PRESIDENCY COLLEGE

(AUTONOMOUS)

AFFILIATED TO BENGALURU CITY UNIVERSITY, APPROVED BY AICTE, DELHI & RECOGNISED BY THE GOVT. OF KARNATAKA

RE-ACCREDITED BY NAAC WITH 'A+' GRADE

OPERATING SYSTEM LABORATORY

LAB MANUAL

AS PER AUTONOMOUS SYLLUBUS

For ***MCA II Semester*** of **Presidency College**

Name of the Candidate _____ **Reg No.** _____

Course / Subject code _____

Written by

Dr Veera N M

Department of Computer Applications

1. Write a shell script that accepts a path name and creates all the components in that path name as directories. For example, if the script is 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 number of arguments”
else
        mkdir -p $1
fi
```

Execution and Output:

$ sh first.sh Presidency/College/Kempapura/Bangalore

$ ls

Presidency

$ cd Presidency

$ ls

College

. . .

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

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

else
        ls -l $1 | cut -d ' ' -f1 > file1
        ls -l $2 | cut -d ' ' -f1 > file2

        if cmp file1 file2
        then
                echo  " \n Both files have same permissions \n"
                cat file1

        else

                echo  " \n \n Files have different permissions \n"
                echo The permissions of file $1
                cat file1
                echo The permissions of file $2
                cat file2
        fi
fi
```

Execution and Output:

$ sh second.sh ext1.txt example.sh # ext1.txt and example.sh are the files in the current directory
file1 file2 differ: byte 4, line 1
Files have different permissions
The permissions of file ex1.txt
-rwxrw-r--
The permissions of file example.sh
-rw-rw-r--
$ sh second.sh
Invalid number of arguments

3. Write a shell script which accepts valid log- in names as arguments and prints their corresponding home directories, if no arguments are specified, print a suitable error message.

```
clear
if [ $# -eq 0 ]
then
        echo “No command line arguments 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 log-in name
        else
                echo Home Directory for $1 is:
                cat /etc/passwd | grep "$1" | cut -d ':' -f6
        fi
        shift
done
```

Execution and Output:

$ sh third.sh
No command line arguments passed
$ sh third.sh veera
Home Directory for veera is:
/home/veera

4. Create a script file called file-properties that reads a file name entered and outputs it properties.

```
clear
echo Enter the filename:\c
    read fn

    if [ -f $fn ]
    then
        echo File permissions are
        echo `ls -l $fn | cut -d ' ' -f1`

        echo Number of links to the file
        echo `ls -l $fn | cut -d ' ' -f2`

        echo File size
        echo `ls -l $fn | cut -d ' ' -f5`

        echo  Last modified Month
        echo `ls -l $fn | cut -d ' ' -f6`
        echo  Last modified  Date
        echo `ls -l $fn | cut -d ' ' -f7`

    else
        echo File not found
    fi
```

Execution and Output:

```
$ sh fourth.sh
  Enter the filename:
  first.sh
  File permissions are
   -rw-rw-r--
  Number of links to the file
  1
 File size
 334
 Last modified Month
 Jun
 Last modified Date
 15
```

5.Write a shell script that accept one or more file names as argument and convert all of them to uppercase, provided they exist in current directory.

```
#! /bin/bash
clear
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
        fi
 done
```

Execution and Output:

```
$ sh fifth.sh
  Invalid number of arguments
$ sh fifth.sh  example.sh  first.sh  fifth.sh
  EXAMPLE.SH
  FIRST.SH
  FIFTH.SH
```

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

```
#! /bin/bash
 clear
 if [ $# -ne 1 ]
 then
        echo Invalid number of arguments
         exit
 fi
 if [ -e $1 ]
 then
        echo File $1 is created on :`ls -l | tr -s " " | cut -d " " -f6,7,8`
 else
        echo  File not found
 fi
```

Execution and Output:

```
$ sh sixth.sh
  Invalid number of arguments
$ sh sixth.sh  sixth.sh
File sixth.sh is created on :
Jul 12 09:13
```

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

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

Execution and Output:

sh seventh.sh

June 2022

Su	Mo	Tu	We	Th	Fr	Sa
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
**	27	28	29	30		

8. Write a shell script to list all the files in a directory whose filename is at least 10 characters. (use expr command to check the length)

```
#! /bin/bash
 clear
        echo `ls > listfiles`
         echo the file with characters greater than 9
        for i in `cat listfiles`
        do
            len=`expr length $i`

            if [ $len -gt 9 ]
            then
                  echo $i
            else
            continue
            fi
        done
```

