

echo "No. of lines: `wc -l $2`"

echo "No. of words: `wc -w $2`"

fi

if test -d $3; then

cd $3

ls>test

echo "No. of files: `wc -l test`"

else

echo "No. of characters: `wc -m $3`"

echo "No. of lines: `wc -l $3`"

echo "No. of words: `wc -w $3`"

fi;;

-c)if test -f $2; then

if test -f $3; then

echo "File already exists. Do you want to continue?"

read ch

case $ch in

y) cp $2 $3;;

\*) echo "You are exited.";;

esac

else

cp $2 $3

fi

else

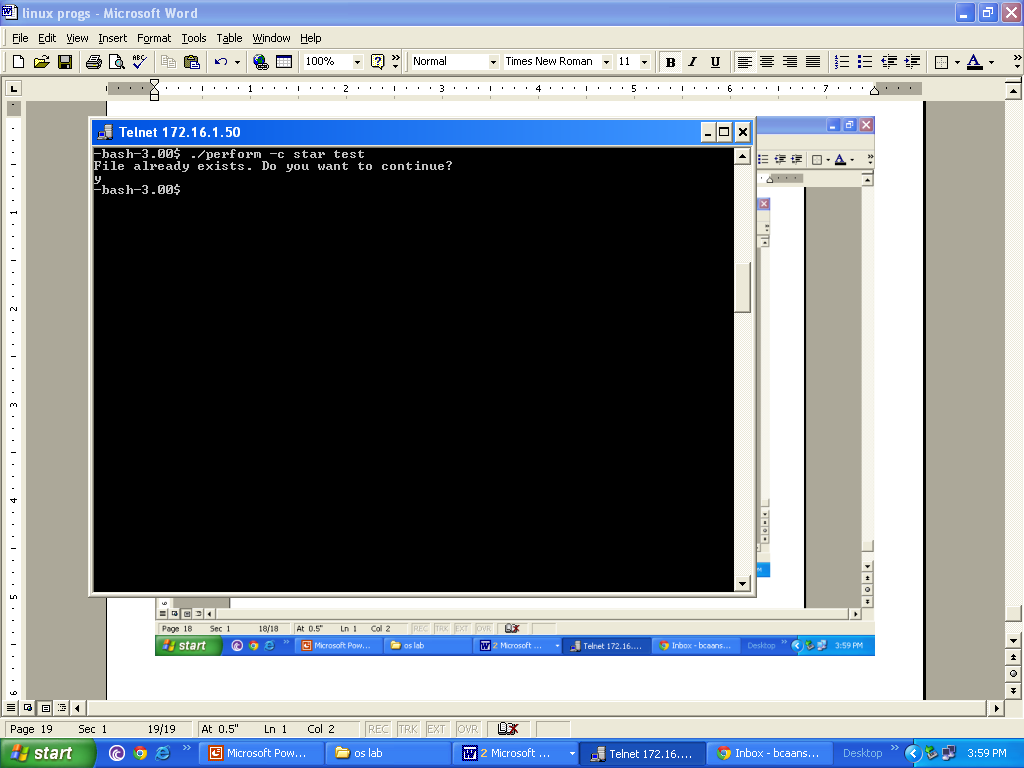
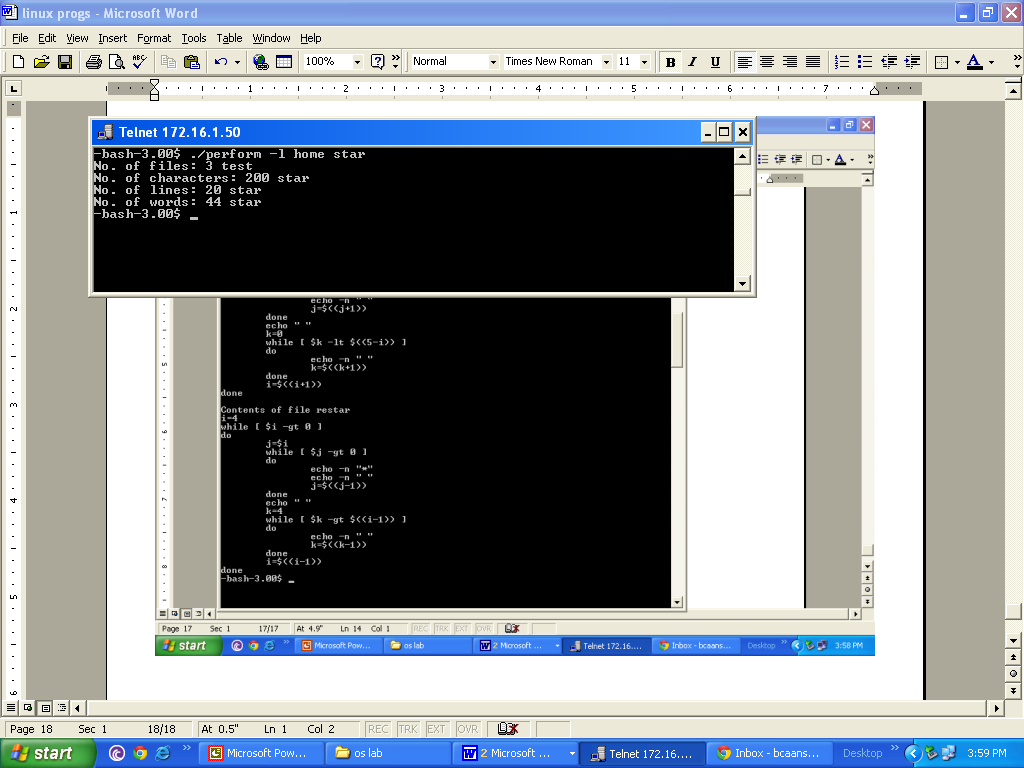
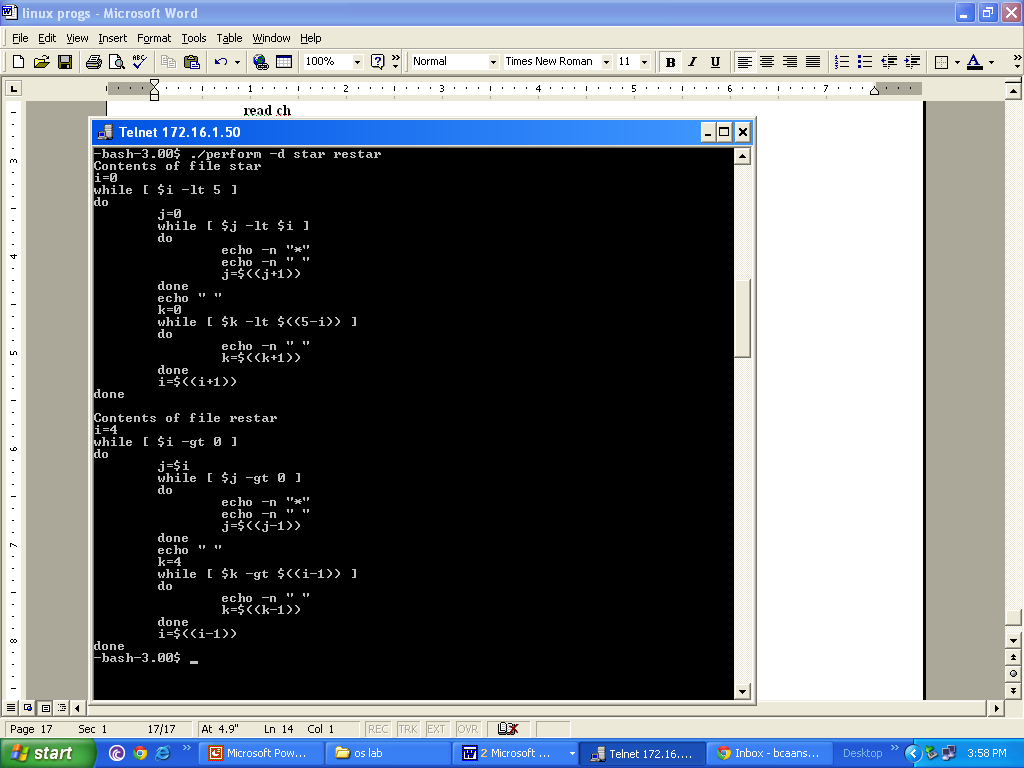
echo "File does not exists"

fi;;

\*)echo "Invalid argument";;

esac

**output:**

****

1. **Write a script to display type information of a directory contents.**
   * **Coding:**

for i in `ls`

do

if test -f $i

then

echo "$i is a file."

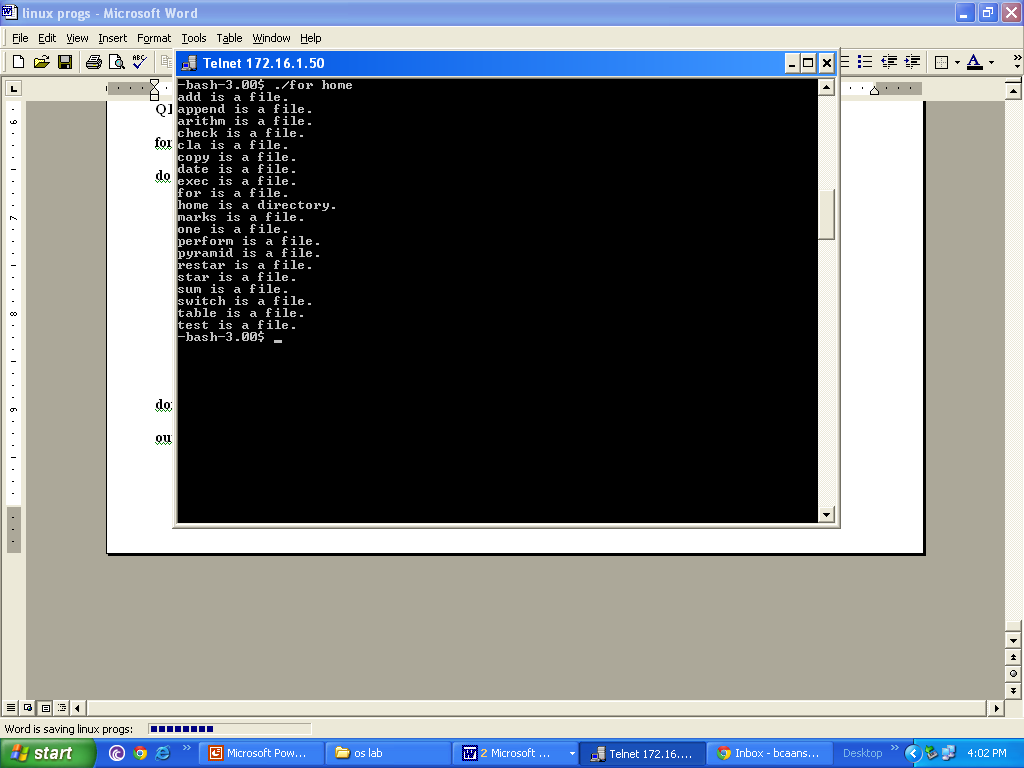
else

echo "$i is a directory."

fi

done

* + **Output:**

****

1. **Write a shell script to accept file names as command line arguments. Combine the contents of all the given files and store it along with the contents of the first file. Minimum 2 command line arguments are required.** 
   * **Coding:**

if test -f $1

then

file=$1

shift

for i in $\*

do

if test -f $i

then

echo "-----------------------" >> $file

cat $i>> $file

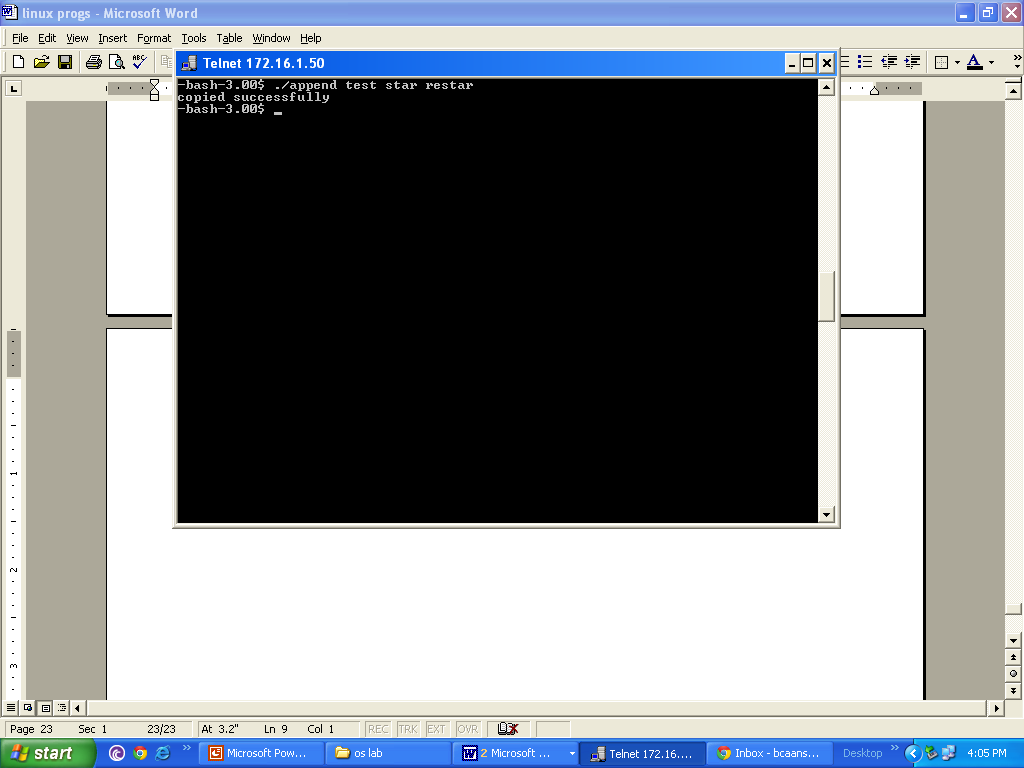
fi

done

echo "copied successfully"