Execution and Output:

$ sh eighth.sh

The files with characters greater than 9 are:
arithmeticcalculations.txt 26
calender.txt 12
example.sh 10
seventh.sh 10
working.txt 11

9. Write a shell script that gets executed displays the message either “Good Morning” or “Good Afternoon” or “Good Evening” depending upon time at which the user logs in.

```
#! bin/bin/sh
        clear
          h=`who | head -1 | tr -s ' ' | cut -d ' ' -f4 | cut -d ':' -f1`

         if [ $h -lt 12 ]
         then
                 echo  Good Morning

         elif [ $h -ge 12 -a $h -lt 17 ]
         then
                 echo  Good Afternoon
         else
                 echo  Good Evening
         fi
```

Execution and Output:

```
$ sh ninth.sh
  Good Evening
```

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

```
         clear
          if [ $# -lt 2 ]
         then
                 echo  invalid arguments
                 exit
         fi
         for word in `cat $1`
         do
                 for file in $*
                 do
                         if [ $file != "$1" ]
                         then
                         echo   count :$word in $file
                         echo   `grep -iow "$word" $file | wc -w`
```

```
                fi
        done
done
```

Execution and Output:

```
$ sh tenth.sh filename1 filename2
count :Ram in filename2.txt    2
count :BTech in filename2.txt   0
count :Raheem in filename2.txt 0
count :M in filename2.txt 0
count :Tech in filename2.txt 0
count :Joseph in filename2.txt  2
count :PhD in filename2.txt  1
```

11. Write a shell script that accept the filename, starting and ending line number as an argument and display all the lines between the given line number.

```
clear
if [ $# -ne 3 ]
then
        echo "Invalid number of arguments"
        exit
fi

c=`cat $1 | wc -l`

if [ $2 -le 0 -o $3 -le 0 -o $2 -gt $3 -o $3 -gt $c ]
then
        echo "Invalid Input" exit
fi

sed -n "$2, $3 p" $1
```

Execution and Output:

```
$ sh eleventh.sh  file1  2 5
```

content of the file1 between 2nd and 5th line should be displayed

12. Write an awkscript that accepts date argument in the form of dd-mm-# yyyy and displays it in the form month, day and year. The script should # check the validity of the argument and in the case of error, display # a suitable message.

```
BEGIN {
        FS="-";
        printf " Day      Month       Year\n";
}
```

```
    {
        printf " \n The date is %d-%d-%d\n ", $1,$2,$3;

        if (( $1 >=1 && $1 <=31) && ($2 >=1 && $2 <=12)){
        printf "The date is valid\t:";

        if($2==1)
                mon="Jan";
        else if($2==2)
                mon="Feb";
        else if($2==3)
                mon="Mar";
        else if($2==4)
                mon="Apr";
        else if($2==5)
                mon="May";
        else if($2==6)
                mon="Jun";
        else if($2==7)
                mon="Jul";
        else if($2==8)
                mon="Aug";
        else if($2==9)
                mon="Sep";
        else if($2==10)
                mon="Oct";
        else if($2==11)
                mon="Nov";
        else if($2==12)
                mon="Dec";

        printf "Date is %s-%d-%d", mon,$1,$3;
     }
  else
     printf "date is invalid\n";
     system ("exit");

}
```

Execution and Output:

```
$ cat dates.data
14-08-1969
39-06-1973
18-04-2004
05-08-2006
17-08-1969
32-11-2010
```

```
$ awk -f twelveth.awk dates.data
Day      Month       Year
 The date is 14-8-1969
 The date is valid       :Date is Aug-14-1969
 The date is 39-6-1973
 date is invalid
 The date is 18-4-2004
 The date is valid       :Date is Apr-18-2004
 The date is 5-8-2006
 The date is valid       :Date is Aug-5-2006
 The date is 17-8-1969
 The date is valid       :Date is Aug-17-1969
 The date is 32-11-2010
 date is invalid
```

13. Write an awk script to delete duplicated lines from a text file. The order of the original lines must remain unchanged.

```
 BEGIN {
        printf " Program starts\n" ;
 }


   {
        if ( data[$0]++ == 0 )

          line[++count]=$0;
   }


 END {
        for (i=1; i<= count; i++)
           printf "\n" line[i];

           printf " \n\n program ends";
           printf "\n"
 }
```

Execution and Output:

Input file: file with repeated lines say --- duplicates.txt

```
$ awk -f thirteen.awk duplicqtelines.txt
 Program starts
 --------------
unique lines
---------------
```