fi

* + **Output:**

****

1. **Write a script to store system date in an array. Display the options given below and show the result based on the user’s choice.**
2. **Date**
3. **Month**
4. **Year**
5. **Time**
6. **Day**
   * **Coding:**

ar=(`date`)

ch='y'

while [ $ch == 'y' ]

do

echo –e "Enter Choice:\n1. Date\n2. Month\n3. Year\n4. Time\n5. Day"

read ch

case $ch in

1)echo ${ar[2]};;

2)echo ${ar[1]};;

3)echo ${ar[5]};;

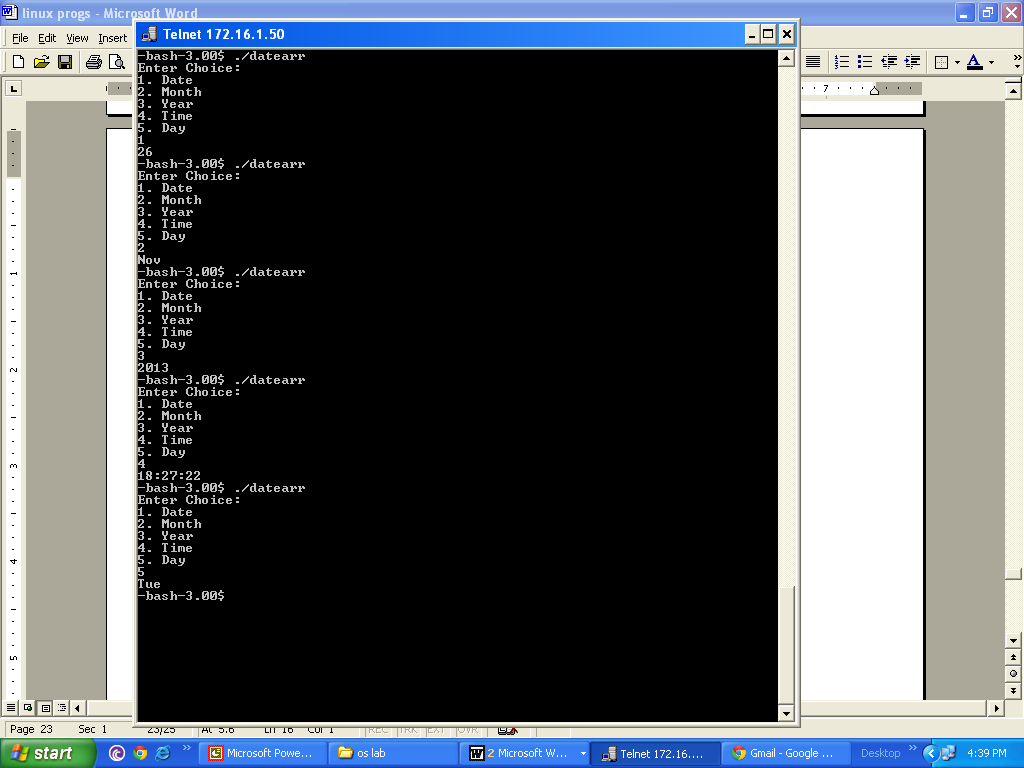
4)echo ${ar[3]};;

5)echo ${ar[0]};;

esac

done

* + **Output:**

****

1. **Write a script to display the contents of a directory in the order of their accessing time using arrays. The directory name has to be accepted from the user and its presence has to be checked.**
   * **Coding:**

echo “Enter the name of the directory”

read dir

if test –d $dir

then

ar=(`ls -t`)

fori in ${ar[\*]}

do

echo $i

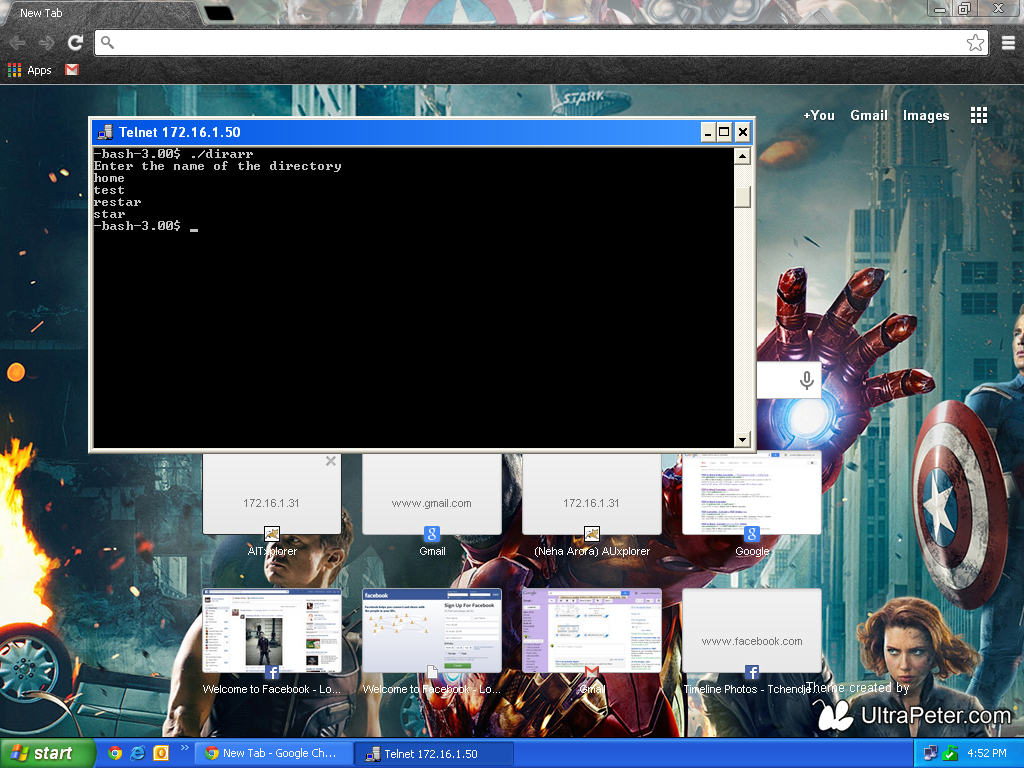
done

else

echo “No such Directory Found”

fi

* + **Output:**

****

1. **Write a shell script to display the contents of a directory in the format given below.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sl. No.** | **Type of File Name** | **File name** | **Length of File Name** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

* + **Coding:**

i=0

u=`ls`

echo "S.No. Type Name Length"

for j in ${u[\*]}

do

echo -n `expr $i + 1`

if test -f $j; then

echo -n " file $j"

else

echo -n " directory $j"

fi

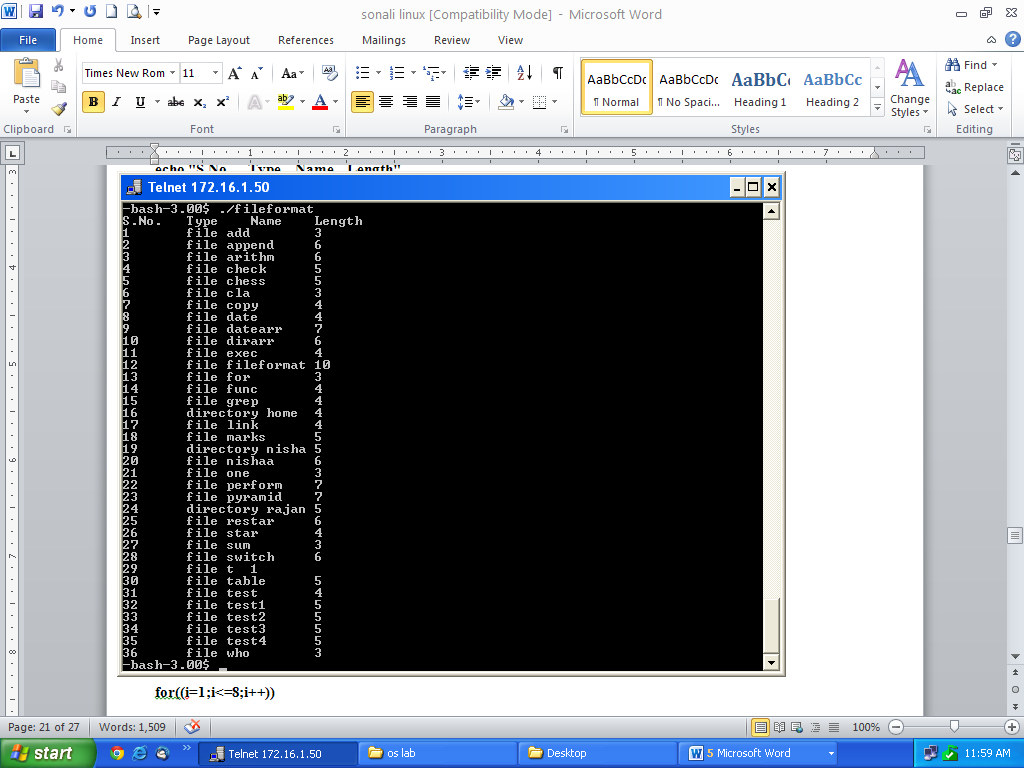
echo -n " "