14 Write an awkscript to find out total number of books sold in each discipline as well as total books sold, using associate array like table as given below.

a. Computer science 34
b. Commerce 67
c. Management 80
d. Journalism 43

```
BEGIN {
        printf " Total number of books each category\n";
}


{
        b[$1] += $2;
}

END {
        for(item in b) {
                printf "%s %s %d \n", item , "=", b[item];
                total += b[item];
        }

        printf "%s %s %s \n", "total books", "=",total;
}
```

Execution and Output:

```
$ cat Books.data
ComputerScience 34
 Commerce 67
 Management 80
 Journalism 43
 ComputerScience 45
 Commerce 10
 Management 40
 Journalism 20
 Commerce 100
$ awk -f fifteenth.awk Books.data
Commerce 67
  Management 80
  Journalism 43
  ComputerScience 45
  Commerce 10
  Management 40
  Journalism 20
  Commerce 100
```

16. Write a program to copy a file into another using system calls.

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

void main() {
        char buf;
        int fd_one, fd_two;

        fd_one = open("bigfile.txt", O_RDONLY);

        if (fd_one == -1) {
                printf("Error opening first_file\n");
                close(fd_one);
                return;
        }

        fd_two = open("second_file",
                                O_WRONLY | O_CREAT,
                                S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH);

        while(read(fd_one, &buf, 1)) {
                write(fd_two, &buf, 1);
        }

        printf("\nSuccessful copy \n");

        close(fd_one);
        close(fd_two);
 }
```

Execution and Output:

```
$ cc  sixteen.c
$ ./a.out
Successful copy
```

17. Write a program using system call: create, open, write, close, stat, fstat, lseek.

```
 C Program for open() system call
#include<stdio.h>
#include<fcntl.h>
#include<errno.h>

extern int errno;
int main() {
          // if file does not have in directory
          // then file foo.txt is created.
          int fd = open("exposure.txt", O_RDONLY | O_CREAT);
```

```
        printf("fd = %d \n", fd);

        if (fd == -1) {
        // print which type of error have in a code
        printf("\n Error Number % d\n", errno);

        // print program detail "Success or failure"
        perror("Program");
        }
        return 0;
}
```

Execution and Output:

```
$ cc seventeen_open.c
$ ./a.out
fd = -1
 Error Number  13
Program: Permission denied
```

C program to illustrate
// read system Call

```
#include<stdio.h>
#include <fcntl.h>
int main() {
int fd, sz;
char *c = (char *) calloc(100, sizeof(char));
fd = open("foo.txt", O_RDONLY);
if (fd < 0) {
perror("r1"); exit(1); }
sz = read(fd, c, 10);
printf("called read(% d, c, 10). returned that" " %d bytes were read.\n", fd, sz);
c[sz] = '\0';
printf("Those bytes are as follows: % s\n", c);
}
```

Execution and output:

```
$ cc seventeen_read.c
$ ./a.out
called read( 3, c, 10). returned that 10 bytes were read.
Those bytes are as follows: // C progr
```

// C program to illustrate write system Call

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

```
#include <string.h>
#include <unistd.h>
int main() {
        int sz;

        int fd = open("example_write.txt", O_WRONLY | O_CREAT | O_TRUNC, 0664);

        printf("fd = %d \n", fd);

        if (fd < 0) {
                perror("r1");
                exit(1);
                }
        sz = write(fd, "Hello Presidentians", strlen("Hello Presidentians"));

        printf("called write(%d, \" Hello Presidentians\\n \")",fd);
        printf("\nIt returned %d\n", sz);
        printf("String length of %ld \n", strlen("Hello Presidentians\n"));

        close(fd);
        }
```

Execution and Output:

```
$ cc seventeen_write.c
$ ./a.out
fd = 3
called write(3, " Hello Presidentians\n ")
It returned 19
String length of 20
```

```
// C program to illustrate close system Call
#include<stdio.h>
#include <fcntl.h>
int main() {
  int fd1 = open("file1.txt", O_RDONLY);
        if (fd1 < 0) {
        perror("c1");
        exit(1);
       }
        printf("opened the fd = % d\n", fd1);
        // Using close system Call
        if (close(fd1) < 0) {
                perror("c1");
                exit(1);
        }
        printf("closed the fd.\n");
}
```

Exectution and Output:

```
$ cc seventeen_close.c
$ ./a.out
opened the fd =  3
closed the fd.
```

/*Program using lseek() system call that reads 10 characters from file “file1” and print on screen. Again read 10 characters and write on screen.
*/

```
#include<unistd.h>
#include<fcntl.h>
#include<sys/types.h>
#include<sys/stat.h>
#include <stdio.h>

int main() {
        int n,f;
        char buff[10];

        f=open("file1",O_RDWR);
        read(f,buff,10);
        write(1,buff,10);
        read(f,buff,10);
        write(1,buff,10);
        printf("\n");
}
```

Execution and Output:

```
$ cc seventeen_lseek.c
$ ./a.out
Hello II semester MC   // 20 characters from file1
```

18. Write a program to create a child process and allow the parent to display “parent” and the child to display “child” on the screen.