echo "`expr length $j`"

i=$((i+1))

done

* + **Output:**



1. **Create a chessboard using shell scripts.**
   * **Coding**:

for((i=1;i<=8;i++))

do

for((j=1;j<=8;j++))

do

k=`expr $i + $j`

k=`expr $k % 2`

if test $k -eq 0

then

echo -e -n "\033[40m "

else

echo -e -n "\033[47m "

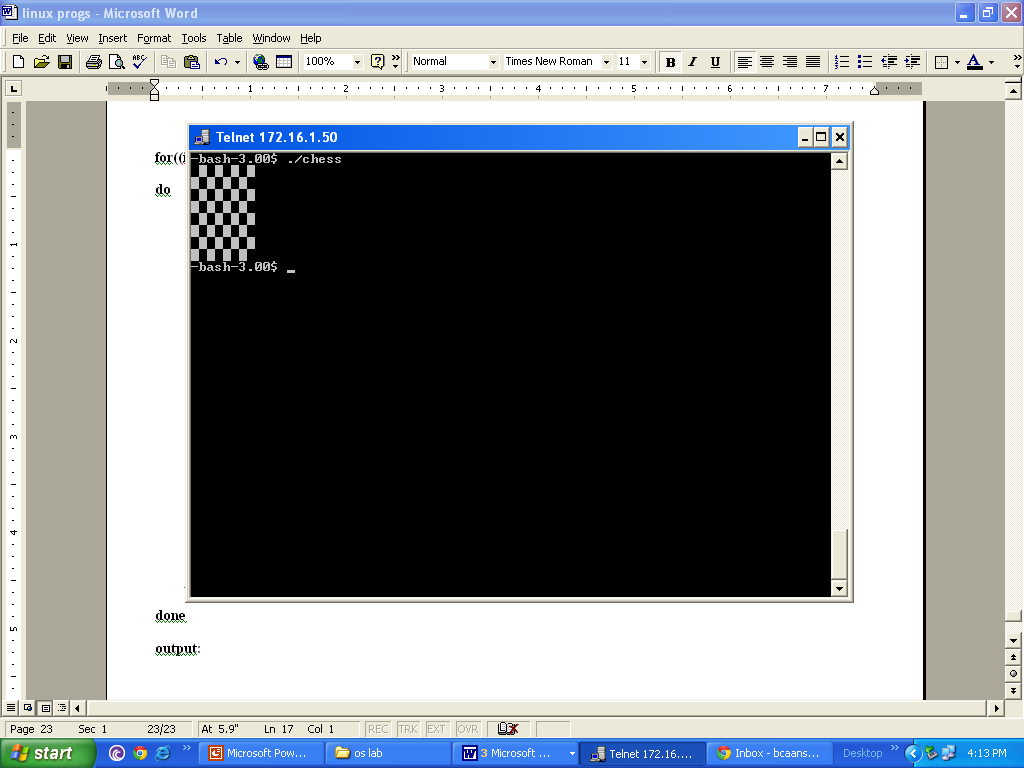
fi

done

echo -e "\033[40m "

done

* + **Output:**

****

1. **Create a shell script functions for copy, display, length calculation. Display menu to call the functions along with required parameters.**
   * **Coding:**

copyfn()

{if test -f $1; then

if test -f $2; then

echo "File Already Exists."

echo -e "Do you want to copy it?\n Enter y for yes and n for no"

read ch

if test $ch == 'y'; then

cat $1 > $2

echo "Successfully Copied."

else

echo -e "Content Not Copied.\nYou are exited."

fi

else

cat $1 > $2

echo "Successfully Copied."

fi

else

echo "File Does not Exists"

fi}

disp()

{echo "File Contents:"

cat $1}

len()

{echo "No. of words: `wc -w $1`"

echo "No. of characters: `wc -c $1`"

echo "No. of lines: `wc -l $1`"}

cho='y'

while [ $cho == 'y' ]

do

echo "Enter choice:"

echo "1. Copy"

echo "2. Display"

echo "3. Length calculation"

read num

case $num in

1)echo "Enter source file"

read file

echo "Enter destination file."

read file1

copyfn $file $file1;;

2)echo "Enter file name"

read file

disp $file;;

3)echo "Enter file name"

read file

len $file;;