```
#include <stdio.h>
#include <sys/wait.h> /* contains prototype for wait */
int main(void) {
        int pid;
        int status;
        printf("Hello World!\n");
        pid = fork( );
        if(pid == -1) {       /* check for error in fork */
                perror("bad fork");
```

```
        exit(1);
    }
    if (pid == 0)
        printf("I am the child process.\n");
    else {
        wait(&status); /* parent waits for child to finish */
        printf("I am the parent process.\n");
    }
}
```

Execution and Output:

```
$ cc parent_child.c
$ ./a.out
Hello World!
I am the child process.
I am the parent process.
```

19. Write a program to create a Zombie process.

```
#include <stdio.h>
#include <stdlib.h> // for exit()
#include <unistd.h> // for fork(), and sleep()

int main() {
// Creating a Child Process
    int pid = fork();
    if (pid > 0) // True for Parent Process
    sleep(60);
    else if (pid == 0)  {     // True for Child Process

        printf("Zombie Process Created Successfully!");
        exit(0);
    }
    else // True when Child Process creation fails
        printf("Sorry! Child Process cannot be created...");
        return 0;
}
```

Execution and Output:

```
$ cc zombie_prg.c
$ ./a.out
Zombie Process Created Successfully!
```

//20. Simulate the following CPU scheduling algorithms

// a. Round Robin

```
#include<stdio.h>

int main() {
	int i, limit, total = 0, x, counter = 0, time_quantum;
	int wait_time = 0, turnaround_time = 0, arrival_time[10],
	burst_time[10], temp[10];
	float average_wait_time, average_turnaround_time;

	printf("\nEnter Total Number of Processes:\t");
	scanf("%d", &limit);

	x = limit;
	for(i = 0; i < limit; i++) {
		printf("\nEnter Details of Process[%d]\n", i + 1);
		printf("Arrival Time:\t");
		scanf("%d", &arrival_time[i]);
		printf("Burst Time:\t");
		scanf("%d", &burst_time[i]);
		temp[i] = burst_time[i];
	}
	printf("\nEnter Time Quantum:\t");
	scanf("%d", &time_quantum);
	printf("\nProcess ID\t\tBurst Time\t Turnaround Time\t Waiting Time\n");

	for(total = 0, i = 0; x != 0;) {
		if(temp[i] <= time_quantum && temp[i] > 0) {
		total = total + temp[i];
		temp[i] = 0;
		counter = 1;
	}
	else
		if(temp[i] > 0) {
		temp[i] = temp[i] - time_quantum;
		total = total + time_quantum;
	}
	if(temp[i] == 0 && counter == 1) {
		x--;
		printf("\nProcess[%d]\t\t%d\t\t %d\t\t\t %d", i + 1, burst_time[i],
		total - arrival_time[i], total - arrival_time[i] - burst_time[i]);
		wait_time = wait_time + total - arrival_time[i] - burst_time[i];
		turnaround_time = turnaround_time + total - arrival_time[i];
		counter = 0;
	}
	else if(i == limit – 1) {
		i = 0;
	}
	else if(arrival_time[i + 1] <= total) {
		i++;
	}
	else {
		i = 0;
```

```
        }
    }
        average_wait_time = wait_time * 1.0 / limit;
        average_turnaround_time = turnaround_time * 1.0 / limit;
        printf("\n\nAverage Waiting Time:\t%f", average_wait_time);
        printf("\nAverage Turnaround Time:\t%f\n", average_turnaround_time);
        return 0;
}
```

Execution and Output:

```
Enter Total Number of Processes: 5
Enter Details of Process[1]
Arrival Time: 0
Burst Time: 3
Enter Details of Process[2]
Arrival Time: 1
Burst Time: 7
Enter Details of Process[3]
Arrival Time: 2
Burst Time: 6
Enter Details of Process[4]
Arrival Time: 3
Burst Time: 2
Enter Details of Process[5]
Arrival Time: 4
Burst Time: 7
Enter Time Quantum: 3
Process ID Burst Time Turnaround Time Waiting Time
Process[1] 3 3 0
Process[4] 2 8 6
Process[3] 6 18 12
Process[2] 7 23 16
Process[5] 7 21 14
Average Waiting Time: 9.600000
Average Turnaround Time: 14.600000
```

//20. (b)shortest Job first Program using Non-Preemptive