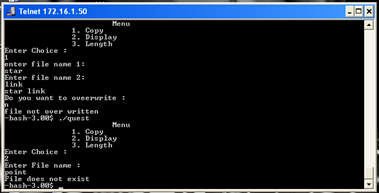
esac

echo "Do you want to continue?(y/n)"

read cho

done

* **Output:**



1. **Write a script to accept a string and a file name. Display the matching lines from the file.**
   * **Coding:**

echo "Enter the file name"

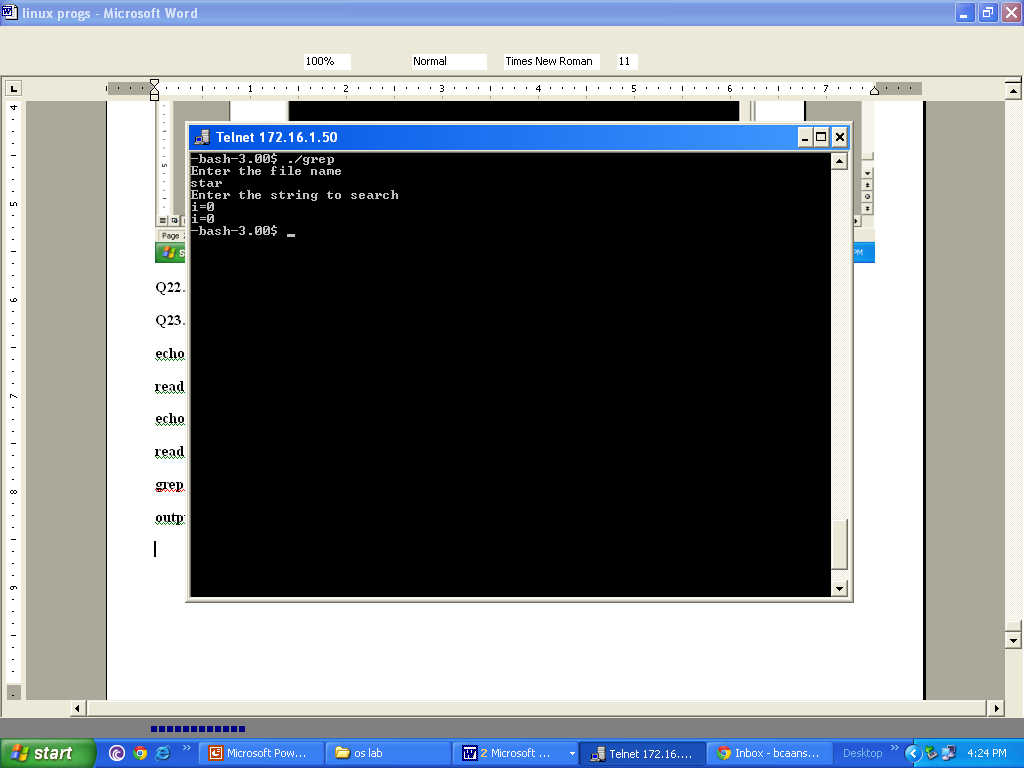
read file

echo "Enter the string to search"

read line

grep -x $line $file

* **Output:**

****

1. **Write a script to link two files accepted from the user. Link type should be the user’s choice. Show the proof of link establishment.**
   * **Coding:**

fl=0

echo "Enter first file name"

read file1

if test -f $file1; then

echo "Enter second file name"

read file2

if test -f $file2; then

echo "Do you want to link?(y/n)"

read ch

if test $ch == 'n'; then

fl=1

fi

fi

if test $fl -eq 0; then

echo "Select type of link:"

echo –e "1. Hardlink\n2. Symbolic Link"

read cho

if test $cho -eq 1; then

ln $file1 $file2

else

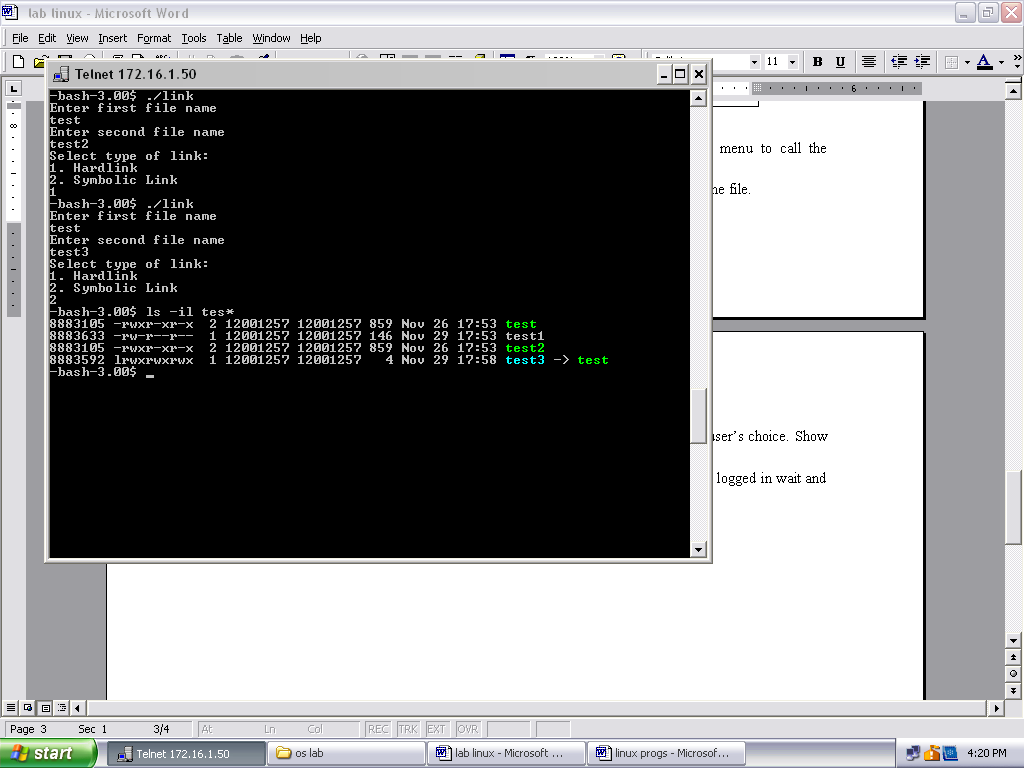
ln -s $file1 $file2

fi

fi

fi

* **Output:**

****

1. **Write a script to accept user name and greet him if he has logged in. if he has not logged in wait and greet him as soon as he logs into the system.**
   * **Coding:**

name=$1

until who | grep $name

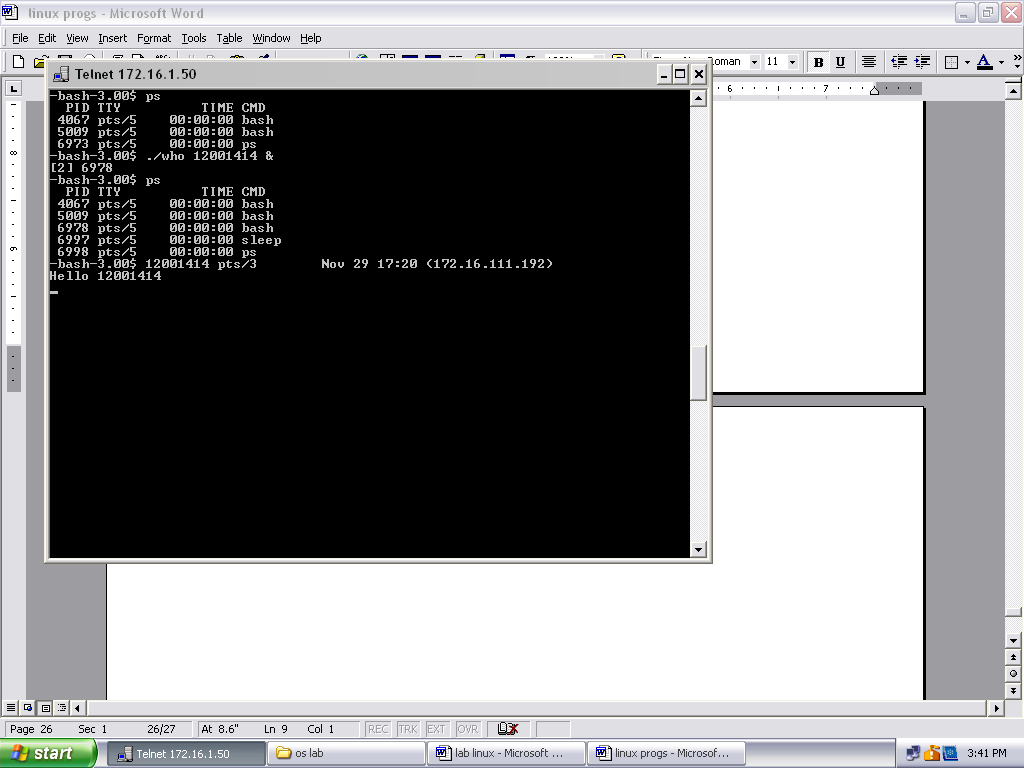
do

sleep 1

done

echo "Hello $name”

* **Output:**

****