```
#include<stdio.h>

int main() {
        int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,temp;
        float avg_wt,avg_tat;
        printf("Enter number of process:");
        scanf("%d",&n);
        printf("\nEnter Burst Time:\n");
        for(i=0;i<n;i++) {
                printf("p%d:",i+1);
                scanf("%d",&bt[i]);
                p[i]=i+1;
        }
        //sorting of burst times
```

```
        for(i=0;i<n;i++) {
                pos=i;
                for(j=i+1;j<n;j++) {
                        if(bt[j]<bt[pos])
                        pos=j;
                }
        temp=bt[i];
        bt[i]=bt[pos];
        bt[pos]=temp;
        temp=p[i];
        p[i]=p[pos];
        p[pos]=temp;
        }
        wt[0]=0;
        for(i=1;i<n;i++) {
                wt[i]=0;
                for(j=0;j<i;j++)
                wt[i]+=bt[j];
                total+=wt[i];
        }
        avg_wt=(float)total/n;
        total=0;
        printf("\nProcesst Burst Time \tWaiting TimetTurnaround Time");
        for(i=0;i<n;i++) {
                tat[i]=bt[i]+wt[i];
                total+=tat[i];
                printf("\np%d\t\t %d\t\t %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);
        }
        avg_tat=(float)total/n;
        printf("\n\nAverage Waiting Time=%f",avg_wt);
        printf("\nAverage Turnaround Time=%f\n",avg_tat);
}
```

Execution and Output:

```
Enter number of process:5
Enter Burst Time:
p1:3
p2:6
p3:2
p4:7
p5:4
Processt Burst Time Waiting TimetTurnaround Time
p3 2 0 2
p1 3 2 5
p5 4 5 9
p2 6 9 15
p4 7 15 22
Average Waiting Time=6.200000
```

Average Turnaround Time=10.600000

//20. (c) shortest Job first Program using Pre-emptive

```
#include <stdio.h>
int main() {
        int arrival_time[10], burst_time[10], temp[10];
        int i, smallest, count = 0, time, limit;
        double wait_time = 0, turnaround_time = 0, end;
        float average_waiting_time, average_turnaround_time;
        printf("\nEnter the Total Number of Processes:\t");
        scanf("%d", &limit);
        printf("\nEnter Details of %d Processes\n", limit);
        for(i = 0; i < limit; i++) {
                printf("\nEnter Arrival Time:\t");
                scanf("%d", &arrival_time[i]);
                printf("Enter Burst Time:\t");
                scanf("%d", &burst_time[i]);
                temp[i] = burst_time[i];
        }
        burst_time[9] = 9999;
        for(time = 0; count != limit; time++) {
        smallest = 9;
                for(i = 0; i < limit; i++) {
                        if(arrival_time[i] <= time && burst_time[i] < burst_time[smallest]
                                && burst_time[i] > 0) {
                                smallest = i;
                }
        }
        burst_time[smallest]--;
        if(burst_time[smallest] == 0) {
                count++;
                end = time + 1;
                wait_time = wait_time + end - arrival_time[smallest] - temp[smallest];
                turnaround_time = turnaround_time + end - arrival_time[smallest];
        }
    }
        average_waiting_time = wait_time / limit;
        average_turnaround_time = turnaround_time / limit;
        printf("\n\nAverage Waiting Time:\t%lf\n", average_waiting_time);
        printf("Average Turnaround Time:\t%lf\n", average_turnaround_time);
        return 0;
}
```

Execution and Output:

```
Enter the Total Number of Processes: 5
Enter Details of 5 Processes
Enter Arrival Time: 1
Enter Burst Time: 3
Enter Arrival Time: 2
```

Enter Burst Time: 6
Enter Arrival Time: 3
Enter Burst Time: 4
Enter Arrival Time: 4
Enter Burst Time: 7
Enter Arrival Time: 5
Enter Burst Time: 2
Average Waiting Time: 4.600000
Average Turnaround Time: 9.000000

15. **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=20% of basic & DA=50% of basic.**

```
BEGIN{
printf"enter the basic salary:Rs"
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=Rs.%.2f\n",gs
}
```